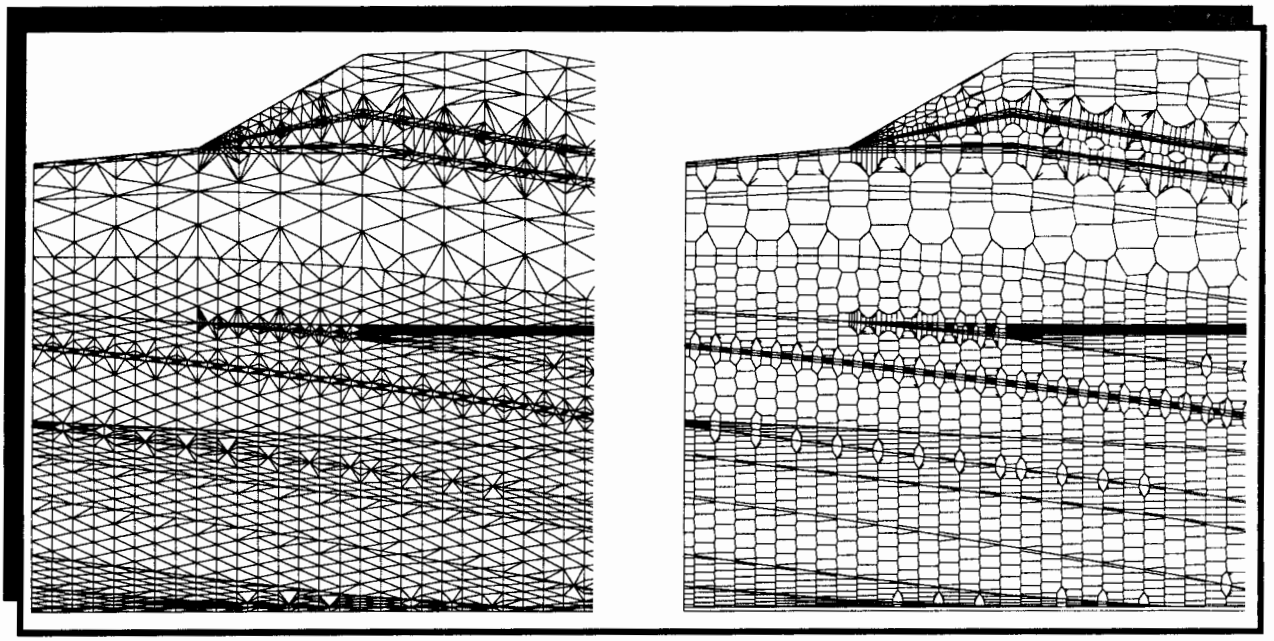
LA-UR-95-4143

Yucca Mountain Site Characterization Project Milestone 4075: Letter Report
December 8, 1995

GEOMESH GRID GENERATION



GEOMESH PROJECT DESCRIPTION

GEOMESH USER'S MANUAL

GEOMESH PROJECT WORKLIST

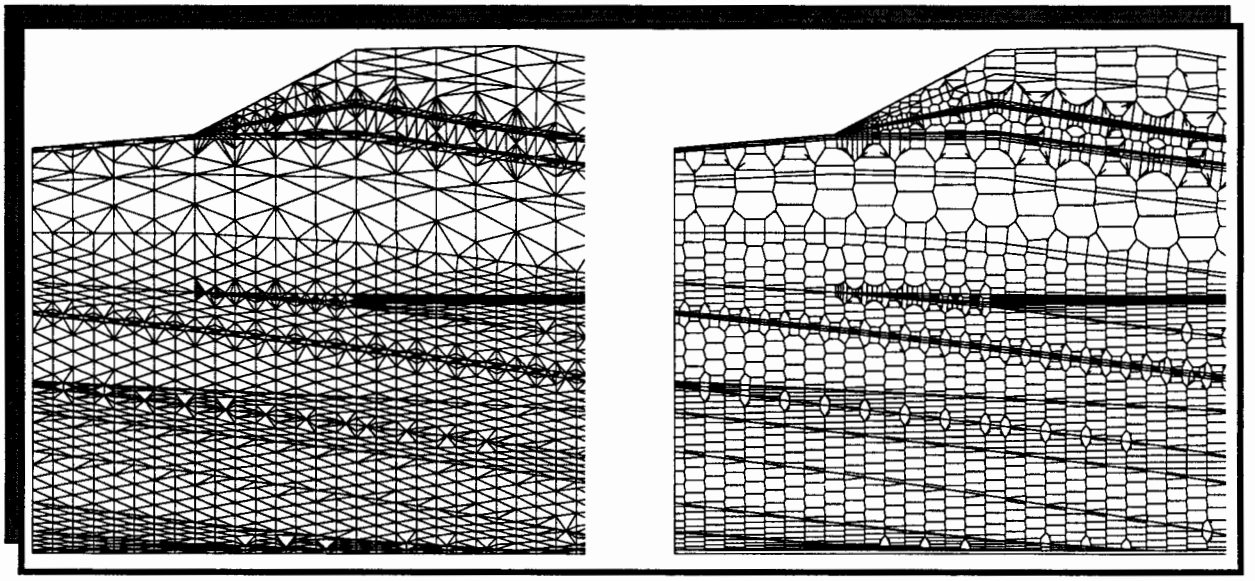
X3D USER'S MANUAL

Los Alamos National Laboratory
Earth and Environmental Science Division
Geoanalysis Group, EES-5

Carl W. Gable
Terry Cherry
Harold Trease
George A. Zyvoloski

LA-UR-95-4143

GEOMESH GRID GENERATION



GEOMESH PROJECT DESCRIPTION

Los Alamos National Laboratory
Earth and Environmental Science Division
Geoanalysis Group, EES-5
Carl W. Gable
Terry Cherry
Harold Trease
George A. Zyvoloski

TABLE of CONTENTS

11.0 DOCUMENT PURPOSE

This document describes the GEOMESH software project. It explains the goals, organization, current capabilities and future plans for the GEOMESH project. A catalog of grids is included to present sample grid uses and capabilities. For further information, refer to the following GEOMESH documents:

- *GEOMESH Project Description* - overview of capabilities, plans, and grid applications.
- *User's Manual for GEOMESH* - how to create computational grids.
- *X3D User's Manual* - how to run the command language for mesh object commands.

Other documents maintained for the GEOMESH projects include:
- *GEOMESH Handout* - a short, summarized version of the Project Description.
- *GEOMESH Programmer's Handbook* - code development details for GEOMESH.
- *GEOMESH Worklist* - running record of all work done for the GEOMESH project.

1.0 DEFINITIONS AND ACRONYMS

AVS - Advanced Visual Systems graphical 3D visualization tool

CMO - Current Mesh Object, code structure defining a mesh

Delaunay - Triangulation of a point distribution

FEHM - Finite element heat and mass transfer code (Zyvoloski, et al. 1988)

FEHMN - YMP version of FEHM (Zyvoloski, et al. 1992)

GEOMESH - Finite element grid generation project for geologic applications

GMV - General Mesh Viewer, a 3D visualization tool for meshes

GMD - Geologic Mesh Data - any possible input into GEOMESH grid generation

GMO - Geologic Mesh Object - GEOMESH version of X3D CMO

LANL - Los Alamos National Laboratory

TINS - Triangulated Irregular Network for a surface representation

YMP - Yucca Mountain Site Characterization Project

UCD - Unstructured Cell Data, AVS data type for finite element meshes

X3D - Grid generation command language for physics applications

2.0 GEOMESH OVERVIEW

GEOMESH began from a need for accurate and automated grid generation for 3D site scale modeling of flow and transport. Since the grids represent the geology being modeled, the accuracy of the grid directly affects the accuracy of the model. It was also found that grid generation was tedious, time consuming, and prone to errors, especially for models with complex structures such as faults, pinch outs and layer truncations.

Automated grid generation algorithms not only streamline work, they offer more flexibility and more consistency. As input constraints change and as better data sets become available, it is not difficult to incorporate the new changes to the computational mesh. Automated grid generation can also be used to produce coarsened grids for preliminary calculations and refined grids for final, high resolution calculations. The grids produced by GEOMESH are not specific to any particular computational flow and transport code. The grids can be used by any numerical algorithm that can utilize unstructured grids.

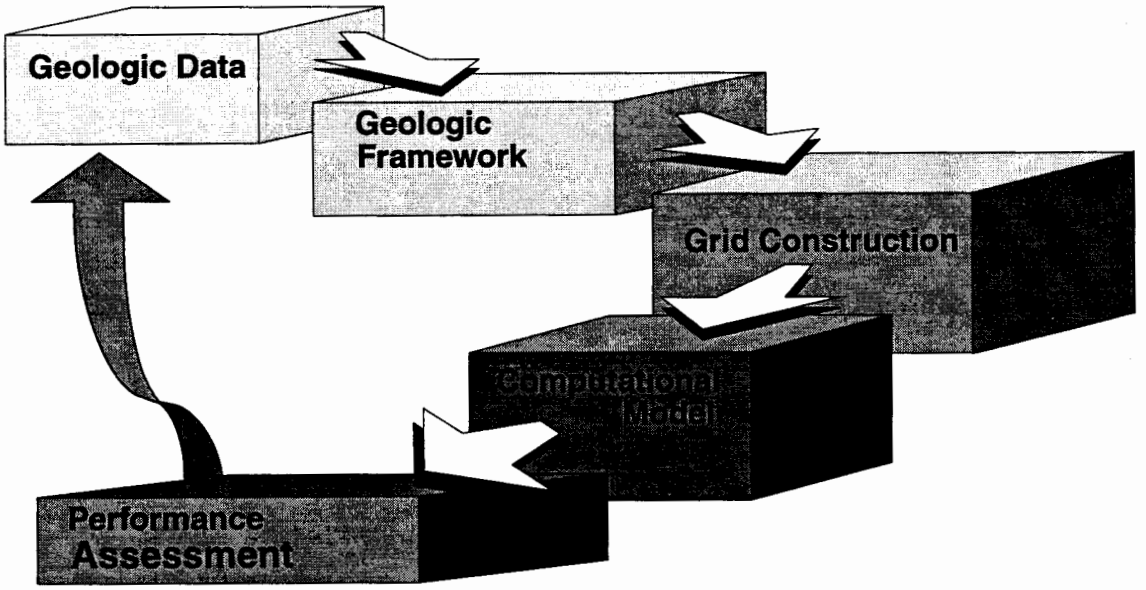
The present GEOMESH project provides an automated finite element grid generation package that is tuned to the special needs of geologic and geo-engineering applications. The code provides for 2D and 3D grids and includes elements that are triangles, quadrilateral, hexahedral and tetrahedral. Capabilities have been expanded to improve grid generation for complex geometries and for providing more choices in designing a grid. At the same time, old routines have been tested and solutions found for degenerate cases. The design of the code is modular, allowing for flexibility and consistency. The project team is now well practiced at generating useful code and looks forward to implementing new capabilities.

Though future development in GEOMESH code needs to provide an intuitive and easy to use interface and some needed utility, emphasis for the coming year will be on documentation updates instead of code development. The capabilities and future plans listed in this document are current as of this document's date.

3.0 GOALS of the GEOMESH PROJECT

The goal of GEOMESH is to produce computational grids which strictly honor the Geo Mesh Data and to produce grids which are optimized for computations. GEOMESH differs from other grid generators because it has been designed to meet needs specific to geologic settings such as faulting, beds which pinch out, irregular external surfaces defined by topography, and the representation of wells and tunnels. There are a number of commercial products that aid in building three dimensional representations of geologic structures (Lynx, Dynamic Graphics, Stratamodel) however, these products do not prepare output which is usable by finite element and finite difference flow and transport modeling programs. There are products which will convert stratigraphic models to computational grids, however these computational grids do not strictly honor the geometric integrity of the original model, GEOMESH does.

4.0 OVERVIEW of INTEGRATED MODELING



GEOMESH approaches grid generation as part of an integrated modeling approach. Automated grid generation provides a link between a geologic framework and a computational model. It can also insure that constraints on grid discretization are optimized to insure the accuracy and stability of numerical solutions.

In general, the steps from a geologic data to a computational model are:

1) **Geologic Data** is gathered and stored as computer accesible files.

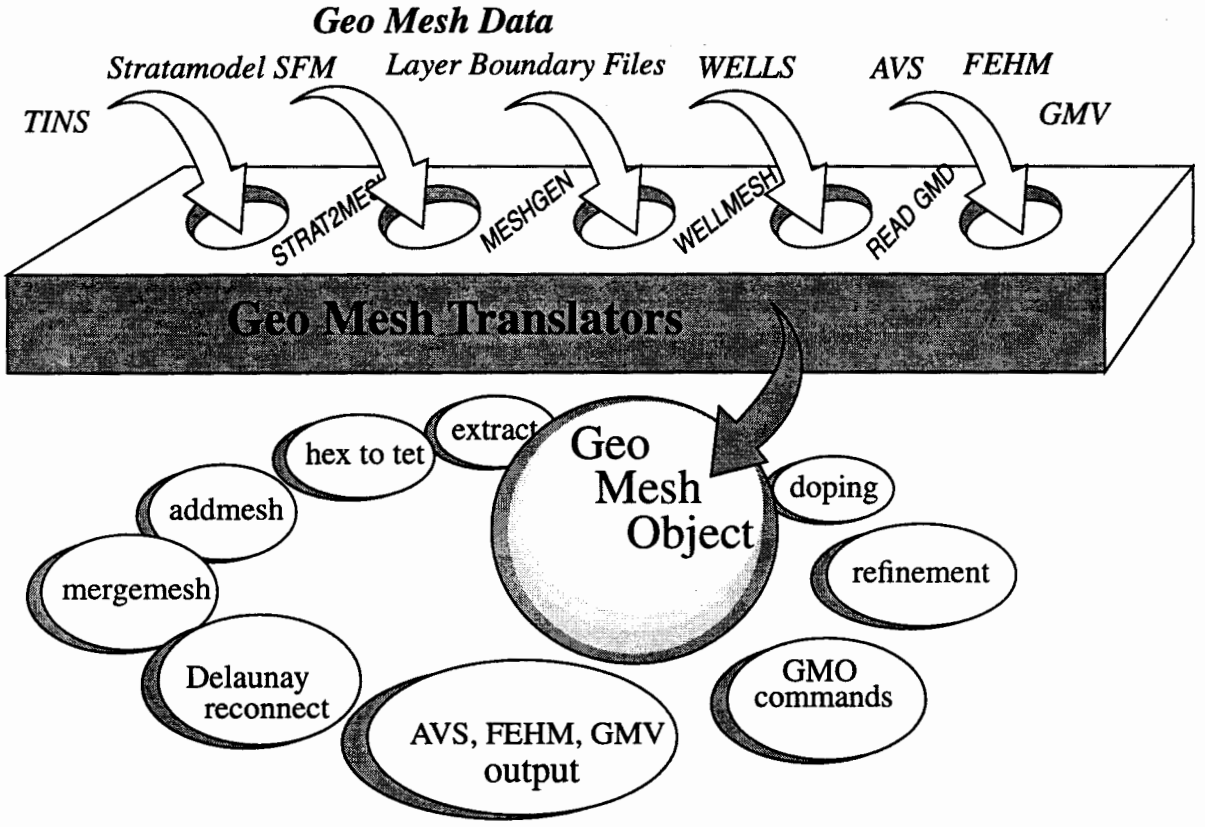
2) A **geologic framework** is defined. This geologic framework represents the area of interest with coordinate point data and material assignments for these points. We call this the Geo Mesh Data and it is the initial definition of the geometry which will be simulated. There are no standards for how this framework should be represented and forms range from simple surface coordinate files to a full stratigraphic model including complex geologic features and material attributes.

3) **Grid generation** is the creation of a 2D or 3D discretized computational grid that maintains the integrity of the original Geo Mesh Data. It is during this process that the Geo Mesh Data is translated into a Geo Mesh Object (GMO). This allows access to a command language (X3D) for object - oriented grid design and optimization.

4) The **computational model** uses the grids for solving differential equations representing the physics of a problem, such as Darcy flow and reactive transport.

5) During **performance assessment** results are evaluated and alternate grids can then be generated from the original Geo Mesh Data. By altering the coarseness of the grid, changing element types and looking at subgrids, the evaluator can get a better sense of how accurate the computational models are and identify locations for additional work.

5.0 OVERVIEW of GEOMESH SOFTWARE



Most of the time spent on code development is spent on data input and the setup of program parameters for each specific problem. The input data needs to be converted into a form that can be used by grid generation routines. Each data set has its own unique format and uses unique descriptions of its complex geometries. We call this input data, of any form, **geologic mesh data (GMD)**. During setup, the program parameters must be defined, the mesh information must be calculated when it is incomplete or missing, and the mesh attributes set. The geologic framework must be accurately and fully represented before grids can be generated.

Eventually, once all the mesh attributes are well defined, the information is stored in a structure called a **geologic mesh object (GMO)**. More than one mesh object can be created, but only one will be current. There is no limit to the number of attributes assigned to a GMO, but at the very least, the attributes include:

- mesh object name
- node coordinates
- number of nodes and elements in the mesh
- type of elements, triangle, quadrilateral, hexahedral or tetrahedral
- connectivity of elements
- node and element materials
- topographic and geometric dimensions of the mesh

The code that reads the input data and creates a mesh object, is called a **GMO Translator**. These translators include modules such as MESHGEN which reads layer boundary files, and STRAT2MESH which reads model information generated with the Stratamodel Stratigraphic Framework Model (SFM). Mesh data can also be input through mesh visualization files, such as those created by Advanced Visual Systems (AVS).

The routines that recognize a mesh object are called **GMO commands**. These are used to design the output grids. They can be used singly or in combinations. Grids can be output during any stage of the design.

The code for GEOMESH currently exists as a combination of stand alone executables such as MESHGEN, WELLMESH and STRAT2MESH. The remaining translators and mesh object commands are a part of the X3D command language. The commands and mesh objects are accessed through library calls or through an interactive command line interpreter. Because GEOMESH is using object - oriented technologies, code development is flexible and consistent and modifications are easily managed.

The GEOMESH project derives considerable benefit from using the command language X3D. This is a program developed by the Semiconductor Grid Team at Los Alamos National Laboratory which presently has a 7 FTE effort to develop software to aid semiconductor design. The code for X3D is generalized so that any grid application can use the commands, as long as a CMO is created that these commands can use. GEOMESH has been creating and testing commands for X3D that relate to geologic applications.

GEOMESH uses FORTRAN 77, C and C++, with most mathematical calculations in FORTRAN and higher level interface routines in C and C++. The code is written in the ANSI standard but will compile under non-ANSI C. The code has been developed on Sun workstations and is being ported to IBM, HP, SGI and CRAY - the same platforms that are supported for FEHMN software. Code is developed into a single source and is maintained by using configuration management software to control code changes. The code for GEOMESH and its auxiliary modules is administered under PVCS version manager on Sun workstations.

6.0 TRANSLATORS for GEO MESH DATA (GMD)

The translation of geologic data into a geologic mesh object includes the definition of regions, assigning material types to regions, point distribution, connectivity and interface definitions. These translators can be as simple as reading an AVS UCD file description of a mesh, and as complicated as STRAT2MESH which incorporates Stratamodel software for the generation of a framework model.

The capabilities included in each of these modules are:

- 2 or 3 Dimensional representations.
- Grids honor the geometric integrity of the original stratigraphic model.
- Geologic structures include faulting, beds that pinch out, and irregular external surfaces defined by topography.
- All capabilities of X3D mesh object commands are inherited by these modules.

The following is a list of GMD Translators. The type of input read by each is included for each translator.

6.1 GRIDDER

INPUT GMD: Interactive user input

Gridder is an interactive,easy to use rectangular grid generator. It can create 1, 2, or 3 dimensional grids with multiple regions. The nodal spacing for each region can be of equal size, geometric, or logarithmic. The final grid is written in AVS UCD format.

6.2 STRAT2MESH

INPUT GMD: Stratamodel SFM

A geologic framework is built using Stratamodel software by Landmark Graphics Corporation. This software provides 3D modeling of inter-well reservoir properties and distributes up to 100 attributes (such as porosity and permeability) using deterministic interpolation to build a geocellular representation of the subsurface geology.

The stratigraphic framework model (SFM) that is constructed with Stratamodel is a logically structured array of hexahedral elements with regular spacing in x and y directions. Spacing is variable in the z direction. While this model provides a well defined model of the 3D stratigraphy, it has qualities which make it impossible to use for flow and transport calculations.

The SFM can be input into GEOMESH to create a computational grid. This grid is made up of tetrahedra 2D or 3D triangles, the grid is Delaunay, coupling coefficients are positive and all material interfaces are accurately preserved. STRAT2MESH allows the model to be preserved in its entirety, or subsets can be selected. These subsets can be 3D blocks or 2D cross-sections.

6.3 MESHGEN

INPUT GMD: Data files containing layer boundaries

Meshgen is the heart of GEOMESH grid generation and where most of the data sets are processed. It is an executable that runs on two control files, one to define the program parameters, the other lists files to be used as the GMD. The following are meshgen capabilities, additional features are available through X3D commands.

- Subsample of input at a lower resolution.
- Ability to specify thickness at which a bed will be considered to have pinched out.
- Control of nonlinear vertical refinement including linear interpolation between 0 and 1, bias towards smaller z values, or bias towards larger z values.
- Extract a rectangular (or 2D slice) subset of the original input data based on node index values or coordinate values.
- Transform the coordinate system.
- User control of layer ordering.
- Generate uniform or rectilinear grids.
- Hexahedral or tetrahedral elements where hexahedral elements can be split into 5 or 24 tetrahedrals each.
- Output formats include AVS, GMV, FEHM, and X3D.

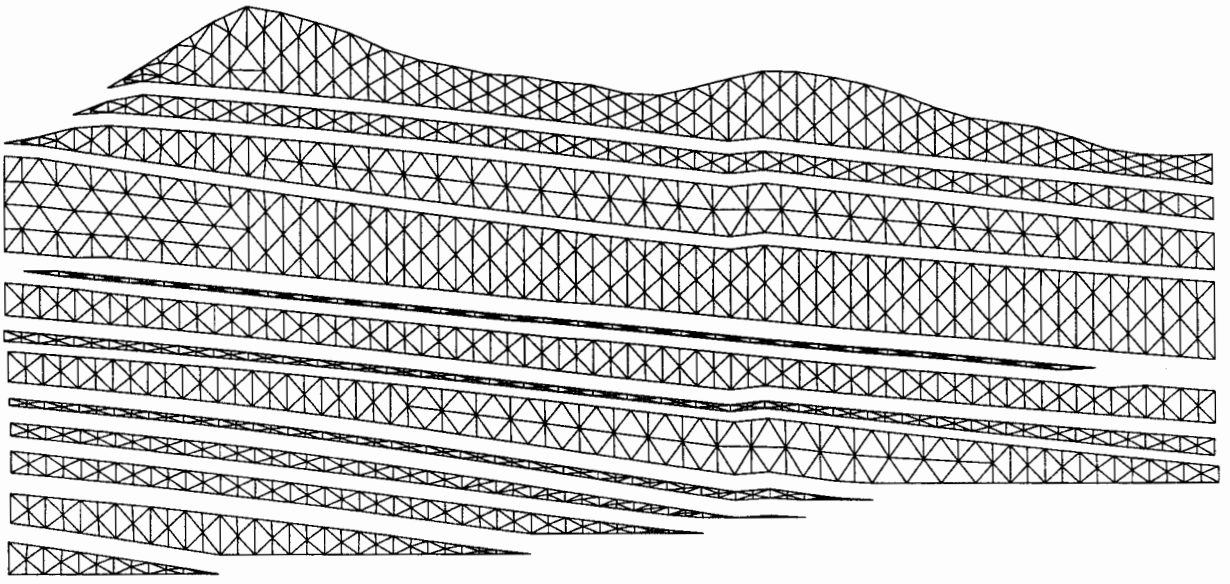


FIGURE 1. Two-dimensional, unstructured finite element mesh of Yucca Mountain through Antler Ridge, separated at stratigraphic layer boundaries. The mesh was created using the MESHGEN program to extract a 2D mesh from a 3D data set.

6.4 READAVS

INPUT GMD: reads a file generated from AVS in UCD file format

A GMO is created by reading an AVS UCD file. A UCD data structure consists of an irregular coordinate model made up of cells. Cells may be quadrilaterals, triangles, tetrahedrons, or hexahedrons. Data is associated with each node. The elements for the mesh can be triangles, quadrilaterals, hexahedrals, or tetrahedrals. Mesh data read from the AVS file includes:

- Number of nodes, elements and node materials.
- Coordinate locations for the nodes
- Element id, type and connectivity
- Node materials
- Node type (optional)
- Node constraints (optional)

6.5 READFEHM

INPUT GMD: reads a FEHM *.fin file

This command reads an output file from FEHM (.fin file) for field coefficients at each node. This is used with the mesh command, DOPING which interpolates onto a new grid (usually more refined than the first). A *.ini is the same format as the *.fin file and is used as a restart file for FEHM.

The FEHM *.fin file contains the final values of pressure, temperature, saturation, and simulation time for the run. Mesh data read from the FEHM file includes flags for gas, tracer, stress, dpdp, and dual. Depending on the flag set, the following can be read:

- Initial time of simulation
- Final temperature at each node
- Final saturation at each node
- Final pressure at each node
- Final capillary pressure at each node
- Number of species
- Species concentration
- Number of particles, final random number seed
- Final node position for each particle
- Fractional time remaining at current node for each particle
- Multiplier to the plug flow for each particle at node position
- Age for each particle

6.6 READPTS

INPUT GMD: reads a x, y, z point distribution, no connection

This command can read a set of scattered points, then form a Delaunay tetrahedralization. This was done to compare a LBL USGS model to FEHM flow calculations.

6.7 WELLMESH

INPUT GMD: center coordinates and radius of a well or tunnel

Wellmesh creates a mesh representation of wells or tunnels along an arbitrary path. This path is described by its center coordinates x, y, and z. This module is used with the mesh object command, ADDMESH to embed wells within a grid. This allows the accurate representation of air, vapor, and fluid flow in the vicinity of wells.

The following are wellmesh capabilities:

- Input center coordinates or calculate center axis from top coordinate to a given depth.
- Radial and height refinement of a well or tunnel.
- Integer material assignment to well layers or stratigraphic layers.
- Hexahedral or tetrahedral elements where hexahedral elements can be split into 5 or 24 tetrahedrals each.
- Reconnect to insure a Delaunay Grid.
- Output formats include AVS, GMV, FEHM, and X3D.

7.0 COMMANDS for GEO MESH OBJECT (GMO)

These commands are accessed either through the X3D command line interpreter, or through FORTRAN calls in the GEOMESH code. These commands can be used separately or can be combined. In some cases, the command is a collection of routines that are dependent on the individual grid being designed.

The following commands summarize the capabilities available through GEOMESH and X3D calls.

7.1 DELAUNAY AND VORONOI RECONNECT

RECONNECT is used to insure that constraints on grid discretization are optimized for accurate and stable numerical solutions. The connectivity of the nodes form elements which will be a Delaunay triangulation in two dimensions, or Delaunay tetrahedralization in three dimensions, of the point distribution. A Voronoi mesh and the Delaunay mesh are duals of one another meaning one can be derived from the other. The Voronoi mesh is unique, however the Delaunay mesh may not be. A median mesh is formed by connecting the midpoint of each element edge to the centroid of the element. Note that the median mesh does not always form convex cells.

A comparison of the Delaunay, Voronoi, and Median mesh forms are shown below.

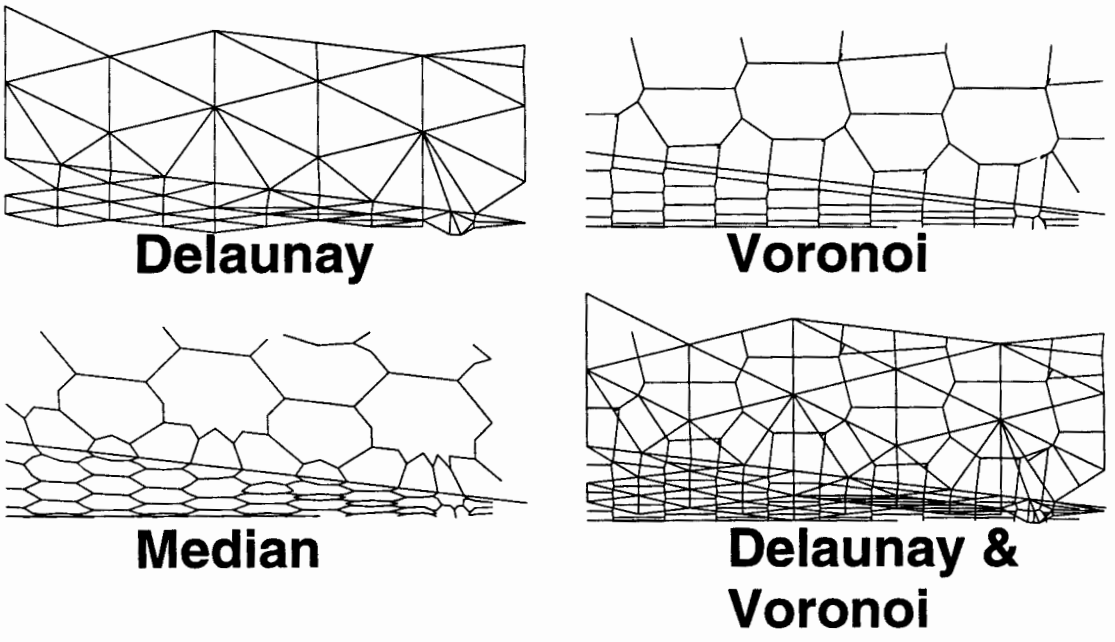


FIGURE 2. Different numerical algorithms use different discretizations of a point distribution to solve computational physics problems. GEOMESH can output grids in these various forms.

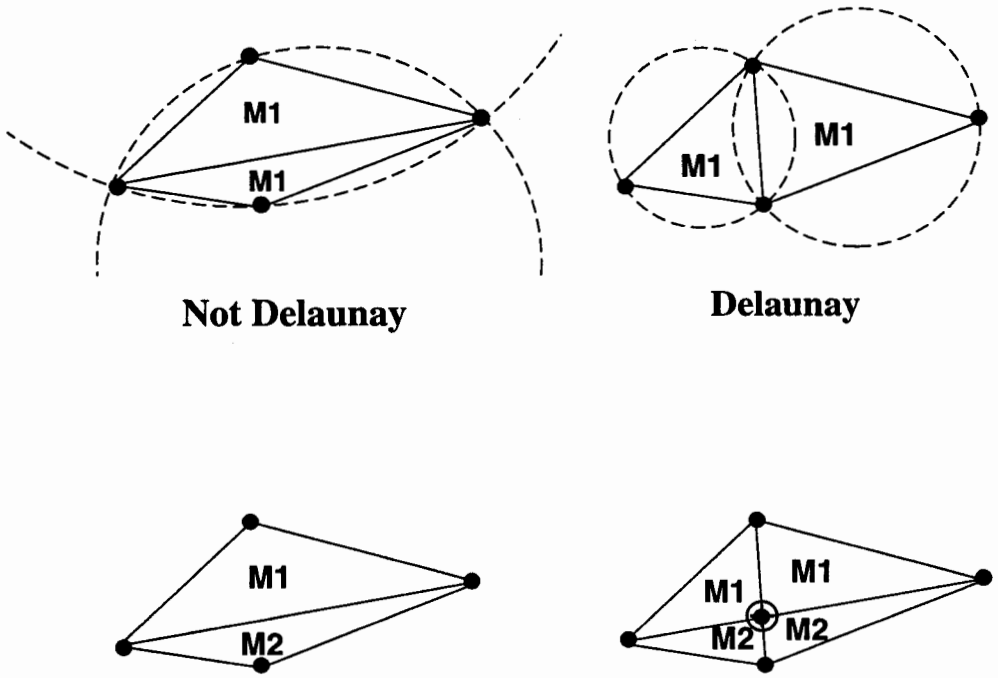


FIGURE 3. Example of four points connected to form a Delaunay triangulation. In the upper figure both triangles are the same material so the triangulation is made Delaunay by flipping a connection. In the lower figure, the triangles on the left are of different materials, so the triangulation is made Delaunay by adding a point on the interface and making 4 triangles from the original 2.

Voronoi grids and Delaunay triangulations have been described by mathematicians and physicists going back nearly one hundred years and there are many ways to define them. A Voronoi point is the center of the circle that passes through the three vertices of a triangle. If the Voronoi point is inside or on the triangle, the triangle is a Delaunay triangle. This can be generalized to 3-D with a sphere circumscribed around a Delaunay tetrahedron. A Delaunay mesh consists of a set of Delaunay triangles (or tetrahedra) such that the circle (or sphere) circumscribing each triangle (tetrahedron) contains no mesh points other than the vertices of the triangle (tetrahedron), excluding the degenerate case where the Voronoi point falls on the edge of the triangle or the face of the tetrahedron.

In cases where there is only a single material, a Delaunay mesh can be constructed by flipping element connections until the Delaunay criteria is satisfied. However, when there are multiple materials, connections cannot be flipped if material interfaces are to be maintained. In that case, new nodes are automatically added along the material interface. This results in a Delaunay grid which maintains the geometric integrity of complex material interface.

7.2 HEXAHEDRAL TO TETRAHEDRAL

A tetrahedral grid is formed from a hexahedral grid. The elements can be split into either 5 or 24 elements each. This results in an unstructured grid of triangles in 2-D or tetrahedra in 3D. In contrast, a structured grid that is typically made up of quadrilaterals in 2D or hexahedra in 3D. While numerical algorithms are often easier to implement on structured grids (i.e. finite difference methods), they lack adaptivity in applications that must represent complex geometry. This command can be used before calling RECONNECT to convert a structured hexahedral grid to a Delaunay tetrahedral grid.

7.3 INTERPOLATION (DOPING)

In order to restart a calculation after grid refinement or to compare solutions on completely different grids, capabilities have been developed to interpolate node based field values from one grid onto another.

One application of this is to use algorithms to interpolate field quantities from one grid to another to speed convergence of fine grid calculations. For example, in order to obtain a steady state saturation field it is faster to interpolate a course grid solution onto a fine grid rather than start the fine grid calculation without any prior information.

A second application involves interpolating the saturation field onto grids using FEHM. This requires additional capabilities to calculate a grid connectivity from a scattered point data set (i.e. x, y, z, coordinates and saturation value with no connectivity provided).

7.4 MESH OPERATIONS (ADD, MERGE, AND EXTRACT)

These operations allow for flexible grid designing. The operations can be combined in a number of ways to achieve the desired grid. The most common operations are:

Extract a surface from a 3D grid. This can be used to create a two dimensional planar slice from a three-dimensional tetrahedral grid.

Merge is the union of two meshes. There is no reconnection involved, the two meshes simply occupy the same geometric space. This is good for checking meshes before performing the add mesh command.

Add allows the user to symbolically add two meshes together. This is useful for embedding fine meshes within course meshes. For example, imbedding a finely resolved site scale model into a coarse regional groundwater model will eliminate the common problem of how to set appropriate boundary conditions for a site scale model. A coarse grid simulation at the regional scale should provide an excellent approximation of the conditions at the edges of the site scale model domain. The figure below illustrates the process used to add two meshes together.

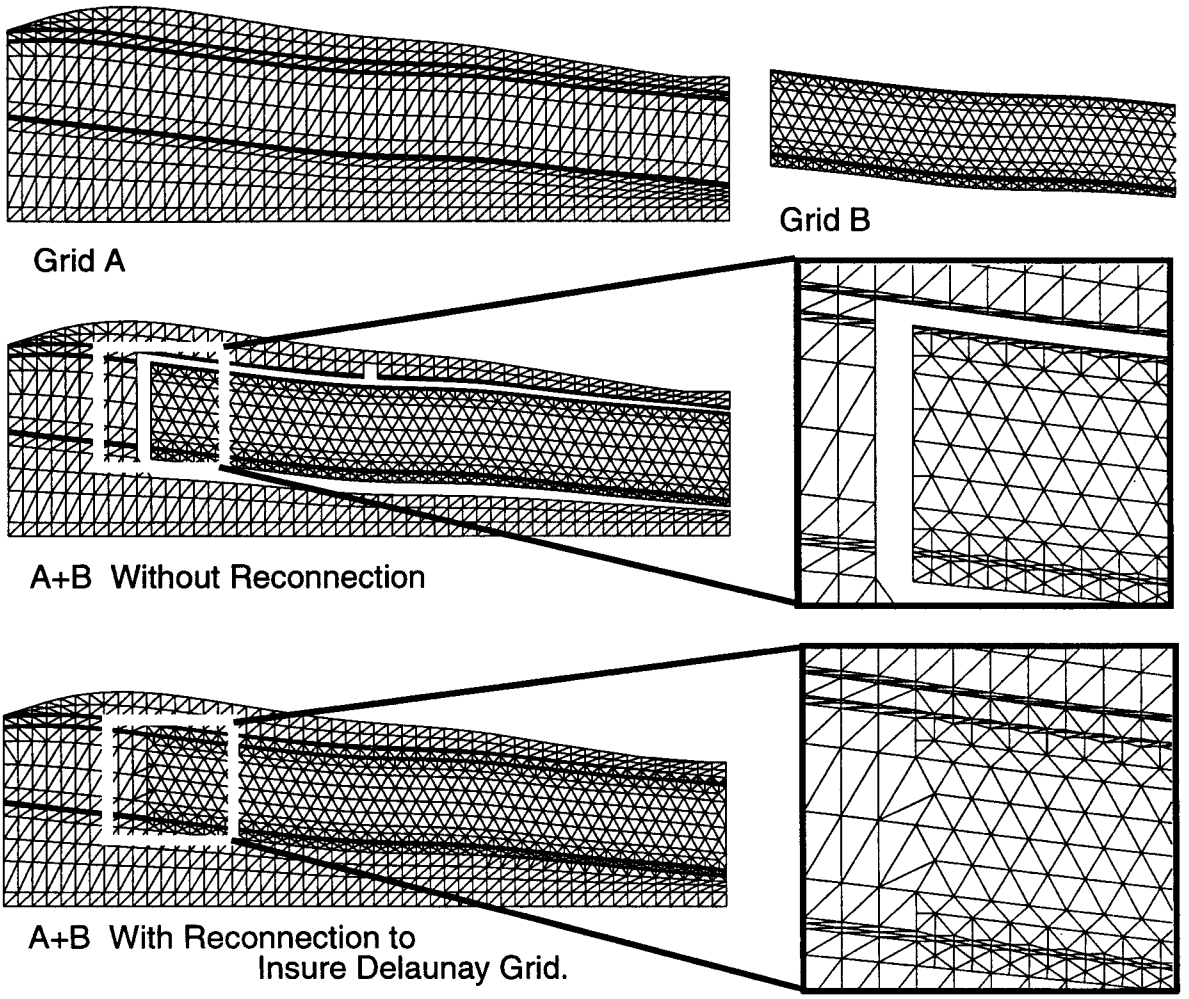


FIGURE 4.
Two grids are added together by first creating the two grids A and B.
Grid A is then excavated in the area that will be occupied by grid B.
During the final step, the interface between the grids A and B is filled and reconnected to insure the entire grid will be Delaunay.

7.5 RESOLUTION

Depending on the physics of the problem being solved, one may wish to increase resolution near interfaces. An example of the ability of GEOMESH to handle this situation is shown below. The figure on the left shows a grid with vertical resolution evenly spaced within each unit, while the example on the right has higher resolution near the interface. This capability will allow us to carefully examine the effect of contrasting material properties at interfaces on the hydrologic and transport model.

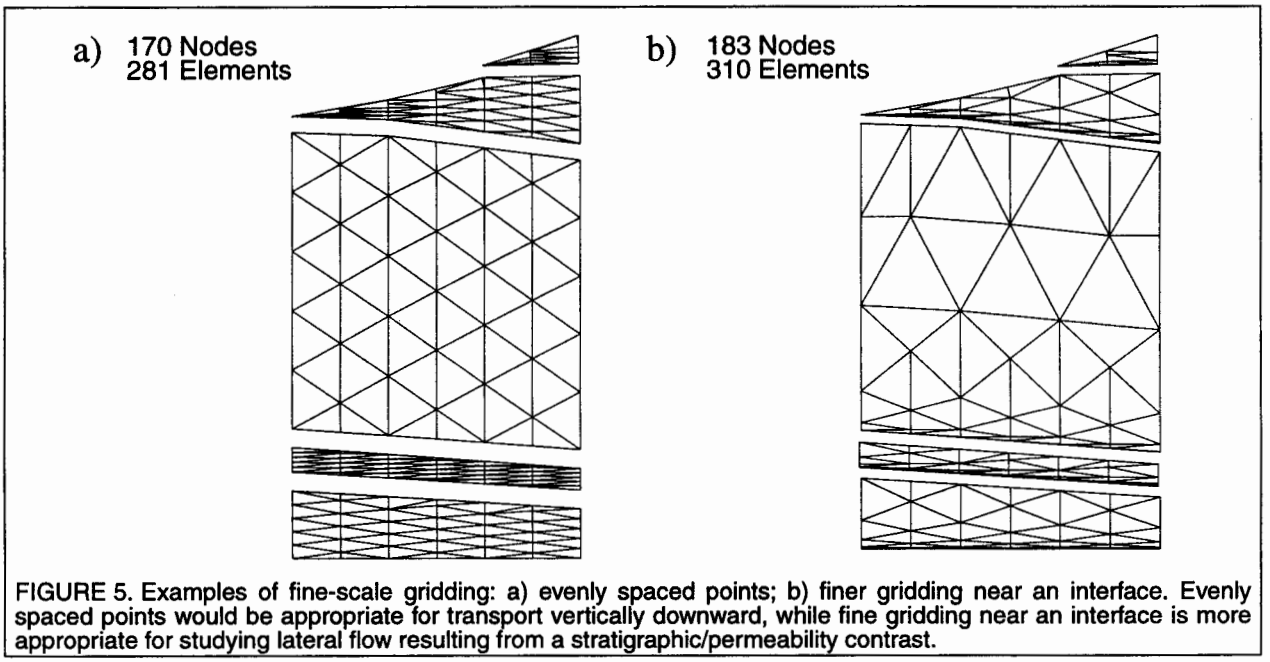
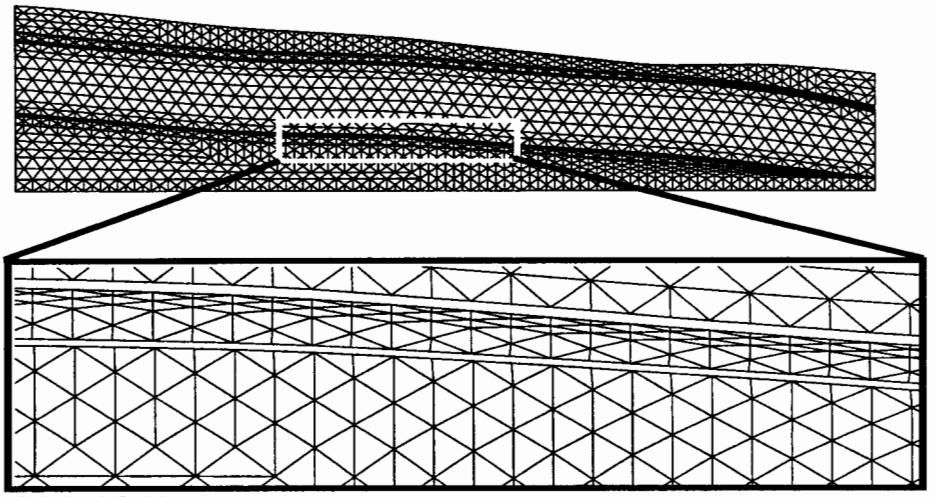


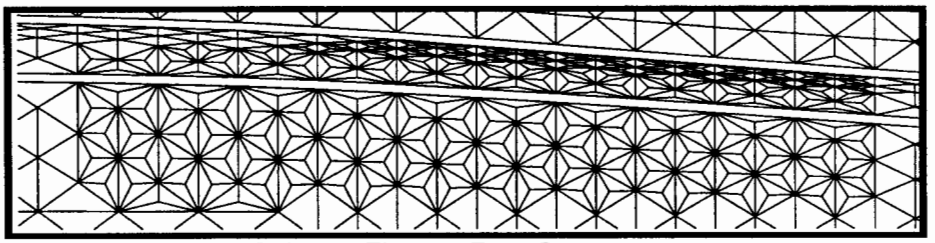
FIGURE 5. Examples of fine-scale gridding: a) evenly spaced points; b) finer gridding near an interface. Evenly spaced points would be appropriate for transport vertically downward, while fine gridding near an interface is more appropriate for studying lateral flow resulting from a stratigraphic/permeability contrast.

7.6 REFINEMENT

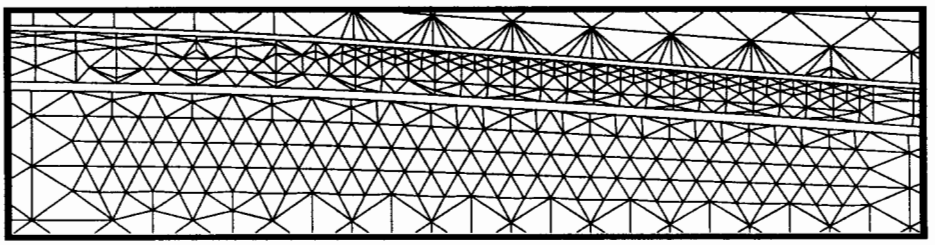
Refinement gives the user the ability to increase and decrease grid resolution by splitting elements. Three methods of grid refinement are available; edge refinement, face refinement and volume refinement. Refinement can be based on location, interfaces, or materials. The integrity of the stratigraphic input and any attributes associated with the nodes are maintained during this process. Refinement is useful when one must resolve physics on a fine scale, or as a test of convergence of a solution. Steady state saturation solutions can be obtained quickly on a course grid, then refined for more detailed and accurate calculations. An example of this process is shown below.



Original Grid



Refinement Adding Points to Element Face Centers



Reconnection To Insure Delaunay Grid

FIGURE 6. The steps in adaptive mesh refinement (AMR) are shown. First an area to be refined is chosen. The elements are then refined by adding nodes to the center of element faces. The last step is to reconnect the mesh to insure a Delaunay grid while not altering material interfaces. The materials have been separated to illustrate the interface between materials.

7.7 STRATIGRAPHIC HORIZONS AND LENSES

Stratigraphic horizons can be incorporated into the computational grid. An example of this is shown below where a potential repository horizon is incorporated as a thin layer. These are used to study transport of tracers released from a repository.

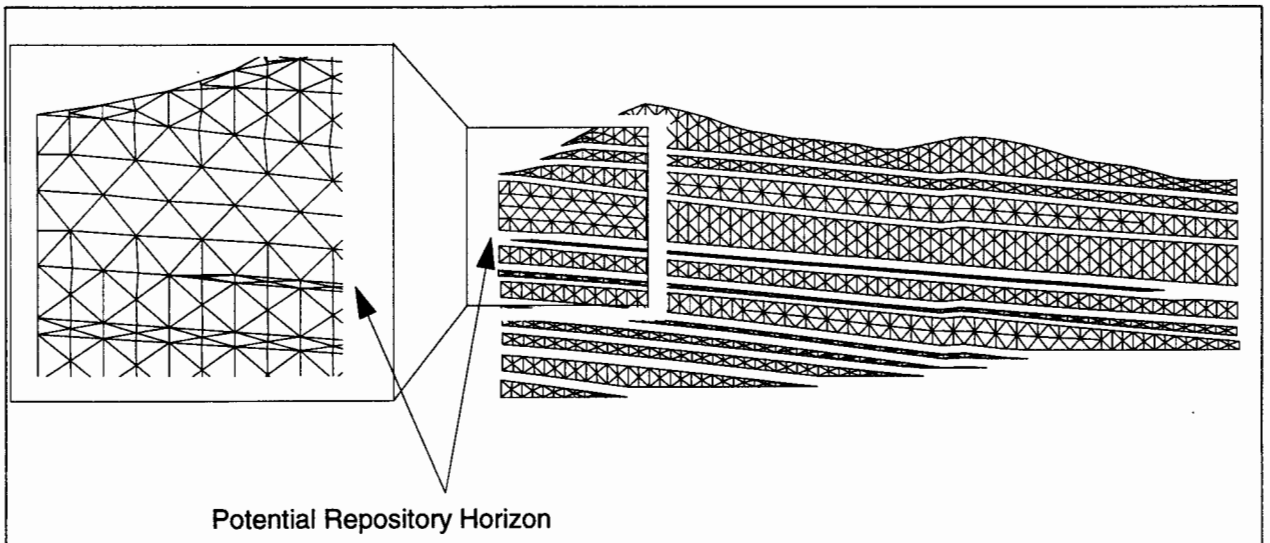


FIGURE 7. Two-dimensional, unstructured finite element mesh of Yucca Mountain's Antler Ridge. The enlarged area shows the western portion of the site. The potential repository is included as a thin zone within an existing mesh that honors the interfaces of the hydrostratigraphic units at Yucca Mountain.

8.0 OUTPUT of FINAL GRIDS

The grid generation process can produce information about the Mesh Object that can be used by finite element, finite volume and integrated finite difference physics codes. Some of the information is used to define finite difference operators (Div, Grad, Curl) while other information is used for setting initial and boundary conditions.

The output files for computational grids contain the following:

- Node coordinates - x, y, z coordinates of every node in the mesh.
- Node color - any number of floating point values associated with each node. This could be the material properties of materials (intrinsic variables, i.e. density, porosity, ...) or could be values derived from a model simulation (i.e. temperature, pressure, ...)
- Node type - every node has an integer associated with it indicating whether it is inside the mesh, outside the mesh or on a material interface. Outside nodes have additional information indicating which of the external faces they are on.
- Node connectivity - list indicating the neighboring nodes connected to any node.
- Element color - every element has an integer indicating which material an element belongs too. This is useful for assigning material properties in physics codes and is used as a constraint during grid reconnection.

- Element connectivity - provides a node list associated with each element describing how nodes are connected to form an element.
- Face connectivity - list associating any face of an element with the corresponding face of the element that shares that face.
- Area coefficients (FEHM *.stor file)
- Zone files - three files providing lists of nodes sorted by: a) material type, b) material interface and c) outside face type. These lists are useful for setting initial and boundary condition.

9.0 OUTPUT FORMATS

Output can be written in various formats. Both AVS and GMV are useful for visualizing the grids, debugging, and for input back into the GMO commands. Shaded views help communicate observations of the model for qualitative interpretation and helps in conceptualizing data results. The ability to rotate and slice through a 3D model can reveal features often missed in 2D.

9.1 AVS UCD

AVS is a data visualization system that allows users to dynamically connect software modules to create data flow networks for 3D interactive rendering and volume visualization. The file format includes number of nodes and elements, node coordinates, node materials, node types, node constraints, element type and connectivity.

9.2 FEHMN

The code FEHM (Finite Element Heat & Mass) simulates fluid flow and heat transfer in porous and fractured media. It uses a finite element technique in 2-D or 3D, is fully coupled and implicit, and handles nonlinear material properties. The *.ini file can be used to restart FEHM.

9.3 GMV

GMV is a 3D visualization tool that can process data from meshes. The GMV file format can include the following; number of nodes and elements, node coordinates, cell data, material data, velocities, polygons and tracers. GMV is also used as input into stereo 3D and virtual reality systems. Both require additional hardware, including LCD shutter glasses, and a glove for virtual reality. These systems improve model evaluation by allowing views inside the grids, making probing and modifications more rapid and intuitive.

9.4 X3D

An output file is written using information from the current mesh object. This file can then be used to restart X3D commands at the point the file was written.

10.0 USER INTERFACE

Currently, each module has its own command line interface and different input methods. A command language is used to design the grids once a mesh object has been defined. The X3D command interpreter is run interactively and creates a file that can be used to run the problem in batch mode.

A user-friendly interactive interface has been prototyped on the wellmesh module and a GUI interface exists for the MESHGEN module as part of the FEHM browser.

11.0 GEOMESH VALIDATION

Quality assurance is verified from version to version by running and checking a variety of test problems that continue to expand as new features are added. Geomesh is used extensively within the group which provides not only measured results, but as modifications are needed for new problems, they are integrated into the released version of GEOMESH.

Validation tests are planned which will include heat transport studies on various grids and features. Software libraries provided by the LANL grid team are tested and validated before being used by GEOMESH.

GEOMESH is also used in the first step of quality analysis of geometric data. The creation of a geometric grid is often the first time the data is visualized and obvious errors can be seen.

12.0 FUTURE PLANS for the GEOMESH PROJECT

There are two primary GEOMESH needs:

1) The design and implementation of a user interface that makes grid generation as easy and seamless as possible.

2) Many routines need to be generalized. Some of the GEOMESH codes have parameters that are hard-wired for the problems that have been worked so far. Though this has enabled timely completion of required grids, and has allowed the development of code to progress, new data sets will require code changes.

12.1 GMD TRANSLATOR PLANS

The highest priority for translators is for one that can read EarthVision TIN files. Currently the best model of Yucca Mountain stratigraphy is a model produced by USGS and archived as EarthVision TINS. Once readtin is implemented, the model can be read into GEOMESH for a more accurate study of the area.

The following additional translators are needed by the GEOMESH project:

- readlynx
- readcps3
- readintergraph

The following code changes will need to be made:

- Strat2mesh will have to be updated to read new Stratamodel SFM output. Data structure and framework model will change, as well as how pinched out layers are terminated.
- Meshgen will continue to be modified as user needs are discovered.
- Wellmesh representations compared to NURB and node representations of wells.
- Meshgen and wellmesh added to mesh object command language for greater versatility.

12.2 GMO COMMAND PLANS

GMO commands are being developed by both the GEOMESH and Semiconductor Grid Team. Many of these have complex usage issues, partially due to the fact that the code has been generalized for all applications, and partially because the code is still in a development phase. The generalization issue can be addressed by creating GEOMESH "wrappers" that apply to commonly used grid commands for geologic applications.

New commands to develop include:

- Output grids compatible with TOUGH2 code
- Grid coarsening
- faultmesh
- better control of screen output, error logs
- version of hextotet that allows mixed elements
- 2D well

12.3 DOCUMENTATION PLANS

Draft versions of the following documents have been made:

- *GEOMESH Project Description* - overview of capabilities, plans, and grids.
- *User's Manual for GEOMESH* - how to create computational grids.
- *GEOMESH Worklist* - running record of work done for the GEOMESH project.
- *X3D User's Manual* - how to run the command language.
- *GEOMESH Handout* - a short, summarized version of the Project Description.
- *GEOMESH Programmer's Handbook* - code development details for GEOMESH.

The GEOMESH Worklist is an ongoing document that is updated approximately every two months.

All documents, except the GEOMESH Worklist, are a first draft in progress. These need to be updated. The programmer's manual needs much work.

User manual for GMO COMMANDS is needed. But until a user interface is developed, and routines are generalized for geologic use, user manuals for the GMO commands will be difficult to write. Some of the commands can be documented, others will have to wait for future code development.

User manual for STRAT2MESH is on hold until Stratamodel releases its new modeling format.

13.0 REFERENCES

Fraser D., "Tetrahedral Meshing Considerations for a Three-Dimensional Free-Lagrangian Code", Los Alamos National Laboratory report, LA-UR-88-3707, 1988.

Gable, C.W., Harold Trease, Terry Cherry, Automated Grid Generation From Models of Complex Geologic Structure and Stratigraphy, Third International Conference/ Workshop on Integrating GIS and Environmental Modeling, abstract, LA-UR-95-2665, 1996.

Gable, C.W., Harold Trease, Terry Cherry, Automatic Grid Generation From Complex Models Of Geologic Structure And Stratigraphy, GSA, abstract, LA-UR-95-2482, 1995.

Gable, C.W., George Zyvoloski, Site Scale Modeling of Radionuclide Transport At Yucca Mountain, NV: Grid Generation and Reactive Tracers, LA-UR-94-1041, 1994.

Gable, C.W., G. A. Zyvoloski, Bruce Robinson, Integration of hydrostratagraphic data in reactive transport models, Chapman Conference on Hydrogeologic Processes: Building and Testing Atomistics- to Basin-Scale Models, Lincoln, New Hampshire, 1994.

George, D.C. and Trease, H.E., "X3D User's Manual", to be published.

Khamayseh, A. and Andrew Kuprat, "Anisotropic Smoothing and Solution Adaption for Unstructured Grids", LA-UR-95-2205, International Journal for Numerical Methods in Engineering, (submitted).

Khamayseh, A., Andrew Kuprat, and Frank Ortega, "A Robust Point Location Algorithm for General Polyhedra", (to appear in Computer Aided Geometric Design).

Moridis, G. and K. Pruess, "Flow and Transport Simulations using T2CG1, a package of conjugate gradient solvers for the TOUGH2 family of codes", LBL 36235, April 1995.

Painter, J.W. and Marshall, J.C., "Three-Dimensional Reconnection and Fluxing Algorithms", *Proceedings of the Next Free-Lagrange Conference*, Jackson Lake Lodge, Wyoming, June 3-7, 1990, Springer-Verlag Press, Vol. 395, pp. 139-148.

Sahota, M.S., "Delaunay Tetrahedralization in a Three-Dimensional Free-Lagrangian Multimaterial Code", *Proceedings of the Next Free-Lagrange Conference*, Jackson Lake Lodge, Wyoming, June 3-7,1990, Springer-Verlag Press, Vol. 395, pp. 130-138.

Sahota, M.S., "An Explicit-Implicit Solution of the Hydrodynamic and Radiation Equations", *Proceedings of the Next Free-Lagrange Conference*, Jackson Lake Lodge, Wyoming, June 3-7, 1990, Springer-Verlag Press, Vol. 395, pp. 57-65.

Trease, H.E., "Adaptive Mesh Refinement (AMR) On Unstructured Tetrahedral Grids", to be published.

Trease, H.E. and Dean, S.H., "Thermal Diffusion in the X-7 Three-Dimensional Code", *Proceedings of the Next Free-Lagrange Conference*, Jackson Lake Lodge, Wyoming, June 3-7, 1990, Springer-Verlag Press, Vol. 395, pp. 193-202.

Trease, H.E. "Three-Dimensional Free Lagrangian Hydrodynamics, "*Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics*, Springer-Verlag, Vol. 238, pp. 145-157, 1985.

Trease, H.E. "Three-Dimensional Free Lagrangian Hydrodynamics", *Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics*, Springer-Verlag, Vol. 238, pp. 145-157, 1985.

Trease, H.E. "Parallel Nearest Neighbor Calculations", *Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics*, Springer-Verlag, Vol. 395, pp. 149-156, 1985.

Trease, H.E. "Three-Dimensional Free Lagrangian Hydrodynamics", *Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics*, Springer-Verlag, Vol. 238, pp. 145-157, 1985.

Trease, H.E. (1981), "A Two-Dimensional Free Lagrangian Hydrodynamics Model", Ph.D. Thesis, University of Illinois, Urbana-Champaign.

14.0 GEOMESH PROJECT TEAM

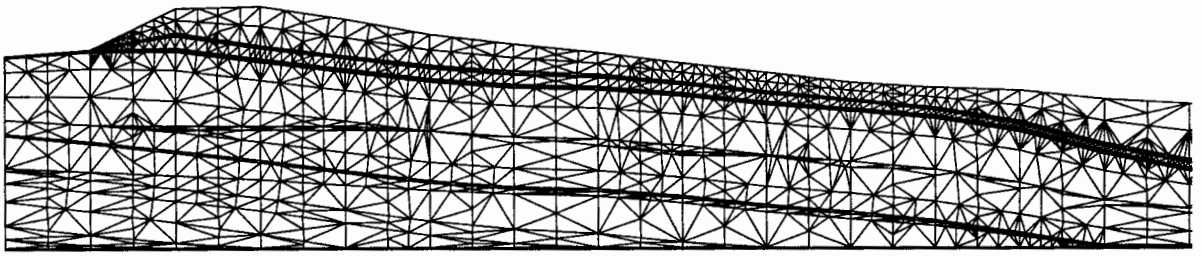
The GEOMESH team includes Carl Gable, Harold Trease and Terry Cherry. This project also derives considerable benefit from the Semiconductor Grid Team at Los Alamos National Laboratory which presently has a 7 FTE effort to develop software to aid semiconductor design.

Contact:	Carl Gable
Address:	EES-5, Mail Stop F665 Los Alamos National Laboratory Los Alamos, NM 87545
Phone:	(505) 665-3533
FAX:	(505) 665-3687
Email:	gable@lanl.gov

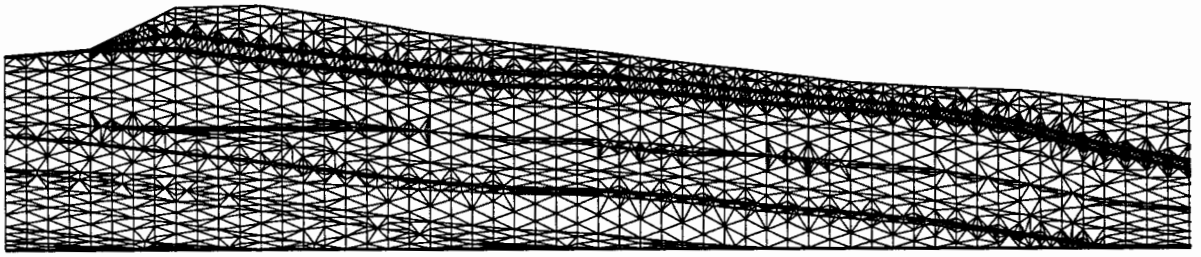
15.0 GRID CATALOGUE

15.1 ANTLER RIDGE CROSS SECTIONS

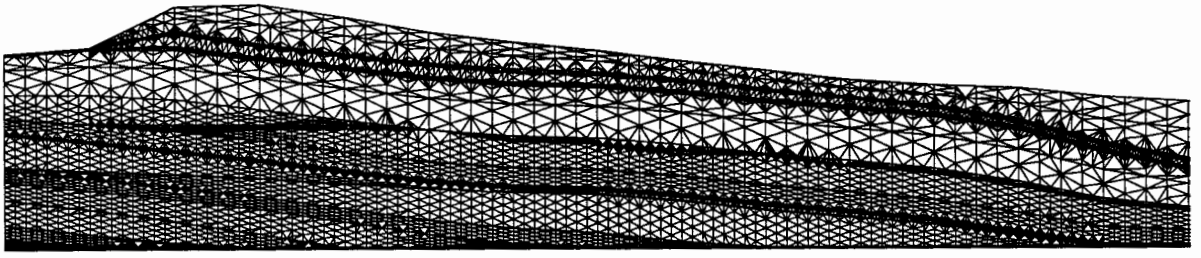
Title	Antler Ridge Cross Sections
Purpose/Application	Produce 2-D cross sections for flow and transport
Date of Project	August 1995
Number of Nodes	2106 (low), 4587 (med), 7977 (high)
Number of Elements	4057 (low), 8944(med), 15,630 (high)
Description	Provide grid with enough resolution for transport calculations modeling release from the potential repository. High resolution grid produced from low resolution grid by first refining the entire grid and then refining stratigraphic units at or below the potential repository.



2106 Nodes, 4057 Elements



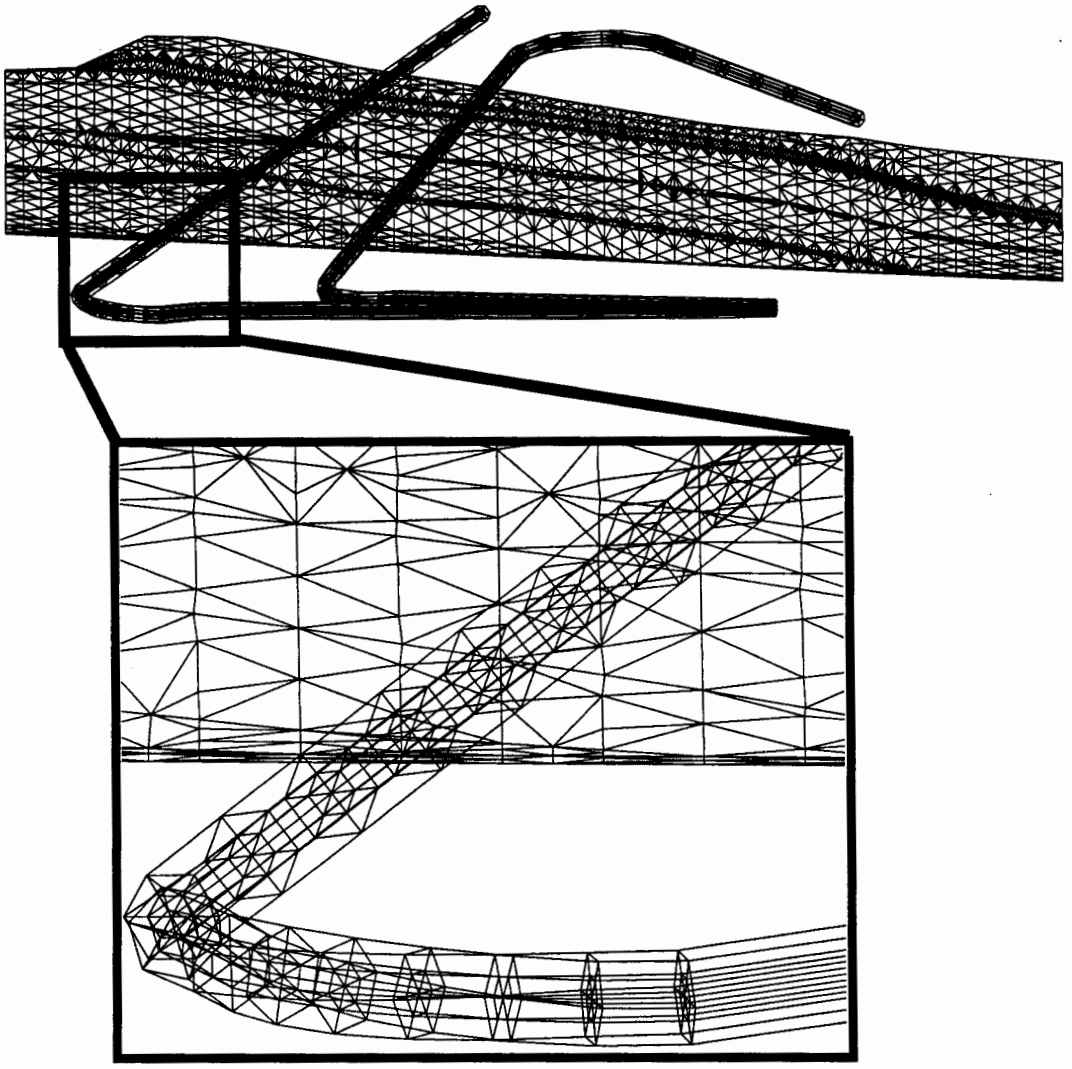
4587 Nodes, 8944 Elements



7977 Nodes, 15,630 Elements

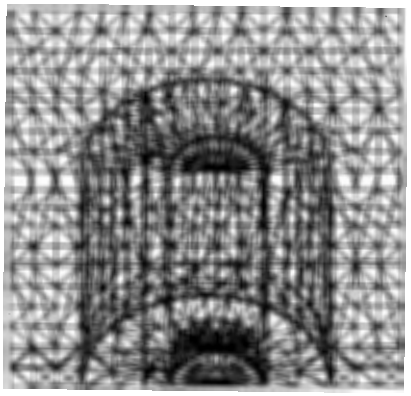
15.2 ESF AND FOCUS DRIFTS INTO YUCCA MOUNTAIN

Title	ESF and Focus Drifts into Yucca Mountain
Purpose/Application	Represent tunnels as part of site scale model for modeling effect of tunnel on flow and transport
Date of Project	March 1995; continuing
Number of Nodes	1404(ESF), 1440 (Focus)
Number of Elements	468 (ESF), 480 (Focus)
Description	Data for location of drifts obtained from D. Jefferis, EG&G. Produce grids from series of x, y, z, locations defining drift. Well mesh has two levels of radial refinement. Antler Ridge cross section shown for scale and location.



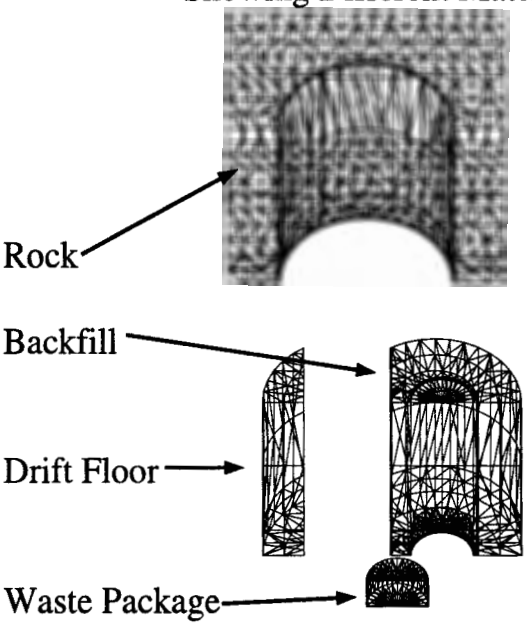
15.3 WASTE PACKAGE, BACKFILL, ROCKMASS

Title	Waste Package, Backfill, Rockmass
Purpose/Application	Model heat and vapor transport near waste package.
Date of Project	August 1995
Number of Nodes	1878
Number of Elements	5736
Description	Develop grid that accurately represent components of near field around waste package in 3-D. Due to symmetry of the problem only 1/4 of the geometry is represented.



Finite Element Grid
1876 Nodes
5736 Elements

Exploded View of Grid
Showing Different Materials

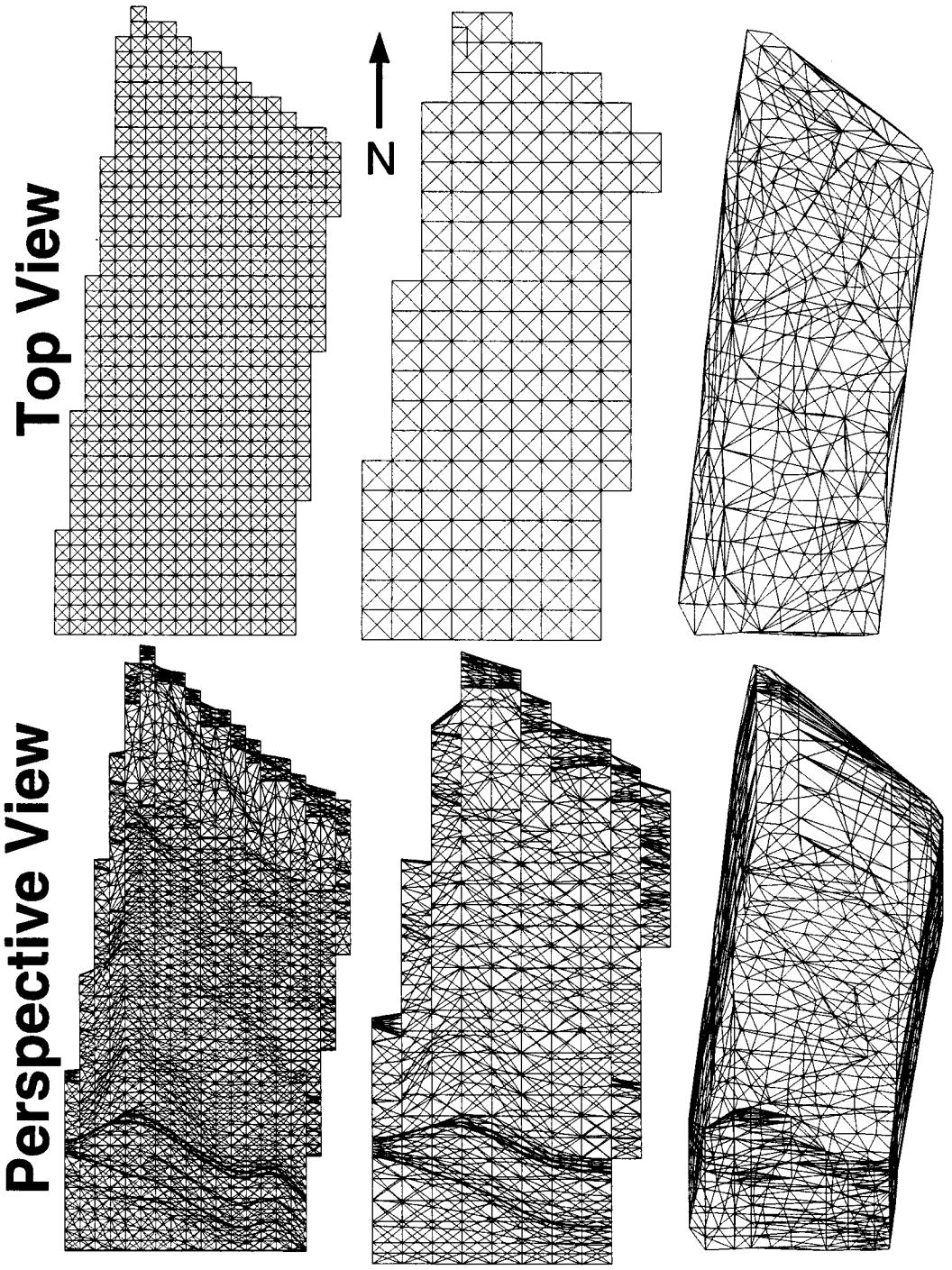


Example of finite element grid produced for Intera by LANL EES-5 for heat and mass transport calculations using FEHM.

cwg_00126

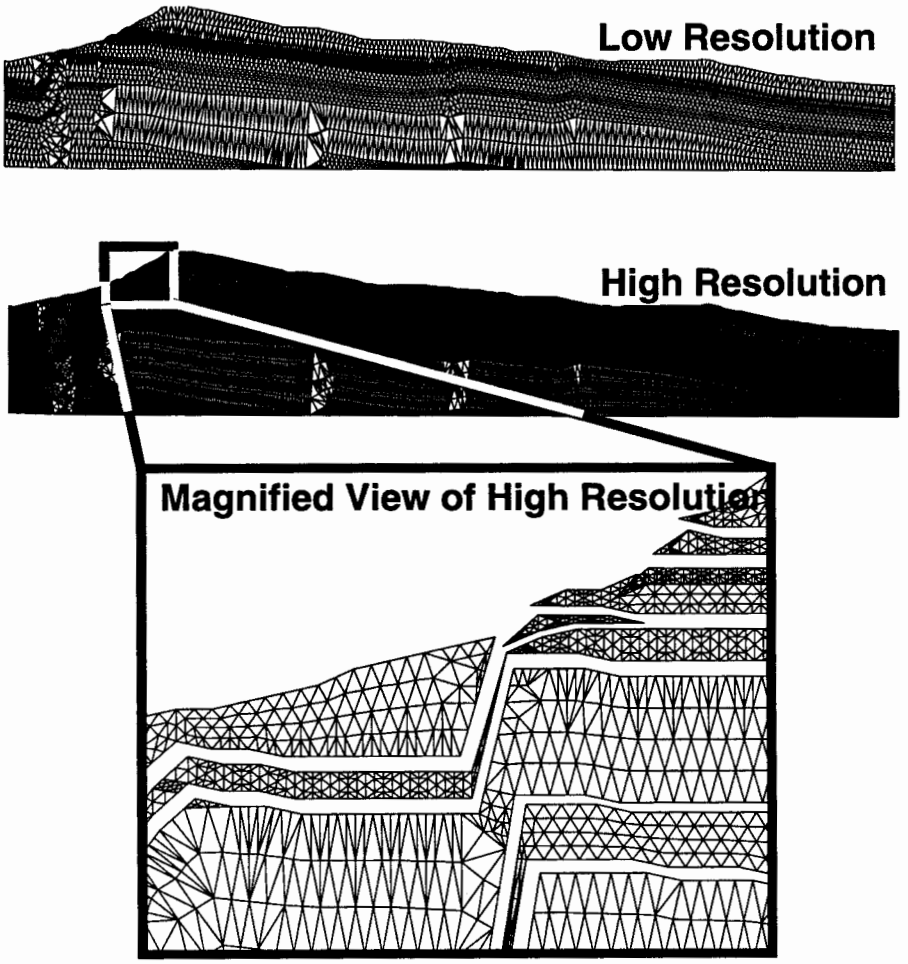
15.4 3D Yucca Mountain Grids

Title	3D Yucca Mountain Grids
Purpose/Application	Compare Saturation field of LBL/USGS, Wittwer et al. 1995 to preliminary calculations on low resolution tetrahedral grid.
Date of Project	August 1995
Number of Nodes	na
Number of Elements	na
Description	Saturation data delivered from LBL as 5,500 x, y, z points with associated saturations values. Because no connectivity was provided, point distribution is used to calculate the Delaunay tetrahedralization of point set so that saturatation values can be interpolated onto LANL low resolution, 14,150 node, 82,495 element grid. High resolution LANL grid, 51,550 nodes, 312,465 elements, is also shown for comparison.


Top View
N
Perspective View

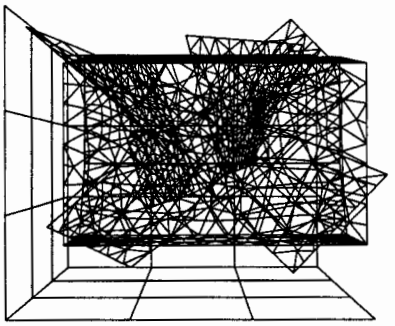
15.5 YUCCA MOUNTAIN CROSS SECTION WITH FAULTS

Title	Yucca Mountain Cross Section with Faults
Purpose/Application	Modeling Flow and Transport
Date of Project	August 1995
Number of Nodes	4694 (Low resolution), 31,490 (High resolution)
Number of Elements	8932 (Low resolution) 35,728 (High resolution)
Description	Grid constructed from data defining layer interfaces provided by Susan Altman, Sandia National Laboratory.

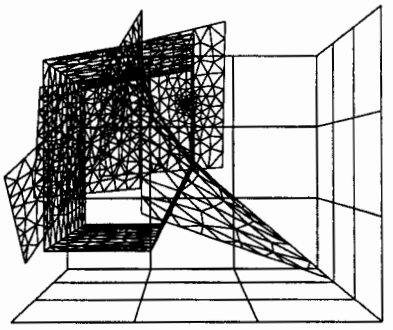


15.6 FRACMAN FRACTURE SURFACES

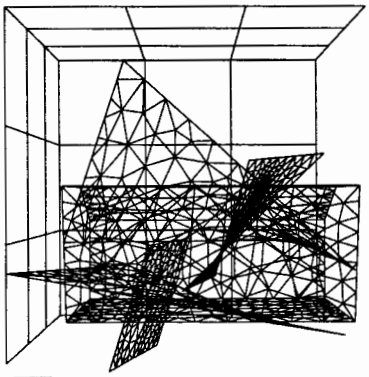
Title	FRACMAN Fracture Surfaces
Purpose/Application	Create Grids from FRACMAN data
Date of Project	April 1995; continuing project
Number of Nodes	na
Number of Elements	na
Description	Develop capibility to insert fracture surfaces defined by FRACMAN into 3D volume. Method maintains FRACMAN plans and intersections and connects to grid it is inserted in.



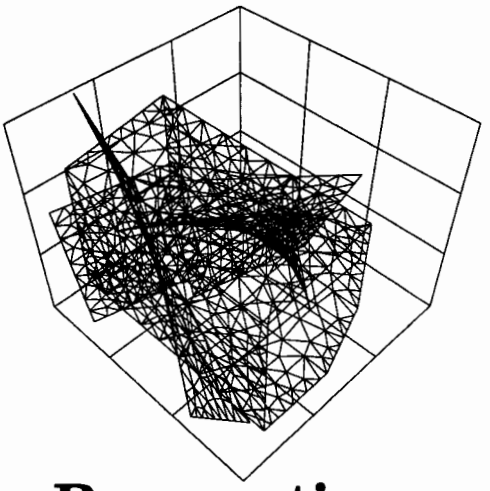
Front



Side



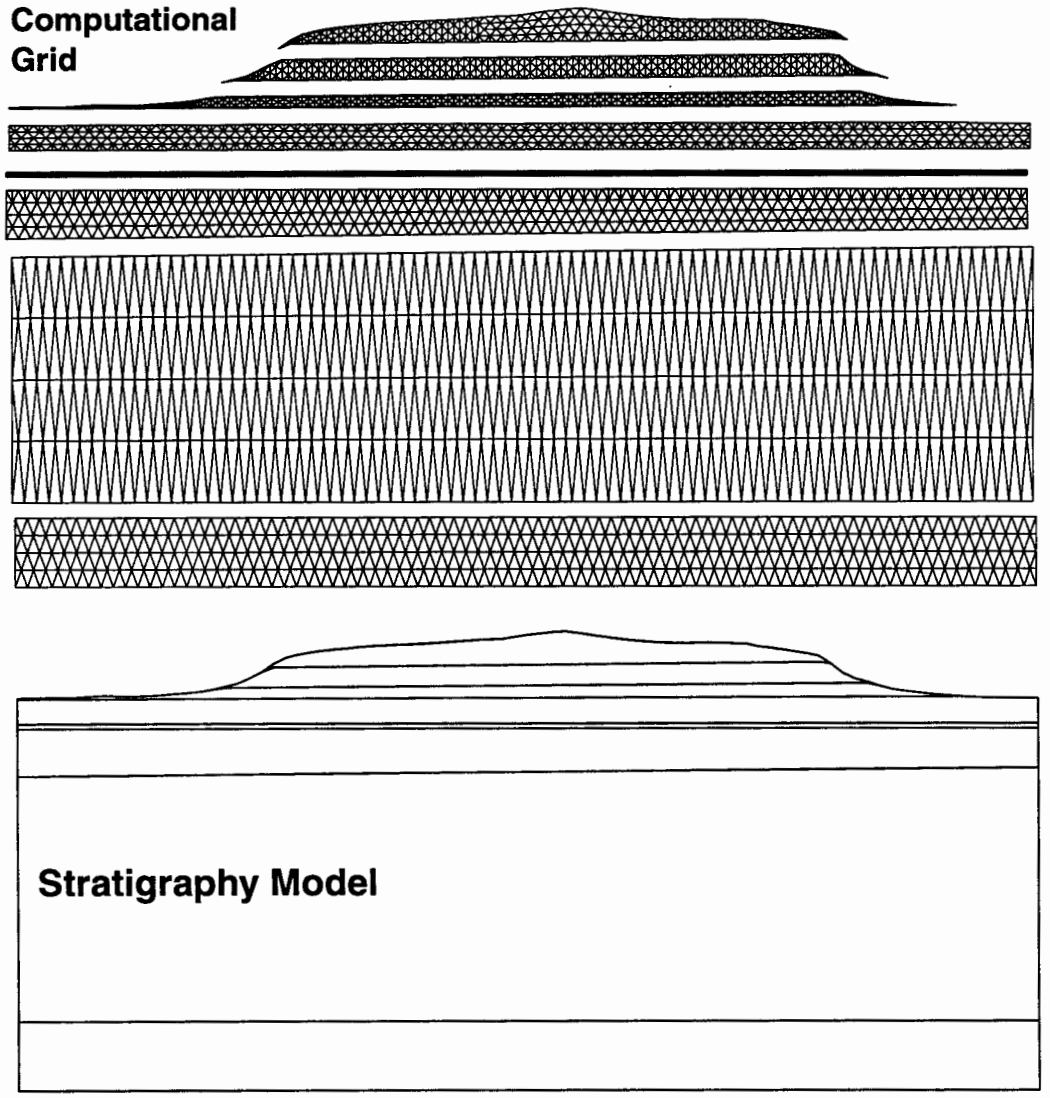
Top



Perspective

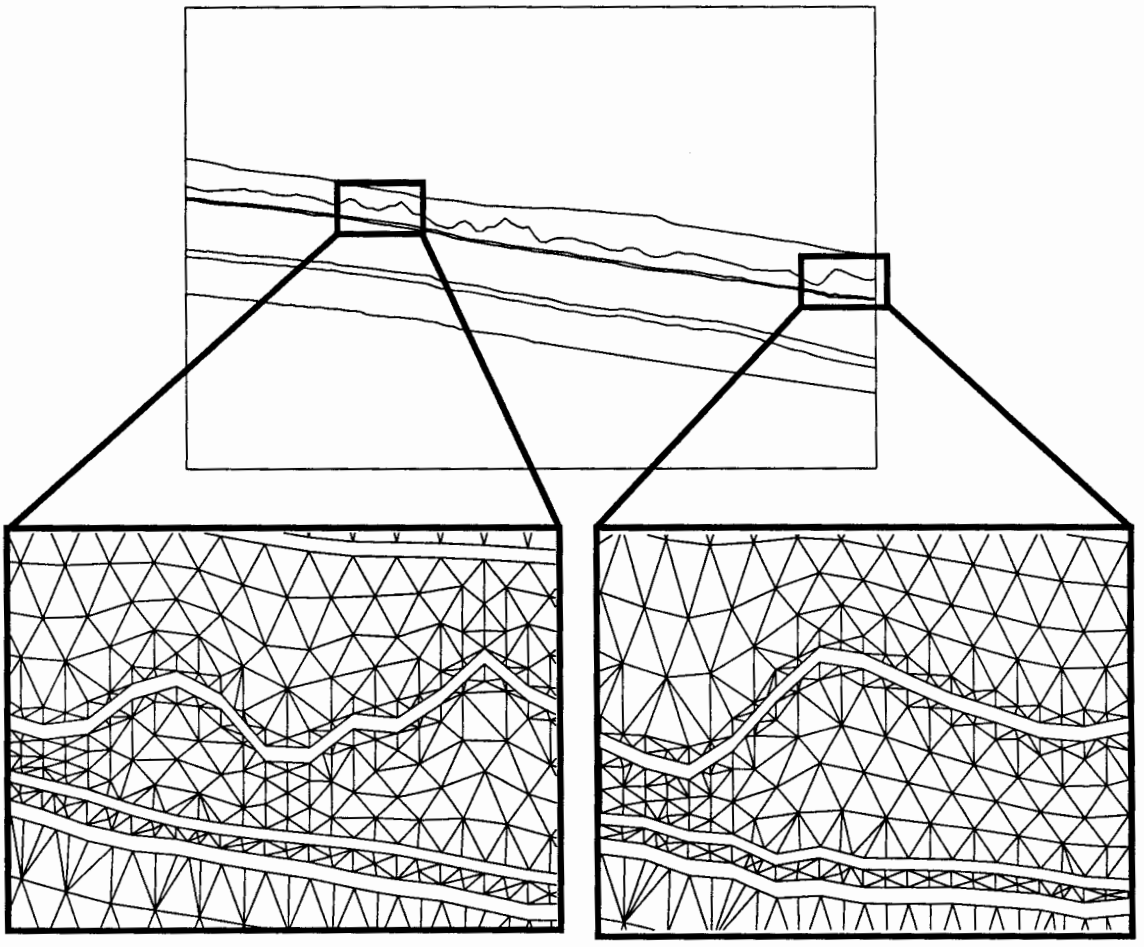
15.7 AREA-G, LOS ALAMOS

Title	Area-G, Los Alamos
Purpose/Application	Model transport of contaminants from buried waste site.
Date of Project	March 1995; continuing
Number of Nodes	2931
Number of Elements	5581
Description	Incorporate mesa topography and stratigraphy as part of model of flow and transport through bedded tuff units.



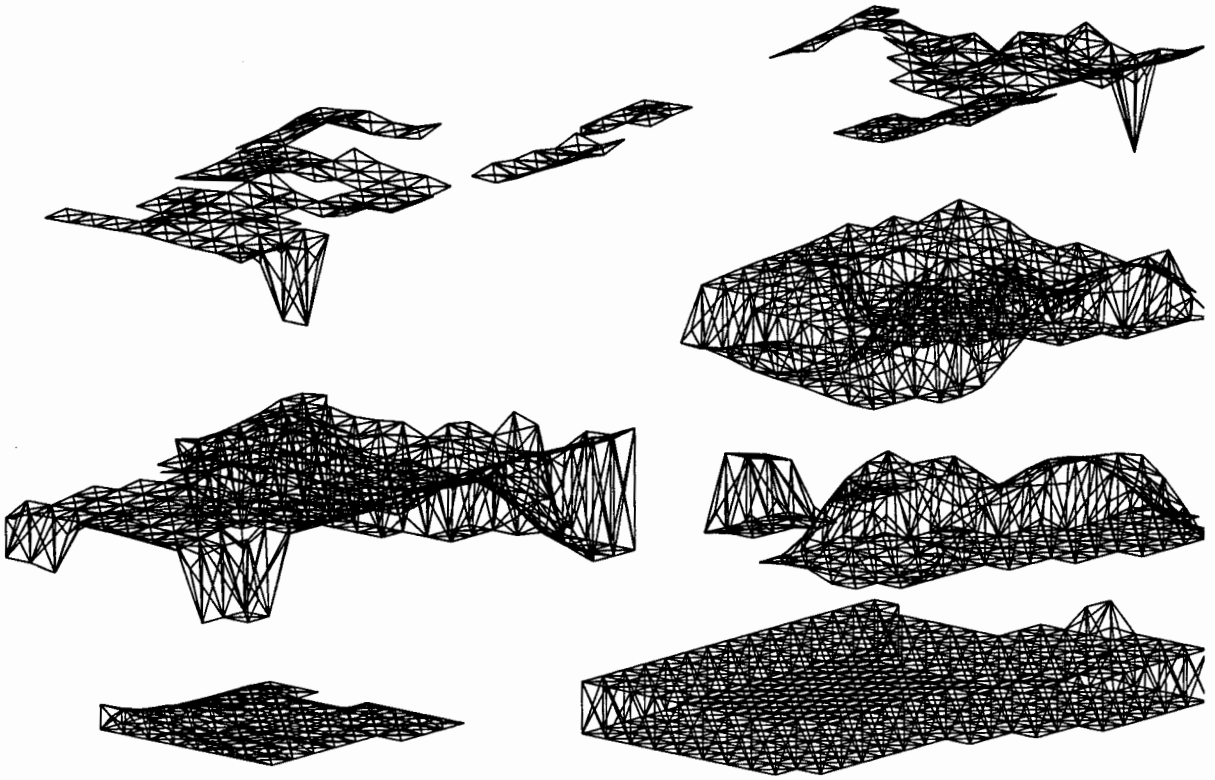
15.8 P-TUNNEL DIGITIZED LAYERING

Title	P-Tunnel Digitized Layering
Purpose/Application	Model P-Tunnel Tracer Tests
Date of Project	August 1995; continuing
Number of Nodes	5837 (Low Resolution), 39833 (High Res.)
Number of Elements	11,332 (Low Resolution), 45328 (High Res.)
Description	Generate 2D computational grid from digitize model of stratigraphy. Photographs of the wall of P-Tunnel were used to digitize coordinates of layer boundaries. Digitized layer boundaries are used as input to GEOMESH.



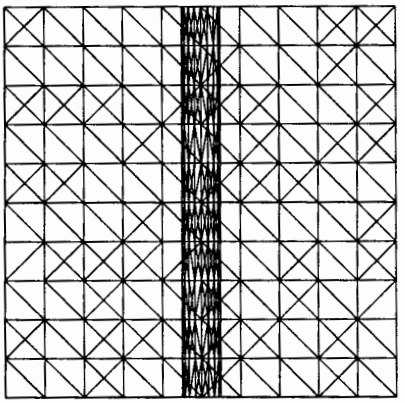
15.9 USGS Saturated Zone Regional Model

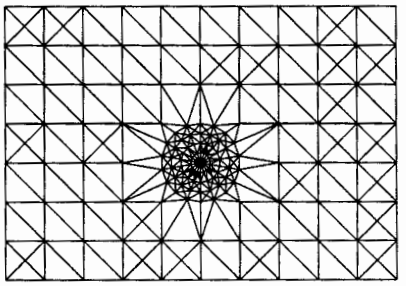
Title	USGS Saturated Zone Regional Model
Purpose/Application	Model saturated zone hydrology in region around Yucca Mountain
Date of Project	August 1995; continuing
Number of Nodes	3603
Number of Elements	19,992
Description	Produce computational grids from stratigraphic model of saturated zone hydrogeology provided by C. Faunt and J. Czarnecki, USGS. This is a small portion of the entire regional model.

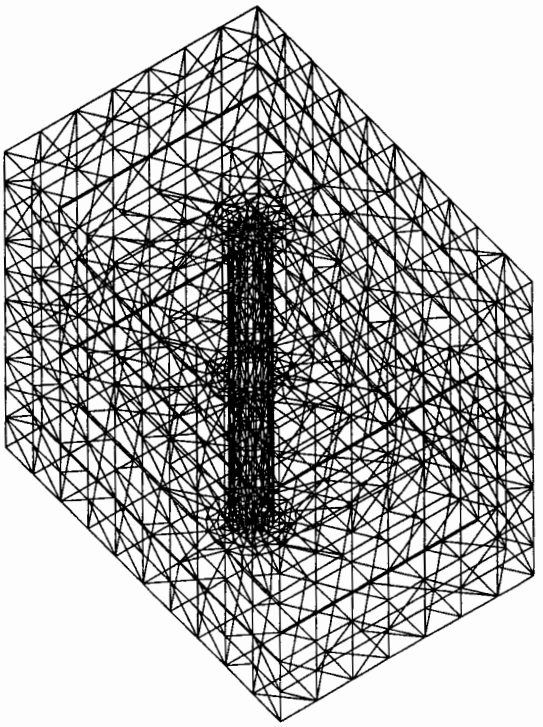


15.10 WELL IN BLOCK

Title	Well in Block
Purpose/Application	Demonstrate addmesh by inserting a well in a uniform block
Date of Project	September 1995
Number of Nodes	2949
Number of Elements	8271
Description	Produce a grid with a cylindrical well in a uniform block.

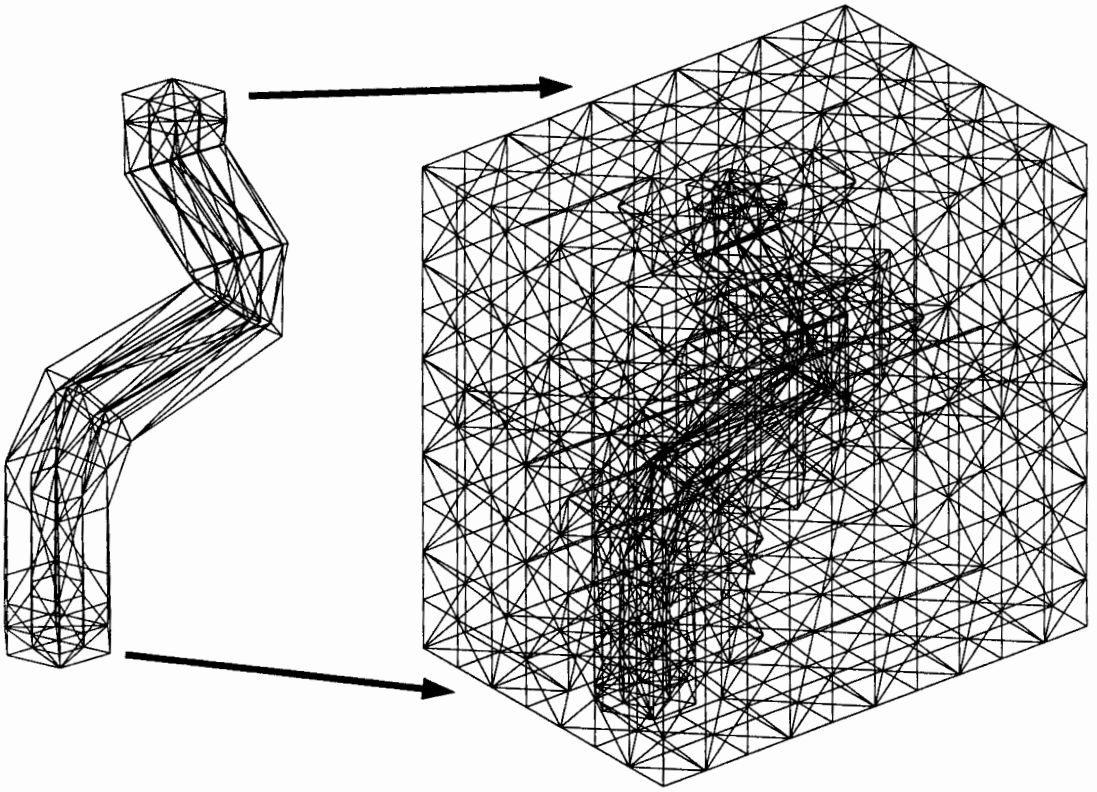






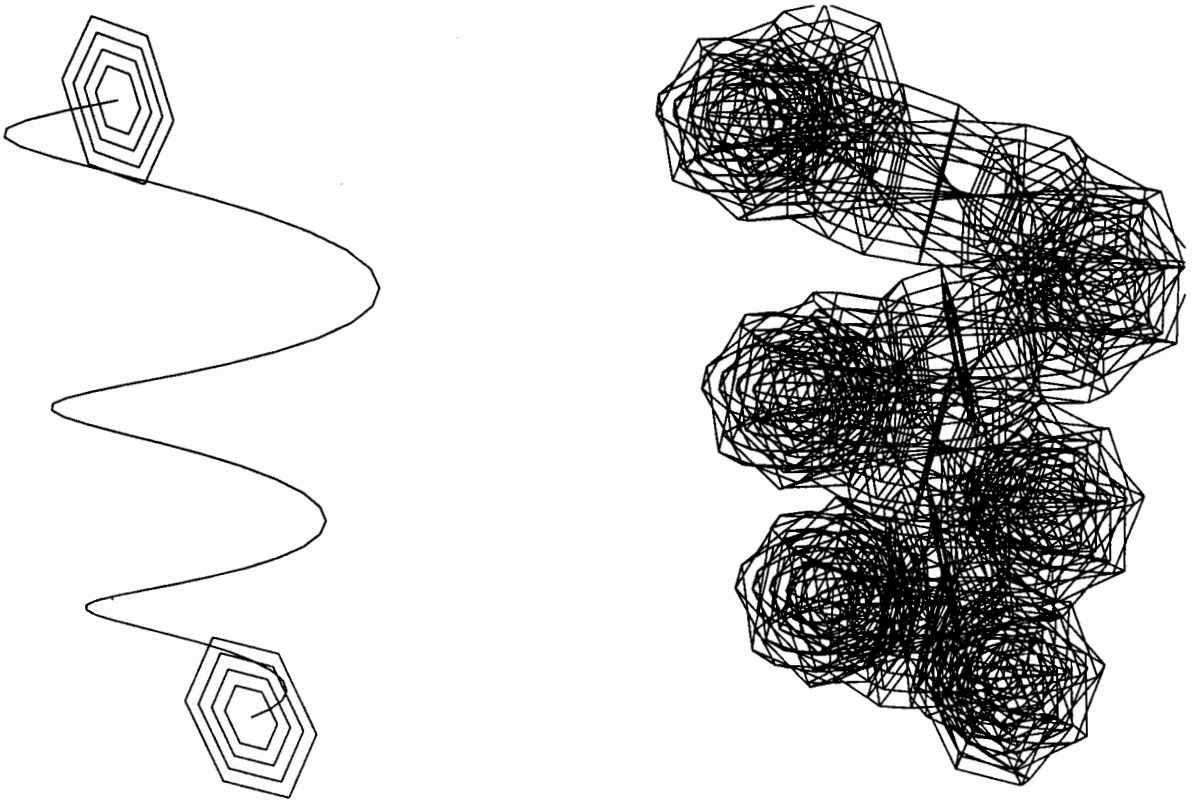
15.11 DEVIATED WELL IN BLOCK

Title	Deviated Well in Block
Purpose/Application	Demonstrate addmesh by inserting a deviated well in a uniform block
Date of Project	June 1995
Number of Nodes	1072
Number of Elements	4689
Description	Insert a well into uniform block by first creating well and block grids. Merge grids by excavating block grid where it intersects well, insert well grid into void and connect grids to form a single grid.



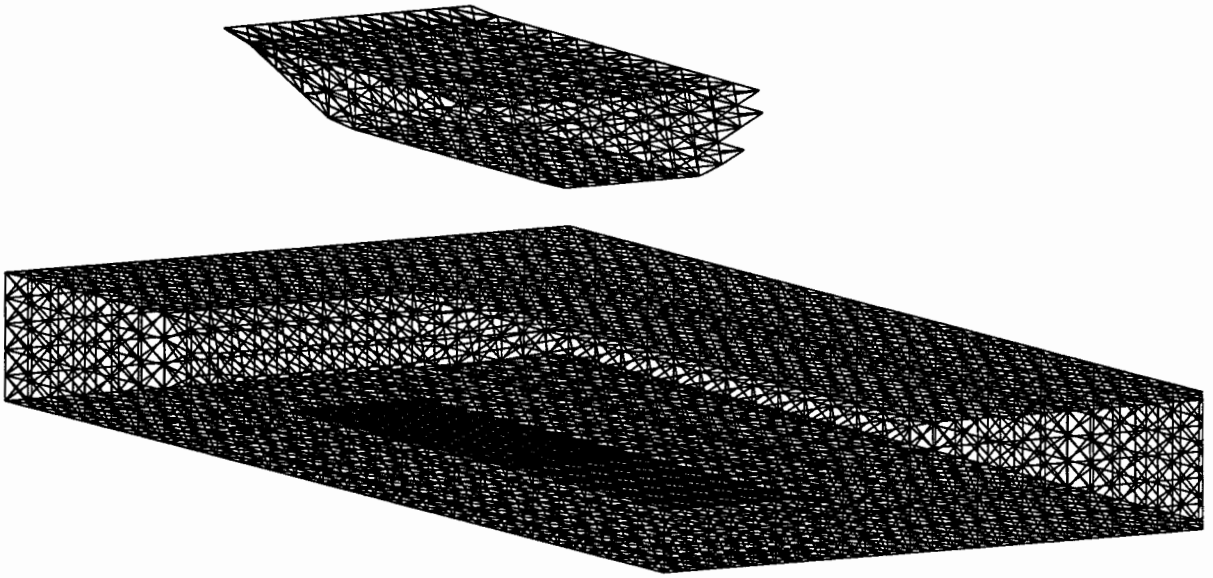
15.12 SPIRAL WELL

Title	Spiral Well
Purpose/Application	Demonstration of generation of well grids from well definition.
Date of Project	May 1995
Number of Nodes	2160
Number of Elements	1770
Description	Create well grid from input of a series of x, y, z triplets.



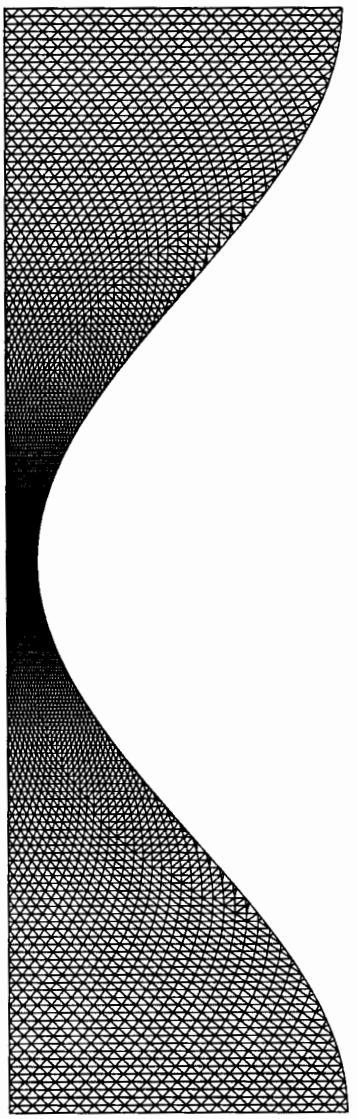
15.13 LANDFILL DESIGN

Title	Landfill Design
Purpose/Application	Model landfill cover and buried waste
Date of Project	July 1995; continuing
Number of Nodes	18,777
Number of Elements	118,173
Description	Data defining 9 layers used to generate grid representing soil, waste, landfill cover and topography used to create one quadrant of lanfill.



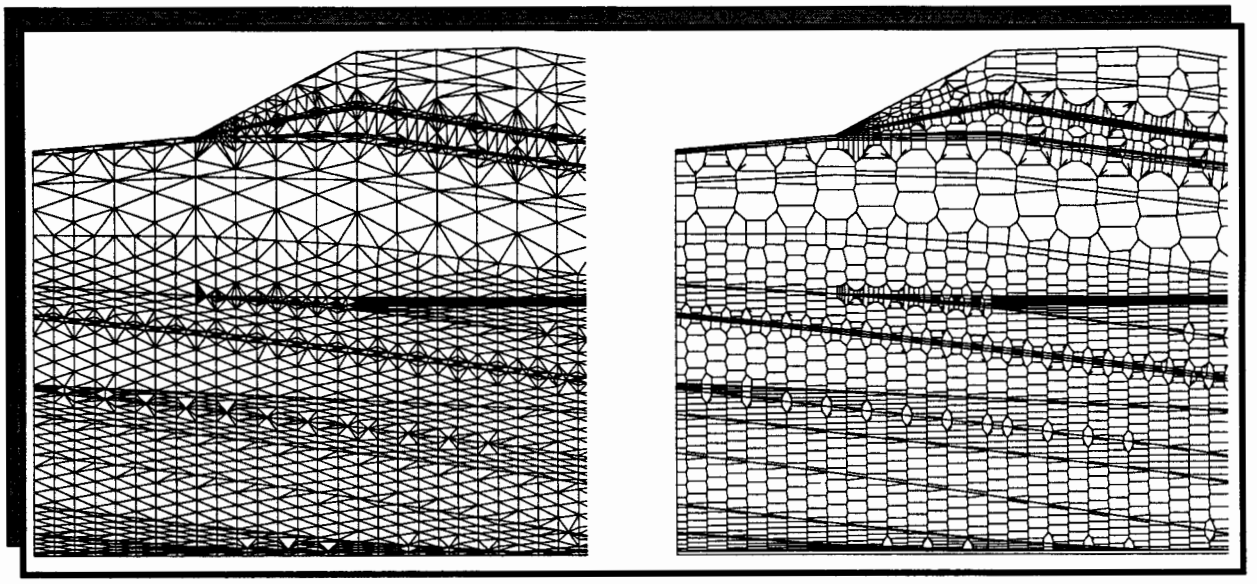
15.14 SINUSOIDAL PORE SPACE

Title	Sinusoidal Pore Space
Purpose/Application	Model flow through idealized sinusoidal fracture
Date of Project	November 1994
Number of Nodes	3437
Number of Elements	6510
Description	Create grid representing sinusoidal fracture. Due to symmetry of the problem only half of the fracture is represented.



LA-UR-95-4143

GEOMESH GRID GENERATION



GEOMESH USER'S MANUAL

Los Alamos National Laboratory
Earth and Environmental Science Division
Geoanalysis Group, EES-5
Carl W. Gable
Terry Cherry
Harold Trease
George A. Zyvoloski

TABLE of CONTENTS

PROGRAM MESHGEN

PROGRAM WELLMESH

1.0 DOCUMENT PURPOSE

This manual is intended for users of the GEOMESH grid generation programs. These programs include; griddr, meshgen, strat2mesh, wellmesh, and x3d grid design commands. The beginning section of this document reviews information relevant to using this document. This is followed by a description of how the programs run. For each of the programs there is a section describing its use, including input and output descriptions and sample problems. For further information on the GEOMESH project, refer to the following documents.

- *GEOMESH Project Description* - overview of capabilities, plans, and grid applications.
- *User's Manual for GEOMESH* - how to create computational grids.
- *GEOMESH Worklist* - running record of work done on the GEOMESH project.
- *X3D User's Manual* - how to run the command language for mesh object commands.
- *GEOMESH Handout* - a short, summarized version of the Project Description.
- *GEOMESH Programmer's Handbook* - code development details for GEOMESH.

It is recommended that first time users read and try the sample geomesh runs in this document. These can be found in the under *"Examples and Sample Runs"*. The document *GEOMESH Project Description* gives a good overview of the programs and contains a catalog of grids that have been generated using GEOMESH.

The programmer's manual is written for GEOMESH programmers. It gives detailed explanations of GEOMESH code variables and data structures.

2.0 DEFINITIONS AND ACRONYMS

AVS - Advanced Visual Systems graphical 3D visualization tool

CMO - Current Mesh Object, code structure defining a mesh

Delaunay - Triangulation of a point distribution

FEHM - Finite element heat and mass transfer code (Zyvoloski, et al. 1988)

FEHMN - YMP version of FEHM (Zyvoloski, et al. 1992)

GEOMESH - Finite element grid generation project for geologic applications

GMV - General Mesh Viewer, a 3D visualization tool for meshes

GMD - Geologic Mesh Data - any possible input into GEOMESH grid generation

GMO - Geologic Mesh Object - GEOMESH version of X3D CMO

LANL - Los Alamos National Laboratory

TINS - Triangulated Irregular Network for a surface representation

YMP - Yucca Mountain Site Characterization Project

UCD - Unstructured Cell Data, AVS data type for finite element meshes

X3D - Command Language for grid generation for physics applications

3.0 DOCUMENT CONVENTIONS

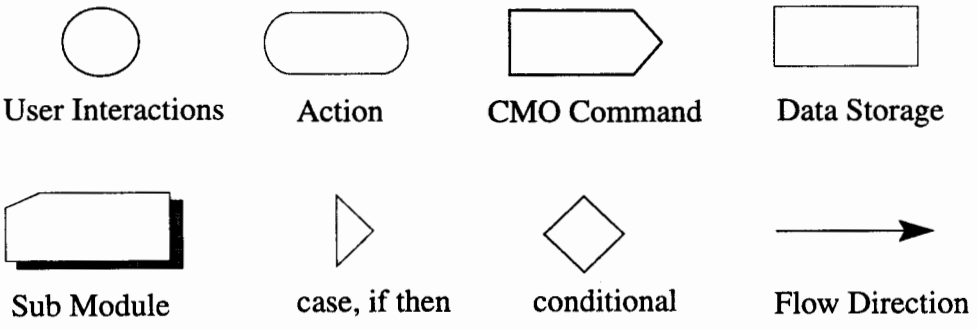
3.1 Typographic Conventions

The following typographic conventions are used in this document:

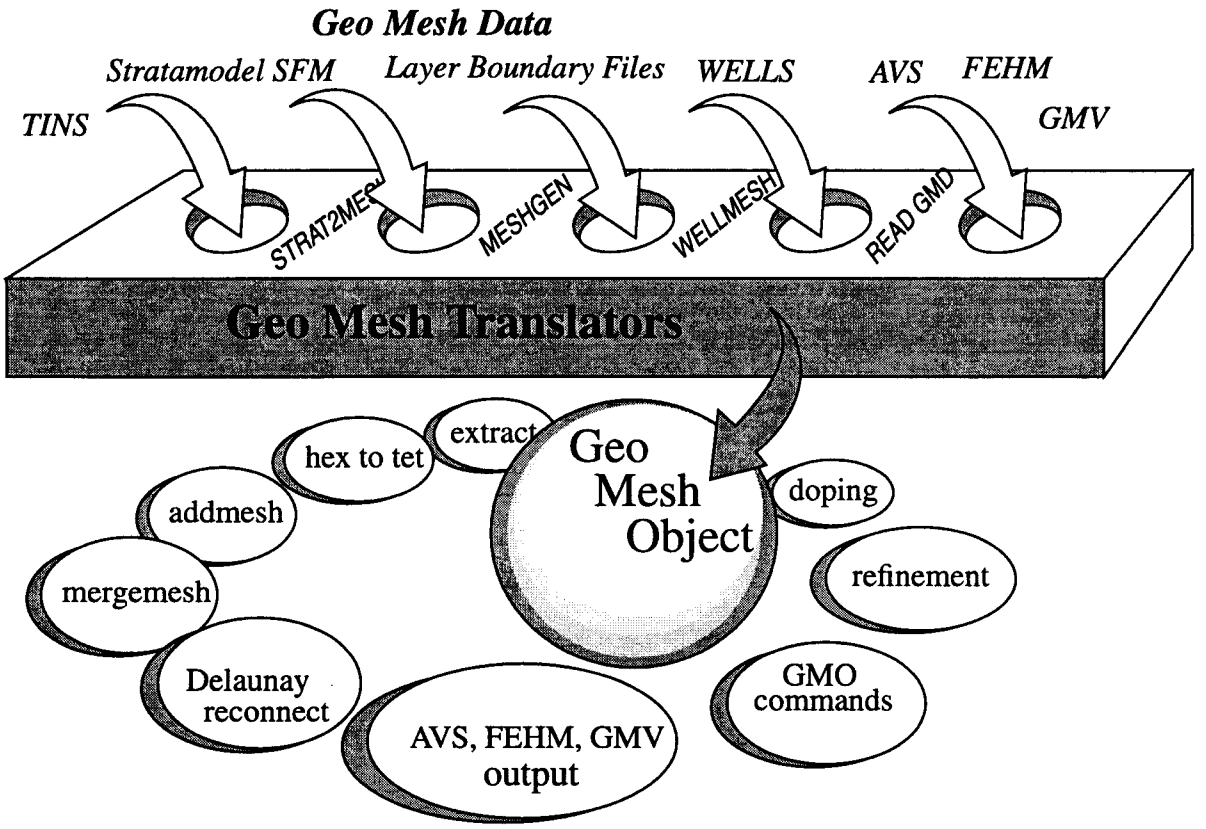
123456789012345	header for file text sample
`sample text from files`	ASCII file text sample
sample text from screen output	screen output to command line
sample code text	text from code files
file names and document titles	text titles
MODULE OR PROGRAM NAMES	code titles
Important Definitions	first definitions of important words
macro names	definitions for program parameters
user input	user typed input at prompt

3.2 Diagram Conventions

The following diagram conventions are used in this document. User interactions indicate location of user input and program exit. The code is developed in modular form. These modules achieve some type of grid design on the mesh data. Actions are important routines that do work for the modules. Data is stored in structures for communication between modules and actions. A current mesh object command (CMO) uses routines from the x3d command language to operate on the mesh object.



4.0 GEOMESH OVERVIEW



The input data needs to be converted into a form that can be used by grid generation routines. Each data set has its own unique format and unique descriptions of its complex geometries. We call this input data, of any form, **Geo Mesh Data (GMD)**. During setup, the program parameters must be defined, the mesh information must be calculated when it is incomplete or missing, and the mesh attributes set. The geologic framework must be accurately and fully represented before grids can be generated.

Eventually, once all the mesh attributes are well defined, the information is stored in a structure called a **Geo Mesh Object (GMO)**. More than one mesh object can be created, but only one will be current. There is no limit to the number of attributes assigned to a GMO, but at the very least, the attributes include:

- mesh object name
- number of nodes and elements in the mesh
- type of elements, triangle, quadrilateral, hexahedral or tetrahedral
- connectivity of elements
- node and element materials
- topographic and geometric dimensions of the mesh
- node coordinates

The code that reads the input mesh data and creates a mesh object, is called a **GMO Translator**. These include modules such as MESHGEN which reads layer boundary files, and STRAT2MESH which reads model information generated with the Stratamodel Stratigraphic Framework Model (SFM). Mesh data can also be input through mesh visualization files, such as those created by Advanced Visual Systems (AVS).

The routines that recognize a mesh object are called **GMO commands**. These are used to design the output grids. They can be used singly or in combinations. Grids can be output during any stage of the design.

5.0 GEOMESH CODE DEVELOPMENT

The code for GEOMESH currently exists as a combination of stand alone executables and commands accessed through a command language for mesh objects called X3D. The modules MESHGEN, WELLMESH and STRAT2MESH are executables, and the remaining translators and mesh object commands are a part of this language. The commands and mesh objects are accessed through library calls or through an interactive command line interpreter. Because GEOMESH is using object - oriented technologies, code development is flexible and modifications are more easily handled. Benefits such as new grid commands and error control is immediate as participating programmers are writing code using a common language and mesh definition.

The GEOMESH project derives considerable benefit from using the command language X3D. This is a program being developed by the Semiconductor Grid Team at Los Alamos National Laboratory which presently has a 7 FTE effort to develop software to aid semiconductor design. The code for X3D is generalized so that any grid application can use the commands, as long as a CMO is created that these commands can use. GEOMESH has been adding and testing commands to X3D that relate to geologic applications.

GEOMESH uses FORTRAN 77, C and C++, with most mathematical calculations in FORTRAN and higher level interface routines in C and C++. The code is written in the ANSI standard but will compile under non-ANSI C. The code has been developed on Sun workstations and is being ported to IBM, HP, SGI and CRAY - the same platforms that are supported for FEHMN software.

Software is developed into a single source and is maintained by using configuration management software to control code changes. The code for GEOMESH is administered under PVCS version manager on Sun workstations.

6.0 GEOMESH CAPABILITIES

GEOMESH has been written to allow flexibility and consistency in designing computational grids. Flexibility is needed to deal with unique grid data and modeling problems. Consistency in the grid routines is needed to insure the accuracy and stability of the numerical solutions used on the final grids. In addition to flexibility and consistency, the code needs to be automated to improve ease of use, and to reduce time spent in the grid generation phase. The following table outlines the capabilities of GEOMESH that contribute towards these goals.

Table I. Capabilities of GEOMESH

I. Generation of unstructured finite element grids.
   - A. 2 or 3 Dimensional.
   - B. uniform, rectilinear, or irregular.
   - C. quadrilateral, triangular, hexahedral or tetrahedral elements.

II. Grids honor the geometric integrity of the original stratigraphic model. Geologic structures include:
   - A. faulting.
   - B. beds that pinch out.
   - C. vertical and deviated wells and tunnels.
   - D. irregular external surfaces defined by topography.
   - E. stratigraphic horizons and lenses.
   - F. dome structures such as salt domes.

III. GMD Translators exist for various geologic framework definitions.
   - A. fracmesh - model discrete fracture networks as part of a 3D volume mesh.
   - B. gridder - simple 3D interactive rectangular grid generator.
   - C. meshgen - reads layer boundary files to create computational grids.
   - D. readavs, readfehm, readtins, readpts - enables input of various data sets.
   - E. strat2mesh - reads a Stratamodel SFM model to create computational grids.
   - F. wellmesh - uses radius and center coordinates of a well or tunnel to create a grid.

IV. GMO Commands are available for grid design.
   - A. CORDSYS - operation in Cartesian, Cylindrical, Spherical coordinate systems.
   - B. Delaunay and Voronoi RECONNECT - tetrahedralization of a point distribution.
   - C. DOPING - interpolation of values from one grid to another.
   - D. HEX TO TET - forms a tetrahedral grid from a hexahedral grid.
   - E. MESH OPERATIONS - enables operations such as add, merge, subset, and extract.
   - F. POINT - generate point distributions.
   - G. REFINEMENT - increased resolution methods, including edges, faces or volumes.
   - H. RESOLUTION - ability to modify grid resolution.

V. Output Grid files at any point of grid design, output files can be used as input for further work.
   - A. AVS - Advanced Visual Systems graphics viewer.
   - B. GMV - Geographic Mesh Viewer.
   - C. X3D - X3D command language output describing the current mesh object.
   - D. FEHM - Finite Element Heat and Mass Transfer output and input files.

7.0 SYSTEM INTERFACE for GEOMESH PROGRAMS

7.1 System-Dependent Features

System dependent features are limited to standard intrinsic math routines in all the GEOMESH code except for the STRAT2MESH module. STRAT2MESH uses libraries licensed by landmark that are limited to the SGI.

7.2 Compiler Requirements

GEOMESH is written in Fortran 77 and C routines. The code has been written in a style that is portable across all platforms. Portions of the code and individual modules have been successfully compiled and run on SGI, SUNOS, SUN SOLARIS, IBM RISC, HP, and Cray computers.

7.3 Hardware Requirements

STAT2MESH is currently the only module requiring special hardware needs, as the program needs to use SGI libraries licensed to LANL by Landmark Graphics.

7.4 Control Sequences or Command Files

None.

7.5 Software Environment

All code is run from the command line, either interactively or in batch form. Refer to the individual sections for environment details for each of the programs. If the FEHM GUI Browser is being used, it provides access to some of the GEOMESH programs. Refer to the FEHM User's Manual for details.

7.6 Installation Instructions

On the EES-5 computer network, no installation is required to run the GEOMESH programs. Remote users will have to use anonymous ftp to obtain executables. Contact the GEOMESH team for executables for the various GEOMESH programs.

8.0 REFERENCES

Fraser D., "Tetrahedral Meshing Considerations for a Three-Dimensional Free-Lagrangian Code", Los Alamos National Laboratory report, LA-UR-88-3707, 1988.

Gable, C.W., Harold Trease, Terry Cherry, Automated Grid Generation From Models of Complex Geologic Structure and Stratigraphy, Third International Conference/ Workshop on Integrating GIS and Environmental Modeling, abstract, LA-UR-95-2665, 1996.

Gable, C.W., Harold Trease, Terry Cherry, Automatic Grid Generation From Complex Models Of Geologic Structure And Stratigraphy, GSA, abstract, LA-UR-95-2482, 1995.

Gable, C.W., George Zyvoloski, Site Scale Modeling of Radionuclide Transport At Yucca Mountain, NV: Grid Generation and Reactive Tracers, LA-UR-94-1041, 1994.

Gable, C.W., G. A. Zyvoloski, Bruce Robinson, Integration of hydrostratagraphic data in reactive transport models, Chapman Conference on Hydrogeologic Processes:

Building and Testing Atomistics- to Basin-Scale Models, Lincoln, New Hampshire, 1994.

George, D.C. and Trease, H.E., "X3D User's Manual", to be published.

Khamayseh, A. and Andrew Kuprat, "Anisotropic Smoothing and Solution Adaption for Unstructured Grids", LA-UR-95-2205, International Journal for Numerical Methods in Engineering, (submitted).

Khamayseh, A., Andrew Kuprat and Frank Ortega, "A Robust Point Location Algorithm for General Polyhedra", (to appear in Computer Aided Geometric Design).

Moridis, G. and K. Pruess, "Flow and Transport Simulations using T2CG1, a package of conjugate gradient solvers for the TOUGH2 family of codes", LBL 36235, April 1995.

Painter, J.W. and Marshall, J.C., "Three-Dimensional Reconnection and Fluxing Algorithms", *Proceedings of the Next Free-Lagrange Conference*, Jackson Lake Lodge, Wyoming, June 3-7, 1990, Springer-Verlag Press, Vol. 395, pp. 139-148.

Sahota, M.S., "Delaunay Tetrahedralization in a Three-Dimensional Free-Lagrangian Multimaterial Code", *Proceedings of the Next Free-Lagrange Conference*, Jackson Lake Lodge, Wyoming, June 3-7,1990, Springer-Verlag Press, Vol. 395, pp. 130-138.

Sahota, M.S., "An Explicit-Implicit Solution of the Hydrodynamic and Radiation Equations", *Proceedings of the Next Free-Lagrange Conference*, Jackson Lake Lodge, Wyoming, June 3-7, 1990, Springer-Verlag Press, Vol. 395, pp. 57-65.

Trease, H.E., "Adaptive Mesh Refinement (AMR) On Unstructured Tetrahedral Grids", to be published.

Trease, H.E. and Dean, S.H., "Thermal Diffusion in the X-7 Three-Dimensional Code", *Proceedings of the Next Free-Lagrange Conference*, Jackson Lake Lodge, Wyoming, June 3-7, 1990, Springer-Verlag Press, Vol. 395, pp. 193-202.

Trease, H.E. "Three-Dimensional Free Lagrangian Hydrodynamics, "*Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics*, Springer-Verlag, Vol. 238, pp. 145-157, 1985.

Trease, H.E. "Three-Dimensional Free Lagrangian Hydrodynamics", *Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics*, Springer-Verlag, Vol. 238, pp. 145-157, 1985.

Trease, H.E. "Parallel Nearest Neighbor Calculations", *Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics*, Springer-Verlag, Vol. 395, pp. 149-156, 1985.

Trease, H.E. "Three-Dimensional Free Lagrangian Hydrodynamics", *Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics*, Springer-Verlag, Vol. 238, pp. 145-157, 1985.

Trease, H.E. (1981), "A Two-Dimensional Free Lagrangian Hydrodynamics Model", Ph.D. Thesis, University of Illinois, Urbana-Champaign.

9.0 USER SUPPORT for GEOMESH PROGRAMS

The GEOMESH team includes Carl Gable, Harold Trease and Terry Cherry. This project also derives considerable benefit from the Semiconductor Grid Team at Los Alamos National Laboratory which presently has a 7 FTE effort to develop software to aid semiconductor design.

Contact:	Carl Gable
Address:	EES-5, Mail Stop F665
	Los Alamos National Laboratory
	Los Alamos, NM 87545
Phone:	(505) 665-3533
FAX:	(505) 665-3687
Email:	gable@lanl.gov

10.0 TRADEMARK ACKNOWLEDGEMENTS

In this documentation for GEOMESH, trademarked commercial products and companies are named. Mention of a commercial company of product does not imply endorsement by EES-5 or LANL. Use for publicity or advertising purposes of information from this publication concerning proprietary products or the tests of such products is not authorized.

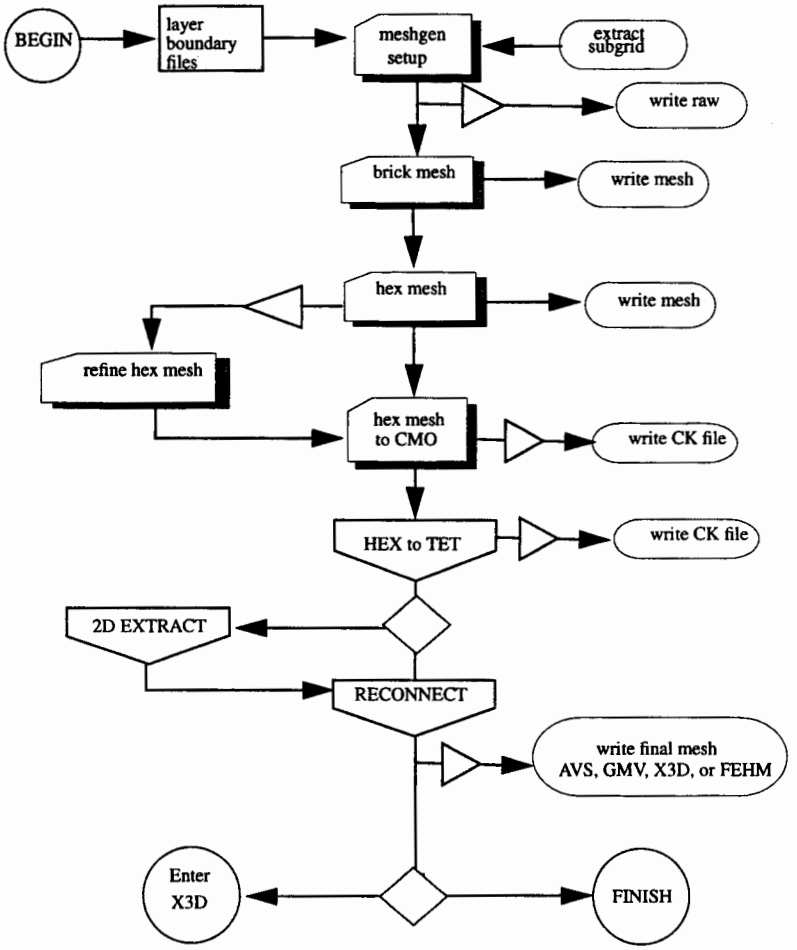
If a trademark symbol does not occur with a trademarked name, note we are using the names only in an editorial fashion with no intention of infringement of the trademark.

PROGRAM MESHGEN

Meshgen is the heart of GEOMESH grid generation and where most of the data sets are processed. It is an executable that runs on two control files, one to define the program parameters, the other lists files to be used as the GMD. Output files include AVS files of the raw data, a hexahedral mesh, and a Delaunay tetrahedral mesh. FEHM input files are also created.

11.0 MESHGEN CODE FLOW

Meshgen begins by creating a 3D rectangular grid of the input data. This module is called Brickmesh. This is followed by the creation of a hex grid which becomes a hex Mesh Object. CMO commands are then used to convert from a hexahedral grid to a tetrahedral grid. If 3D, the grid is reconnected to created a tetrahedral Delaunay grid. If 2D, extraction routines are used to extract a triangular grid from the 3D CMO.



12.0 INPUT DATA FILES

12.1 Control File (*namelist.in*)

This file contains the control parameters for specifying the type, resolution, and region of the input stratigraphy for which a grid should be produced.

EXAMPLE file namelist.in:

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789012
      $input1
istep = i, jstep = i, kstep = i,
buffer = x,
dzcutoff = x, stratcutoff = x,
botz = x, topz = x,
ixrefine = i, iyrefine = i, izrefine = i,
zrefineskew = x,
x0 = x, y0 = x,
dx = x, dy = x,
nx = i, ny = i,
ifsubgrid = i,
xmin = x, xmax = x,
ymin = x, ymax = x,
nlayer = i,
convert_type = 'a',
num_hex_to_tet = i,
io_level = i,
      $end
```

12.2 Control File (*file_list.in*)

This file specifics the name and format of grids which define the stratigraphy. The layers will be sorted by elevation unless the parameter **layer_order** = 1, then the files should be in order from top elevation, to bottom. If the layer order is reversed, use the parameter **layer_order** = -1 to reverse the input file order. Format for file_list.in:

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789012
number_input_grids
file_name(1)    file_format(1)   file_type(1)   material_id(1)
file_name(2)    file_format(2)   file_type(2)   material_id(2)
...             ...              ...            ...
file_name(n)    file_format(n)   file_type(n)   material_id(n)
```

EXAMPLE file:

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789012
2
tcw_02.gpt       3  1  2
ptn_03.gpt       3  1  3
```

12.3 Input Data File (file_format 0)

This is the free format list of z values on an x,y grid. The code for reading these files is:

```
read (13,*) uname
read (13,*) ((x(i,j,nl),i=1,nx),j=1,ny)
```

12.4 Input Data File (file_format 1)

For these files, data is read in a block of nx by ny numbers. These include files that describe the Sandia Yucca Mt. data. The code for reading these files is:

```
read (13,*) uname
do j=1,ny
      read (13,2000) (x(i,j,nl),i=1,nx)
enddo
2000 format(f7.2,12f8.2/7x,12f8.2/7x,12f8.2)
```

12.5 Input Data File (file_format 2)

The code for reading these files is:

```
do j=1,ny
      do i=1,nx
      read(13,*)dumx,dumy,x(i,j,nl)
      enddo
enddo
```

12.6 Input Data File (file_format 3)

The code for reading these files is:

```
fread (13,*) uname
read (13,*) dumchar
do j=1,ny
      do i=1,nx
      read(13,*)dumx,dumy,x(i,j,nl)
      enddo
enddo
```

12.7 Input Data File (file_format 4)

The code for reading these files is:

```
read (13,*) uname
read (13,*) dumchar
do j=1,ny
      do i=1,nx
      read(13,*)dumx,dumx,dumy,x(i,j,nl)
      enddo
enddo
```

13.0 INPUT PARAMETERS

These parameters specify the type, resolution and region of the input stratigraphy for which a grid should be produced and are set in *namelist.in*.

Table II. Input Parameters for Control File *namelist.in*

Entry Parameter	Need	Format	Default	Description
$input1	need			First line in file. (Need 1 space before the $)
istep, **jstep,** **kstep**		int int int	1 1 1	Used to subsample the input at lower resolution. For example, if your database has a resolution of 250 ft in the x and y horizontal directions, but you want a course grid with resolution of 500ft in the x direction, and 750 ft in the y direction set istep = 2, jstep = 3.
buffer		real		This is used to add a layer of constant elevation above the top grid.
dzcutoff, **stratcutoff**		real		Should be set to the same value. They allow the user to specify the thickness at which a bed will be considered to have pinched out and the grid will no longer try to maintain the actual thickness of the input grids. So if dzcutoff = stratcutoff = 10 ft so that when a bed is less that 10 feet thick, it will be pinched out to zero thickness. This can be useful for cases where you may not need to maintain very thin beds as part of your grid, so these parameters can be made larger. However, if you have a case where you have a very thin bed which can have a large effect on your solution, you may wish to make these parameters small. Try different values and have a look at the results. You have to use your judgement about what you want to resolve.
botz, **topz**	optional	real real		Used for specifying a constant elevation for the bottom or top of the grid. If you have a top/bottom grid input file and you specify a value for botz/topz you will get an error message saying use one or the other, but not both.

Table II. Input Parameters for Control File *namelist.in*

Entry Parameter	Need	Format	Default	Description
ixrefine, **iyrefine,** **izrefine**	optional	int int int	1 1 1	Allows the user to specify the horizontal (ixrefine, iyrefine) and vertical (izrefine) refinement of the grids. These are integer input parameters greater than or equal to 1. The code will always refine in the x and y directions the amount specified by the user. However, the internal workings of the code can override the user and increase izrefine to a value greater than the input values but will never reduce it. The reason for this is to keep the unstructured grid from having vertically elongated cells. If you specify fine horizontal resolution or have layers that are very thick compared to the horizontal resolution, the code will automatically insert additional vertical subdivisions in a layer.
zrefineskew	optional	real	1.0	Allows the user to control having nonlinear vertical refinement. 1.0 Linear interpolation between 0 and 1. > 1.0 Results in bias towards smaller z values. < 1.0 Bias towards larger z values
x0, **y0**		real real		Specifies the origin of the horizontal x, y axis of the input grids
dx, **dy**		real real		Specifies the uniform incremental spacing of the input grids.
nx, **ny**		int int		Specifies the dimension of the grid.
ifsubgrid		int	0	RANGE: 0 - 2 Allows the user to use a rectangular (or 2D slice) subset of the original input data. There are two possible ways to take a subset of the data. Specify min, max values of the x and y coordinates or specify min, max grid index values (i.e. if the x grid has 39 nodes you can use that data from node number 20 to 30.) 0 No action 1 Extract subgrid based on NODE index values. 2 Extract subgrid based on COORDINATE values

Table II. Input Parameters for Control File *namelist.in*

Entry Parameter	Need	Format	Default	Description
xmin, xmax, ymin, ymax		real real real real		The parameters used to figure out the extents of the subgrid. When ifsubgrid=1 the min, max values are converted to integers and represent node index values. When ifsubgrid=2, the min, max values are taken in the coordinate system of the input mesh. Note however, that the node index values used are not from the database file itself, but are internal to the code. You can run into some confusion if, for example, you have a database with 100 x values, but you coarsen it using istep=2. Internal to the code, the grid now only has 50 nodes so the center x value is now 25, not 50. It is safest to use ifsubgrid=2 and use absolute x and y coordinate values.
convert_type	optional	char	none	Used to transform the coordinate system. Possible choices are (note the quotes ' ' since this is input as a character string): none No action. 'feet/meters' Convert to meters (assuming feet). 'meters/feet' Convert to feet (assuming meters).
num_hex_to_tet	optional	int	24	Number of times hexahedra is divided into tetrahedras. Choice of 24 or 5: For large grid problems, use 5 as it does not add to the number of nodes in the problem.
io_level io_avs i0_gmv io_x3d io_fehm	optional	int	0 1 0 0 0	io_level is the amount of program output to the screen. io_avs, io_gmv, io_x3d, io_fehm are format types for the grids. The default format is AVS.
layer_order	optional	int	0	Used to control layer order of input files: 0 is the default, layers will be sorted by pre-determined elevation points 1 causes files to not be sorted, order will be top to bottom from file_list.in -1 is not sorted, reversed order of file_list.in
$end				Last line in file. (Need 1 space before the $)

14.0 DATA INITIALIZATION

The following are the default settings for the program parameters.

Table III. Initial (Default) Values

Variable	Value	Variable	Value	Variable	Value
botz	-1.e30	topz	1.e30	istep, jstep,kstep	0
ifsubgrid	0	zrefineskew	1.0	io_level	0
buffer	1.0	dzcutoff	1.0	stratcutoff	1.0
ixrefine	1	iyrefine	1	izrefine	0

15.0 MESHGEN USAGE

MESHGEN runs on the command line with the following input at the prompt:

example%: ***meshgen***

There will be output to the screen while the program runs, when work is complete and there have been no errors, the program will indicate it is done:

example%: meshgen done.

16.0 OUTPUT of FINAL GRIDS

The grid generation process can produce information about the Mesh Object that can be used by finite element, finite volume and integrated finite difference physics codes. Some of the information is used to define finite difference operators (Div, Grad, Curl) while other information is used for setting initial and boundary conditions.

The output files for computational grids contain the following:

- Node coordinates - x, y, z coordinates of every node in the mesh.
- Node color - any number of floating point values associated with each node. This could be the material properties of materials (intrinsic variables, i.e. density, porosity, ...) or could be values derived from a model simulation (i.e. temperature, pressure, ...)
- Node type - every node has an integer associated with it indicating whether it is inside the mesh, outside the mesh or on a material interface. Outside nodes have additional information indicating which of the external faces they are on.
- Node connectivity - list indicating the neighboring nodes connected to any node.

- Element color - every element has an integer indicating which material an element belongs too. This is useful for assigning material properties in physics codes and is used as a constraint during grid reconnection.
- Element connectivity - provides a node list associated with each element describing how nodes are connected to form an element.
- Face connectivity - list associating any face of an element with the corresponding face of the element that shares that face.
- Area coefficients (FEHM *.stor file)
- Zone files - three files providing lists of nodes sorted by: a) material type, b) material interface and c) outside face type. These lists are useful for setting initial and boundary condition.

17.0 OUTPUT FORMATS

Output can be written in various formats. Both AVS and GMV are useful for visualizing the grids, debugging, and for input back into the GMO commands. Shaded views help communicate observations of the model for qualitative interpretation and helps in conceptualizing data results. The ability to rotate and slice through a 3D model can reveal features often missed in 2D.

17.1 AVS UCD

AVS is a data visualization system that allows users to dynamically connect software modules to create data flow networks for 3D interactive rendering and volume visualization. The file format includes number of nodes and elements, node coordinates, node materials, node types, node constraints, element type and connectivity.

17.2 FEHM

The code FEHM (Finite Element Heat & Mass) simulates fluid flow and heat transfer in porous and fractured media. It uses a finite element technique in 2D or 3D, is fully coupled and implicit, and handles nonlinear material properties. If values are interpolated onto a finer FEHM grid, GEOMESH can output a *.ini file which can be used to restart FEHM.

17.3 GMV

GMV is a 3D visualization tool that can process data from meshes. The GMV file format can include the following; number of nodes and elements, node coordinates, cell data, material data, velocities, polygons and tracers. GMV is also used as input into full 3D and virtual reality systems. Both require additional hardware, including glasses, and a glove for virtual reality. These systems improve model evaluation by allowing views inside the grids, making probing and modifications more rapid and intuitive.

17.4 X3D

An output file is written using information from the current mesh object. This file can then be used to restart X3D commands at the point the file was written.

18.0 MESHGEN SAMPLE PROBLEMS

This section will outline, in sequential order, how to run meshgen. For this sample run you will produce a FEHM mesh using Yucca Mt. data. You will first be taken through the steps necessary to produce a simple 2D slice through the data set and then a number of more complicated examples are presented and explained which exercise some of the options of meshgen.

18.1 Setup for Sample Runs

These files are all located on vega.lanl.gov. You will need to login to retrieve the necessary files.

Utilities/Programs you will need access to for this tutorial reside in: /usr/local/bin. You will have to insure this part of your Unix search path or type out the entire path (e.g. usr/local/bin/meshgenmeshgen). They are the following:

- meshgen - mesh generation program that produces 2D/3D/structured/unstructured mesh from input data defining stratigraphy.
- avs_to_fehm - converts AVS graphics file *.inp to FEHM format coordinate and, connectivity file.
- link_ymp_data - creates a symbolic link between Yucca Mountain data set and the current user directory.

18.2 Sample Input Data

To make a grid, create a new directory where you will be working on this project.

mira%***mkdir demo1_2d_x_slice***
mira%***cd demo1_2d_x_slice***

The first thing to do is copy the Yucca Mt. data set into this directory. The script link_ymp_data does this for you.

mira%***link_ymp_data***

This does not produce an actual copy of the data, it just creates a symbolic link from the data in my file space to your directory (see the man page for ln if you want to know more about this Unix utility).

Have a look at a couple of the files. These are ASCII grid files with the first line a character string of up to 72 characters, which will be used as the identifier in the mesh generation software. The second line is also a header line. The rest of the file is x,y,z triplets.

18.3 Sample Control Files

Now we need a couple of input files to control the mesh generation program. Copy them from the demo directory.

mira%***cp /home/cwg/fehm/meshgen/demo/demo1_1d_x_slice/file_list.in file_list.in***
mira%***cp /home/cwg/fehm/meshgen/demo/demo1_1d_x_slice/namelist.in namelist.in***

18.3.1 Control File *file_list.in*

The file *file_list.in* will specify the name and format of your grids which define the stratigraphy, the file *namelist.in* contains the control parameters for specifying the type, resolution and region of the input stratigraphy for which a grid should be produced.

The *file_list.in* file has a header entry, which is an integer (nfile) specifying the number of input grid files. This is followed by nfile lines of the form:

```
123456789012345678901234567890123456789012345678901234567890123456789
FileName (character)          FileFormat (integer)      FileType (integer)
MaterialID (integer)
```

The first 3 lines of the *demo1_2d_x_slice/file_list.in* are:

```
123456789012345678901234567890123456789012345678901234567890123456789
17
tcw_02.gpt       3    1    2
ptn_03.gpt       3    1    3
```

The **FileFormat** of all the input files does not have to be the same, however they must all have the same x,y extents and the same nx, ny grid size.

18.3.2 Control File *namelist.in*

The next file is *namelist.in*, which for this example looks like:

```
123456789012345678901234567890123456789012345678901234567890123456789
     $input1
istep = 1,
jstep = 1,
kstep = 1,
dzcutoff = 10.,
stratcutoff = 10.,
botz = 2600.,
ixrefine = 1,
iyrefine = 1,
izrefine = 1,
zrefineskew = 1.0,
x0 = 557000.00,
y0 = 750000.00,
dx = 250., dy = 250.,
nx = 37, ny = 89,
ifsubgrid = 2,
xmin = 557000.00,
xmax = 566000.00,
ymin =763000.,
ymax =763000.,
```

```
convert_type = 'feet/meters',
        $end
```

The `$input1` at the top and `$end` at the end are important. Don't forget them.

18.4 Parameter Definitions

istep, jstep, kstep are used to subsample the input at lower resolution. For example, if your database has a resolution of 250ft in the x and y horizontal directions, but you want a course grid with resolution of 500ft in the x direction, and 750ft in the y direction set

```
istep =2, jstep = 3
```

The **FileType** is specified as follows:

FileType	Meaning
1	base of stratigraphic unit
2	water level
3	internal surface which may cross stratigraphy but defines hydrogeologic properties (i.e. top of zeolite zone)
4	bed with zero thickness
5	topography
6	bottom surface of data
7	fault plane
8	repository floor (-99 outside repository, floor elevation inside repository)

Types 2 through 8 are considered special layers. In general you will be using **FileType** = 1 and 5.

The **MaterialID** specifies the integer identifier which will be associated with a volume of the mesh. In general you will give each input file a different number, so that the ZONE lists of nodes for FEHM will each specify a different stratigraphic bed. However, it is permissible to give two or more units the same **MaterialID**. You will get an error message however, if you use a **MaterialID** which is larger than nfile.

dzcutoff, stratcutoff should be set to the same value. They allow the user to specify the thickness at which a bed will be considered to have pinched out and the grid will no longer try to maintain the actual thickness of the input grids. So in this example,

```
dzcutoff=stratcutoff =10 ft
```

so that when a bed is less than 10 feet thick, it will be pinched out to zero thickness. This can be useful for cases where you may not need to maintain very thin beds as part of your grid, so these parameters can be made larger. However, if you have a case where you have a very thin bed which can have a large effect on your solution, you may wish to make these parameters small. Try different values and have a look at the results. You have to use your judgement about what you want to resolve.

botz, topz are used for specifying a constant elevation for the bottom or top of the grid. In the above example, **botz** is used to set the bottom elevation of the grid but the top is set using the grid file entry in *file_list.in*:

```
topo_01.gpt      3    5    1
```

Since the **FileType** = 5, the grid generator interprets that grid as the top surface of the grid. If you have a top/bottom grid input file and you specify a value for **botz/topz** you will get an error message saying use one or the other but not both.

ixrefine, iyrefine, izrefine allow the user to specify the horizontal and vertical refinement of the grids.

These are integer input parameters greater than or equal to 1. The code will always refine in the x and y directions the amount specified by the user. However, the internal workings of the code can override the user and increase **izrefine** to a value greater than the input values but will never reduce it. The reason for this is to keep the unstructured grid from having vertically elongated cells. If you specify fine horizontal resolution or have layers that are very thick compared to the horizontal resolution, the code will automatically insert additional vertical subdivision in a layer. You can see this occurring in this example by looking at the screen output during the run. Part of the output looks like:

```
LAYER_REFINEMENT: Test layer thickness and assign
          refinement factor.
 dx     =    76.2000 dy     =    152.400
 ixrefine =   1 iyrefine =   1
 test =     76.2000
Layer refinement factor   1  1
Layer refinement factor   2  1
Layer refinement factor   3  1
(....)
Layer refinement factor   13  3
Layer refinement factor   14  1
Layer refinement factor   15  1
Layer refinement factor   16  2
```

This output is indicating that although **izrefine** is set to 1, layer 13 is subdivided 3 times, and layer 16 has 2 additional subdivisions.

zrefineskew is not necessary in this input file since it will default to 1. It's purpose is to allow the user to control having nonlinear vertical refinement.

zrefineskew	Meaning
1.0	1.0 results in linear interpolation between 0-1
> 1.0	results in bias towards smaller z values
< 1.0	bias towards larger z values
and > 0	

This is demonstrated in another demo (demo5_2d_x_slice_skew_z_up, demo5_2d_x_slice_skew_z_down).

In general, the input grid files do not determine the horizontal coordinate system of the problem. This is determined from the following 6 parameters.

x0, y0 specify the origin of the horizontal x,y axis of the input grids.

dx, dy specify the uniform incremental spacing of the grid.

nx, ny specify the dimension of the grid.

ifsubgrid allows the user to use a rectangular (or 2D slice) subset of the original input data. There are two possible ways to take a subset of the data. Specify min, max values of the x and y coordinates or specify min, max grid index values (i.e. if the x grid has 39 nodes you can use that data from node

number 20 to 30.) This is specified as follows:

ifsubgrid	Meaning
0	no action (default)
1	Extract subgrid based on NODE index values
2	Extract subgrid based on COORDINATE values

xmin, xmax, ymin, ymax are the parameters used to figure out the extents of the subgrid. As noted above, when **ifsubgrid**=1 the min, max values are converted to integers and represent node index values. When **ifsubgrid**=2, the min, max values are taken in the coordinate system of the input mesh.

Note however, that the node index values used are not from the database file itself, but are internal to the code. You can run into some confusion if, for example, you have a database with 100 x values, but you coarsen it using istep=2. Internal to the code, the grid now only has 50 nodes so the center x value is now 25, not 50. It is safest to use **ifsubgrid=2** and use absolute x and y coordinate values.

convert_type is used to transform the coordinate system. For example, the Yucca Mt. database is in feet but FEHM operates in meters. This parameter allows one to make this conversion. Possible choices are (note the quotes ' ' since this is input as a character string):

convert_type	Meaning
none	(default) no action
'feet/meters',	convert to meters (assuming feet)
'meters/feet'	convert to feet (assuming meters)

When the calculation is done, the directory will include the following files:

```
-rw-r--r-- 1 gable     569 Jul  7 08:45 bottom_plane.zones
-rw-r--r-- 1 gable   39168 Jul  7 08:46 log_meshgen
-rw-r--r-- 1 gable  240431 Jul  7 08:46 mesh_2d_x_slice.inp
-rw-r--r-- 1 gable    5931 Jul  7 08:46 mesh_2d_x_slice.zones
rw-r--r-- 1 gable     359 Jul  7 08:46 mesh_2d_x_slice_bottom.zones
-rw-r--r-- 1 gable     423 Jul  7 08:46 mesh_2d_x_slice_top.zones
-rw-r--r-- 1 gable  104246 Jul  7 08:43 mesh_bric10001.inp
-rw-r--r-- 1 gable    3487 Jul  7 08:43 mesh_brick.zones
```

```
-rw-r--r-- 1 gable    171 Jul  7 08:43 mesh_brick_bottom.zones
-rw-r--r-- 1 gable    339 Jul  7 08:43 mesh_brick_repository.zones
-rw-r--r-- 1 gable    218 Jul  7 08:43 mesh_brick_top.zones
-rw-r--r-- 1 gable 206485 Jul  7 08:43 mesh_raw10001.inp
-rw-r--r-- 1 gable 948364 Jul  7 08:45 tet_delaunay.inp
-rw-r--r-- 1 gable 182143 Jul  7 08:43 mesh_ucd10001.inp
-rw-r--r-- 1 gable 251197 Jul  7 08:43 mesh_ucd10002.inp
```

18.5 Sample Output Files

Files that end in *.inp* contain the finite element mesh in AVS UCD format (see the AVS manual for a definition of the file format).

Files that end in *.zones* are in the FEHM ZONE definition format. The information can be cut and pasted into your FEHM input file.

Files that end in *.stor* are in the FEHM STOR definition format.

Files which contain the string *raw* can be used to visualize the raw, unaltered input database planes used for grid generation.

Files which contain the string *brick* contain information relevant to the structured rectangular grids.

Files that contain the string *ucd* are grids of irregularly shaped hexahedra. These can be used for visualization purposes to debug and understand your grid, however they are not appropriate for use in a FEHM calculation.

Files that contain the string *tet* are tetrahedral grids. The only one appropriate for use in calculations is called *tet_delaunay.inp*

If io_level is greater than 0 for debugging, additional files will be generated at important check points, these begin with the suffix *CK*.

18.6 Conversion of AVS format files to FEHM format files

In order to use the grids produced with meshgen, they must be converted from AVS file format to FEHM file format. This is accomplished with the program *avs_to_fehm*. The session logged below should be self explanatory.

```
mira% /home/cwg/fehm/util/a.out
enter AVS filename(32 character maximum) mesh_bric10001.inp
enter FEHM filename(32 character maximum) mesh_bric10001.fehm
```

18.7 Additional Sample MESHGEN Runs

There are more examples in the following directories:

```
/home/cwg/fehm/meshgen/demo/demo1_2d_x_slice
/home/cwg/fehm/meshgen/demo/demo2_2d_y_slice
/home/cwg/fehm/meshgen/demo/demo3_2d_x_slice_refine_brick_mesh
/home/cwg/fehm/meshgen/demo/demo4_2d_x_slice_course_mesh
/home/cwg/fehm/meshgen/demo/demo5_2d_x_slice_skew_z_up
/home/cwg/fehm/meshgen/demo/demo6_2d_x_slice_skew_z_down
home/cwg/fehm/meshgen/demo/demo7_3d_2000ft_2000ft_250ftcenters
home/cwg/fehm/meshgen/demo/demo8_3d_2000ft_2000ft_refined
home/cwg/fehm/meshgen/demo/demo2_2d_y_slice
```

/home/cwg/fehm/meshgen/demo/demo1_2d_x_slice

1080 Nodes, 174 CPU, Sun Sparc 10

The first demo in the above list is the mesh generation just covered. The next one differs from the first demo in that it is a y slice through the data rather than an x slice. The difference in the *namelist.in file* is:

```
 ifsubgrid = 1,
xmin = 10.00,
xmax = 10.00,
ymin = 20.,
ymax = 40.,
```

The **ifsubgrid** = 1 now takes the x/y min/max values as node values so this mesh will be one node wide in the x direction using the 10th x node in the data base and uses the 20-40 nodes in the y direction.

/home/cwg/fehm/meshgen/demo/demo3_2d_x_slice_refine_brick_mesh

1080 Nodes, 174 sec CPU, Sun Sparc 10

This shows how you can trick the software into increasing the number of vertical elements in the rectangular structured 'brick' mesh. The unstructured tetrahedral and triangular mesh are unchanged from demo1. The software will produce the same number of vertical subdivision in the brick mesh as there are input files, therefore you can input the same file repeatedly to increase the number of input files. In this example, *namelist.in* is the same as demo1, but *file_list.in* is different. The file *trw_16.gpt* has been entered 18 times. This increased the total number of files to 34, so the header is changed from 17 in demo1 to 34 in demo3.

/home/cwg/fehm/meshgen/demo/demo4_2d_x_slice_course_mesh

395 Nodes, 55 sec CPU, Sun Sparc 10

This example shows how the horizontal resolution of the mesh can be coarsened. The *namelist.in* file is the same as demo1 except not **istep**=3 rather than 1. This results in horizontal grid spacing of 750ft rather than 250ft. The final 2D triangular mesh in demo1 has 395 nodes while demo1 has 1079 nodes..

/home/cwg/fehm/meshgen/demo/demo5_2d_x_slice_skew_z_up

530 Nodes, 84 sec CPU, Sun Sparc 10

This show how the user can force the vertical refinement of layers to be skewed toward the tops of the layers. Again, the input file *namelist.in* is the same as demo1 except for the following change:

```
izrefine = 5,
zrefineskew = 0.3,
```

This forces 5 vertical subdivisions of each layer with the subdivisions skewed toward the top of each layer. The closer **zrefineskew** is to 0.0, the more the subdivisions are pushed up. **zrefineskew** = 1.0, which is the default results in linear subdivision of the layers.

/home/cwg/fehm/meshgen/demo/demo6_2d_x_slice_skew_z_down

581 Nodes, 79 sec CPU, Sun Sparc 10

This is similar to demo5, however **zrefineskew**=3.0 so the subdivisions are pushed towards the bottom of each layer.

home / cwg / fehm / meshgen / demo / demo7_3d_2000ft_2000ft_250ftcenters

3487 Nodes, 254 sec CPU, Sun Sparc 10

This produces a 3D mesh, 2000ft on a side with 250ft horizontal resolution.

home / cwg / fehm / meshgen / demo / demo8_3d_2000ft_2000ft_refinedx_coursey

18,953 Nodes, 1963 sec CPU, Sun Sparc 10

This produces a 3D mesh, 2000ft on a side with horizontal resolution 3 times finer (apx. 83ft) than the input grids.

19.0 MESHGEN ERROR PROCESSING

Two types of errors can occur. Those that are non-fatal will not terminate the program. Warnings will be printed to screen. The following are errors which can cause meshgen to abort.

Table IV. Error Conditions Resulting in Program Termination

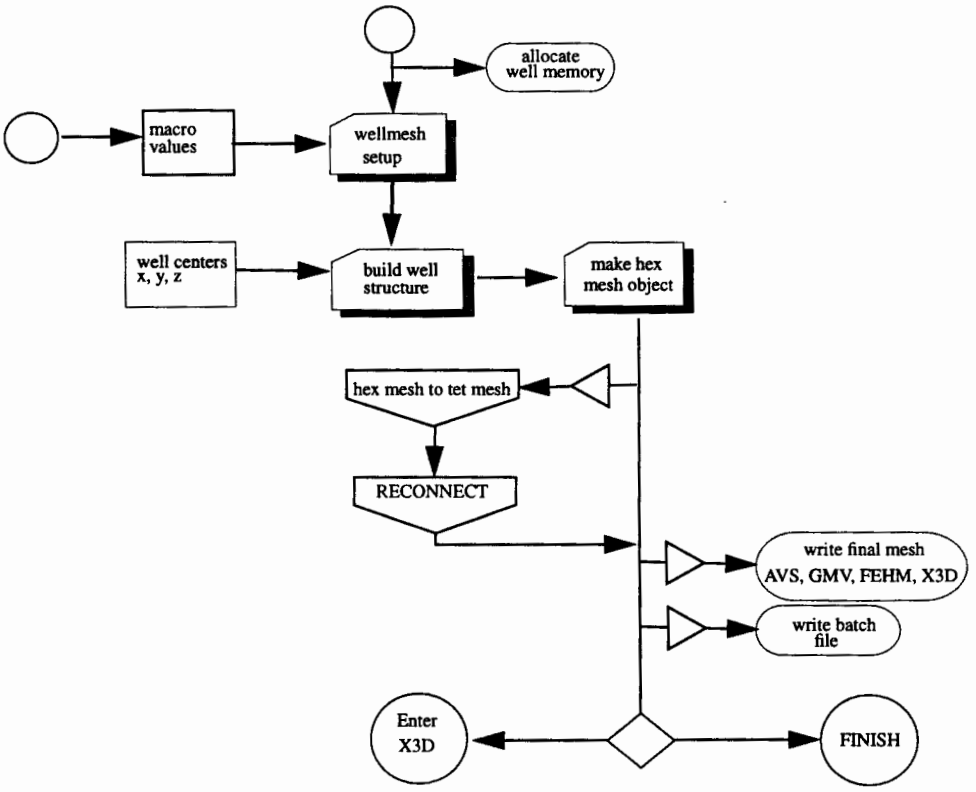
GEO>> MREAD>> ERROR: Problem Reading Input File
Input file containing macro definitions has been opened but could not be read.
GEO MESHGEN>> FILES: "file" does not exist.
This error occurs if the control files or a data file is not found.
GEO>> ERROR: Exiting "routine()", program stopped.
If a fatal error occurs during a "routine()", the program will stop. Error messages will have been printed by the "routine()" that failed.
ERROR >> 'number' > 'constant' max
The number is larger than the number allowed.

PROGRAM WELLMESH

Wellmesh creates a grid representation of wells or tunnels along an arbitrary path and with a given radius. This path is described by its center coordinates x, y, and z. The mesh can be hexahedral or tetrahedral. The resulting mesh description is written in AVS UCD, X3D, GMV, or FEHM formats. This module is used with ADDMESH to embed wells within a grid. This allows the accurate representation of gas and fluid flow in the vicinity of the wells.

20.0 WELLMESH PROGRAM FLOW

As shown below, WELLMESH first defines all program parameters as defined by the macro commands. This is done by a macro *.dat* file, or through an interactive prompt. A mesh structure is built around the center coordinates and then copied into a current mesh object (CMO). The CMO has commands that write the requested files. The hexahedral mesh can be transformed into a tetrahedral mesh, with the appropriate output files being written.



21.0 WELLMESH OPTIONS

WELLMESH defines program parameters through a set of macro commands. These can be set either interactively by prompt, or by a macro *.dat* file.

Table V. Capabilities of WELLMESH with Macro Command References

VI. Grids of wells or tunnels.
- A. Input center coordinates with *.well* file or calculate center axis from top coordinate to a given depth **(coo)rdinate**.
- B. Explicit or calculated segment sizes **(seg)ment**.
- C. Hexahedral or tetrahedral elements **(ele)ment**.
- D. Radial refinement of well or tunnel **(rad)ius**.
- E. Integer id assignment to well layers **(mat)erial**.
- F. Material assigned by well layer or stratigraphy **(typ)e**.

VII. Output formats available **(out)put**.
- A. AVS UCD
- B. GMV
- C. X3D
- D. FEHM

VIII. User Interface Options
- A. Interactive, cycles through all macro definitions **(prompt)**.
- B. Batch mode for automated runs.
- C. Online help for macro definitions **(help)**.

22.0 WELLMESH MACROS

These macros are used to define global values for the program wellmesh. They are defined in either a macro *.dat* file or interactively at the prompt.

Macro conventions:

< > is a place holder for a value, int, real or "character string".

[ ] means the usage on the macro line is optional.

"no" is used to turn toggles off, otherwise the toggle is turned on.

Table VI. Macro Values for WELLMESH

macro	argument syntax	description
coordinate	<"file name">Strat file has an integer material after each coordinate. The default is "coord".	File containing center coordinates of well.
	<real x> <real y> <real z> <real depth>	Top coordinates of well and depth. Used along with macro segment to create a vertical well without center coordinates.
radius	<real length> [int refinement]	This macro allows the radius length to be defined and/or the radius to be divided into layers up to 50. If not used, radius will be 1.0 and there will be no radial refinement.
segment	<int refinement> [real rounded length] [[no]"connect"]	This macro allows the distance between coordinates to be rounded to a chosen length. It also can be used to divide this distance into additional segments. If not used, there will be no vertical refinement and the distance between coordinates will be determined by the input coordinates.
element	["hex"] <[no] tet> [int 5 or 24]	The well mesh is created with hex elements. This macro can be used to set or unset the tet toggle. If on, the mesh will be converted from hex elements to tet elements.
material	<int material id> <material id> ...	Well layers start from the inside and radiate outward. The first material id will be assigned to the inside layer, the next id will be assigned to the next layer out. The default material is 1
stop	no arguments	Used in macro *.dat* file to end reading of macros or to stop interactive session.
prompt	no arguments	Turns interactive session on so that all macros will be prompted at least once.
help	no arguments	Shows a list of all available macros.
project	< "project name">	The project name will form the rootname to be used in file names. If not set on command line or in macro .dat file, the project name will be "wellmesh".
output	< [no] avs > < [no] x3d > < [no] gmv > < [no] sav > <[no] log > < [no] debug >	These are toggles that can be set on or off with the output macro. If none are selected, there will be no files written. If sav is set, the macro setup will be saved to file - this file can then be used as the macro .dat file. The log file contains program output. If debug is set, program output will be verbose.

23.0 WELLMESH VARIABLES

Wellmesh variables are initialized to default values before the macro commands are used. These variables are shown in the table below.

Table VII. Initial (Default) Values for WELLMESH Variables

Variable	Value	Variable	Value	Variable	Value
nwellnode	12000	nwellelm	24000	nwellseg	500
maxlayer	50	nlayer	1		
well_type	"coord"				
nlen	1	nrad	1	radius	1.0
midpt	0	hex	1	tet	0
ioavs	0	iogmv	0	iox3d	0
iolog	0	iosav	0	debug	0

Once the wellmesh parameters have been setup, the well structure is created. Conceptually, the well is made up of segments, each segment center being the distance from one coordinate to the next. The well is built from the bottom coordinate up to the top coordinate.

Each well segment is a hex tube object. The hex tube is created using 6 elements that are eight noded quadrilateral polyhedrons. The basic hex tube has 6 elements and 24 nodes. The top inside row of nodes join to become the top center node at (0,0,1) and the bottom inside row become the bottom center node at origin (0,0,0). The center of the hex_tube therefore has 10 redundant nodes. The hex_tube can be refined along the z or y axis. nlength is the number of divisions along the length (z axis). nradius is the number of divisions along the radius (y axis). Segment totals can be calculated from nlength and nradius:

nnode = *6*(*nlength*+1)*(*nradius*+1)*
nelements = *6* * nradius * nlength
nrows = *(*nlength*+1)*(*nradius*+1)*
nlist*(connectivity list)* = nelements**8*

To construct the final well mesh, each segment in the well is created, transformed to its proper position and size, then attached to the previous segment. Once all segments are attached, a hex well has been created.

24.0 INPUT DATA FILES for WELLMESH

There are two possible input files for wellmesh. The first is the macro input file (*.dat*). The second is a file containing the center coordinates of the desired well. The macro input file is fully described in the "User Interface" section at the beginning of this document. The following are descriptions of the input files for wellmesh.

24.1 Macro Input (*.dat*)

24.1.1 Use by Program

This file contains macro commands which define program data values. If this file is not present at run time, the user will be prompted with each macro command available for wellmesh.

The default name for the macro file is wellmesh.dat. The macro file can also be found from the command line. For the command line...

example% ***wellmesh spiral***

the program wellmesh will look for the macro file *spiral.dat*. "Spiral" will be defined as the project name and will be used for the root name for output files. So if a hex AVS mesh is requested, it will be called *spiral_hex.inp*.

24.2 Well Coordinates (.well)

24.2.1 Use by Program

If the well description is given by its center coordinates, then a *.well* input file will be needed. Center coordinates can be calculated from a top x, y, and z coordinate and the vertical depth of the well. Otherwise, an ASCII file listing the center x, y, z coordinates from the top to the bottom is required. This file has the suffix .well. Units should be the same as the intended background geometry.

24.2.2 Content

The file contains a list of x, y, and z coordinates locations for the center of the well. The coordinates are listed from the highest elevation to lowest, in the format of x, y, and z on each line. The following example is *spiral.well*. The top of the well are the coordinates (1.0, 0.0, 5.0) and the ending coordinates are (0.990747, -2.211171, 12.114924). Note: In most cases the last z coordinate will be lower than the first z coordinate.

```
12345678901234567890123456789012345678901234567890
1.000000 0.000000 5.000000
0.969775 0.299986 5.075565
0.850471 0.581838 5.152273
(...)

-0.418086 -2.313907 11.756874
0.288690 -2.369389 11.934557
0.990747 -2.211171 12.114924
```

25.0 OUTPUT for WELLMESH

Wellmesh will write the resulting mesh in AVS, GMV, or X3D formats. These can be set with the **output** macro.

25.1 Hex Well Mesh Files (*_hex.*)

These files contain the hex version of the well mesh. They end in the suffix indicating the format type. These suffixes include "inp", "gmv", and "x3d'.

25.2 Tet Well Mesh Files (*_tet.*)

These files contain the tet version of the well mesh. It is written after converting the hex mesh to a tet 24. The files end with a suffix indicating the format type. These suffixes include ".inp", ".gmv", and ".x3d'.

25.3 Tet Well Mesh Files (*_tet5.*)

These files contain the hex version of the well mesh. It is written after converting the hex mesh to a tet 5. The files end in the suffix indicating the format type. These suffixes include ".inp", ".gmv", and ".x3d'.

25.4 Saved Macro File (*.dat.sav*)

The final macro values can be saved to file. This file can be renamed to end in the *.dat* suffix then used as a macro file to run the program in batch mode. When the command

example% ***wellmesh spiral***

is given, the output file *spiral.dat.sav* will be written. This file can then be copied to *spiral.dat* to rerun the program. If spiral.dat.sav already exists, ".sav" is appended to the name resulting in the save file *spiral.dat.sav.sav*.

25.5 FEHM Files (*.zone and *.stor)

The FEHM output include both zone and stor files.

25.6 Log File (*.log*)

This file contains program output. (not yet implemented)

26.0 "H3" SAMPLE PROBLEM for WELLMESH

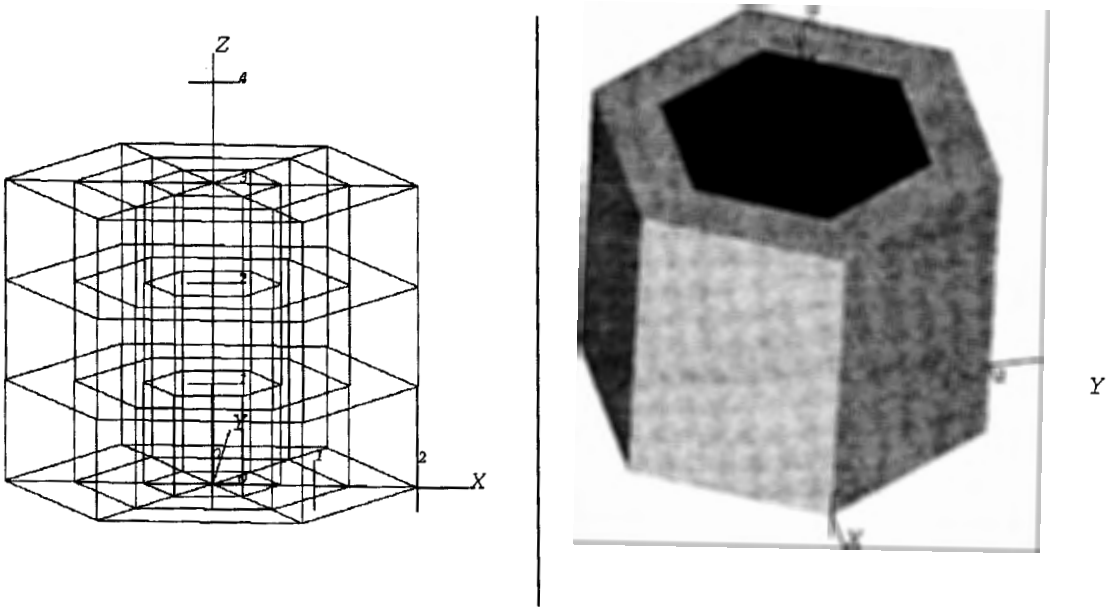


Figure 1. These images show the resulting mesh created from the "h3" data set. They show a radial refinement of 3 and a radius length of 2.0. Three well segments were created from the input of four center coordinates. The wire frame image on the left shows the radial and segment divisions of the well mesh. The image on the right shows the 3 material layers that were created for this sample run.

These sample runs are from the wellmesh test directory using *h3.well* input coordinates. The well is a simple vertical type with 4 coordinates and 3 segments. The file *h3.well* has the following contents:

```
12345678901234567
0 0 3
0 0 2
0 0 1
0 0 0
```

The top center coordinate is located at (0,0,3) and the bottom is at (0,0,0). The radius length and refinements are defined by the wellmesh macros. In the following examples we will create 3 well layers, label the layers with integer values and define the radius length. Figure 1 shows how this mesh will look with AVS.

26.1 WELLMESH Sample Run (interactive)

This example uses coordinates in *h3.well* and does not have a macro file for the macro definitions. The user will be prompted for values. Below is a sample session which also describes each of the macros as they are prompted.

```
example% wellmesh

[[ GEO>> WELLMESH>> 95/21/06_14:47:10 ]]

GET MEM WELL: Memory allocation (WORDS)
  50    well_layer
 12000    imt_nwell
 24000    iclr_ewell
 192000    icon_ewell
 12000    xwell
 12000    ywell
 12000    zwell
-----------------------------------------
Total memory allocated =  264050 words

GEO>> wellmesh names initialized:
      .dat     wellmesh.dat
      .sav     wellmesh.dat.sav
      .log     wellmesh.log
GEO>> WARN>> file:"wellmesh.dat                " does not exist.
GEO>> WARN>> wellmesh will prompt for macro values.

GEO>> WELLMESH INTERACTIVE MODE.
GEO>> ALL MACROS.
GEO>> Type Changes or "Enter" to continue.
```

Since a macro file was not found, wellmesh will cycle through the macros. For each macro the current values will be displayed, along with the syntax usage. If no changes need to be made to the macro values, press the enter key and continue.

project macro allows a rootname to be defined for wellmesh. It can also be defined on the command line. Wellmesh will try to form file names from this rootname, so it should not contain any spaces. File names are created for the coordinate.well file, macro saved *.dat.sav* file and the log file. In this example, the rootname is changed from "wellmesh" to "h3". The new names will include *h3.well*, *h3.dat.sav*, and *h3.log*.

```
|=================================================
| MACRO: (pro)ject
|_________________________________________________
| VALUES:
| wellmesh
|_________________________________________________
| SYNTAX: < "Project_name (i.e. rootname)">
|_________________________________________________
| CHANGES: h3
|_________________________________________________
```

coordinate macro sets the parameters necessary to define the center coordinates of the well. In this example, the coordinates are found in *h3.well* so no change is made to this macro.

```
|=====================================================
| MACRO: (coo)rdinate
|_____________________________________________________
| VALUES:
| Well Type:  coord
| Coordinates read from h3.well
|_____________________________________________________
| SYNTAX: <"filename.well"> containing coordinates
| or      real top(<x>, <y>, <z>) <depth>
|_____________________________________________________
| CHANGES:
|_____________________________________________________
```

radius is the distance from the inside center of the well to the furthest outside point. The length can be set as well as the refinement (number of divisions made along the radius length). In this example, the radius length is set to 2.0 and it will be divided 3 times, to create 3 layers in the well.

```
|=====================================================
| MACRO: (rad)ius
|_____________________________________________________
| VALUES:
| Length:    1.0000000000000
| No refinement (One layer).
|_____________________________________________________
| SYNTAX: <real radius_length> <int radial_refinement>
|_____________________________________________________
| CHANGES: 2.0 3
|_____________________________________________________
```

segment refers to the portion of the well between two center coordinates. By default, the length of each well segment will be the distance between each of two center coordinates. This length can be rounded off, it can also be refined. By default, the segments of the well are not connected except at the center. This is useful for determining accuracy of the coordinates. If the user chooses to connect the segments, a midpoint node location will be generated between the end segments, and the segments will then be connected at these points. In this example the segments are connected.

```
|=====================================================
| MACRO: (seg)ment
|_____________________________________________________
| VALUES:
| Lengths set by coordinates.
| No refinement.
| Not connected.
|_____________________________________________________
| SYNTAX: <real rounded_length> <[no]"connect">
|_____________________________________________________
| CHANGES: connect
|_____________________________________________________
```

element macro allows the hex well to be converted into a tetrahedral mesh with a conversion ratio of either 5:1 or 24:1. In this example, there is no change made and wellmesh will create only a hexahedral mesh.

```
|=====================================================
| MACRO: (ele)ment
|_____________________________________________________
| VALUES:
| Hex elements.
|_____________________________________________________
| SYNTAX: <"hex"> <"tet"> int<5 or 24>
|_____________________________________________________
| CHANGES:
|_____________________________________________________
```

material macro allows each layer in the well to be assigned an integer value. In the list provided by the user, the inside layer will be assigned the first value, the next layer out will be assigned the second value, and so on until the last integer value. If the number of values are less than the total number of layers, then the outer layers will be assigned the last id in the list. In this example, the inner layer of the well is assigned a material id of 1, the second layer is assigned id 2, and the outside layer is assigned the id 5.

```
|=====================================================
| MACRO: (mat)erial
|_____________________________________________________
| VALUES:
| Layer  1:  0
| Layer  2:  0
| Layer  3:  0
| WARNING: 1 Material: no AVS element color.
| WARNING: Number of materials =  1
|        Number of layers   =  3
|_____________________________________________________
| SYNTAX: int<material_id> <material_id> ...
|_____________________________________________________
| CHANGES: 1 3 5
|_____________________________________________________
```

output macro controls toggles which control wellmesh output. Mesh files can be written in formats compatible with AVS, X3D, and GMV. The macro values can be saved to file, which can then be used as a macro *.dat* file. The debug toggle results in verbose debugging messages to the screen. These toggles can be turned on or off. In this example, the format types for AVS and GMV are turned on, as well as the toggle to save the macro values to file.

```
|=====================================================
| MACRO: (out)put
|_____________________________________________________
| VALUES:
| NO DEBUG
| WARNING: No mesh files will be written.
```

```
|_____________________________________________________
| SYNTAX: <"avs"><"x3d"><"gmv"><"sav"><"log"><"debug">
|_____________________________________________________
| CHANGES: avs gmv sav
|_____________________________________________________
```

At this point, the user can choose to further check or change macros. If the "enter" key is pressed, wellmesh will run to completion. If the user needs to be reminded what macros are available, enter **help** for a display.

```
GEO>> WELLMESH INTERACTIVE MODE.

GEO>> Type "help", <macro>, or "Enter" to continue.
GEO PROMPT> help
|=====================================================
| MACRO: (hel)p
|_____________________________________________________
|
| Available Macros:
|  (all)          cycle through all macros
|  (coo)rdinate   filename or top coordinates
|  (rad)ius       length and refinement
|  (seg)ment      length and refinement
|  (ele)ment      type
|  (mat)erial     id numbers for layers
|  (out)put       type of files, screen output
|  (pro)ject      name
|  (sto)p         discontinue
|_____________________________________________________
```

By typing at least the first 3 characters of the macro ("out"), the user can get a display of the current values for the macro **output**. In the following example, the user decides to not have GMV files written. This is done by typing the toggle along with the word **no** or **not** to turn it off.

```
GEO>> Type "help", <macro>, or "Enter" to continue.
GEO PROMPT> out

|=====================================================
| MACRO: (out)put
|_____________________________________________________
| VALUES:
| avs
| gmv
| sav
| NO DEBUG
|_____________________________________________________
| SYNTAX: <"avs"><"x3d"><"gmv"><"sav"><"log"><"debug">
|_____________________________________________________
| CHANGES: no gmv
|_____________________________________________________
```

At this point, the user types "enter" and wellmesh runs to completion. The final values of all macros are also displayed.

```
GEO>> Type "help", <macro>, or "Enter" to continue.
GEO PROMPT>
GEO>> WELLMESH>> Final Setup>>
_____________________________________________________
       <<< GEO WELLMESH >>>

      SETUP FOR "h3"
       95/21/06_14:47:10
_____________________________________________________
|===================================================
| MACRO: (fil)es
|___________________________________________________
| VALUES:
| wellmesh.dat                21
| h3.well  26
| h3.dat.sav                  23
| hex_well.inp
|___________________________________________________
|===================================================
| MACRO: (out)put
|___________________________________________________
| VALUES:
| avs
| sav
| NO DEBUG
|___________________________________________________
|===================================================
| MACRO: (pro)ject
|___________________________________________________
| VALUES:
| h3
|___________________________________________________
|===================================================
| MACRO: (coo)rdinate
|___________________________________________________
| VALUES:
| Well Type:  coord
| Coordinates read from h3.well
|___________________________________________________
|===================================================
| MACRO: (mat)erial
|___________________________________________________
| VALUES:
| Layer  1:  1
| Layer  2:  3
| Layer  3:  5
|___________________________________________________
|===================================================
| MACRO: (rad)ius
|___________________________________________________
| VALUES:
| Length:    2.0000000000000
| Refinement (Number of layers):    3
|___________________________________________________

|===================================================
| MACRO: (seg)ment
```

```
|_____________________________________________________
| VALUES:
| Connected.
| Lengths set by coordinates.
| No refinement.
|_____________________________________________________


|=====================================================
| MACRO: (ele)ment
|_____________________________________________________
| VALUES:
| Hex elements.
|_____________________________________________________

 GEO>> FILES: Opening save file:h3.dat.sav
 GEO WELLMESH>> OPEN FILE: Input file:h3.well
 MK WELLMESH: 3 segments,     96 nodes,     54 elements    2.00 radius
SET WELL CMO XYZ: hex_well
 Mins:   -2.0000000000000  -1.7320508075689  0.
 Maxs:    2.0000000000000   1.7320508075689   3.0000000000000
 Total nodes set:  96
 GEO>> WELL2CMO:     3 hex segs processed into well cmo.
GEO>> WELLMESH>> The following files were created:
   h3_hex.inp
   h3.dat.sav
 GEO>> END WELLMESH.

example%
example% cat h3.dat.sav
# 95/21/06_14:47:10
project h3
output avs sav
coordinates h3.well
materials  1  3  5
radius     2.0000000     3
segment connect      0.0000000
element hex
stop
example%
```

26.2 WELLMESH Sample Run (batch mode)

This is the same run as before, except that now we have a macro *wellmesh.dat* file. Now wellmesh will read the macro values from the macro file, and there will be no prompt for the user to respond to.

```
example% wellmesh

 [[ GEO>> WELLMESH>> 95/21/06_16:10:25 ]]

GEO>> SET NAMES>> files initialized:
      .dat     wellmesh.dat
      .sav     wellmesh.dat.sav
      .log     wellmesh.log
 GEO>> MREAD>> Opening dat input file: "wellmesh.dat                "
 GEO>> MREAD: Done Reading Input File wellmesh.dat
 GEO>> WELLMESH>> Final Setup>>
 ____________________________________________________
        <<< GEO WELLMESH >>>

      SETUP FOR "h3"
       95/21/06_16:10:25
 ____________________________________________________
 |================================================
 | MACRO: (fil)es
 |________________________________________________
 | VALUES:
 | wellmesh.dat                21
 | h3.well  26
 | h3.dat.sav                23
 |________________________________________________
 |================================================
 | MACRO: (out)put
 |________________________________________________
 | VALUES:
 | avs
 | sav
 | NO DEBUG
 |________________________________________________
```

The program will continue to run to completion until the requested files have been written.

```
 GEO>> WELL2CMO:     3 hex segs processed into well cmo.
 1  dumpavs h3_hex.inp  hex_well
dumpavs h3_hex.inp  hex_well
Command: dumpavs Number of parameters:        3
 GEO>> WELLMESH>> The following files were created:
   h3_hex.inp
   h3.dat.sav
 GEO>> END WELLMESH.

 example%
```

26.3 WELLMESH Sample Run (command line arguments)

WELLMESH can be run with 1 command argument. The argument is a character string of up to 32 letters. This string will be read in as the rootname and will be use to form the files names.

```
example%
example% wellmesh h3

 [[ GEO>> WELLMESH>> 95/22/06_13:12:48 ]]

GEO>> MREAD>> Reading command line.
 GEO>>  wellmesh names initialized:
       .dat     h3.dat
       .sav     h3.dat.sav
       .log     h3.log
 GEO>> MREAD>> Opening dat input file: "h3.dat                    "
 GEO>> MREAD: Done Reading Input File h3.dat
 GEO>> WELLMESH>> Final Setup>>
_____________________________________________________
        <<< GEO WELLMESH >>>

       SETUP FOR "h3"
        95/22/06_13:12:48
 ____________________________________________________
|====================================================
| MACRO: (fil)es
|____________________________________________________
| VALUES:
| h3.dat                    21
| h3.well  26
| h3.dat.sav                  23
|____________________________________________________

GEO>> WELL2CMO:      3 hex segs processed into well cmo.

    1  dumpavs h3_hex.inp  hex_well
dumpavs h3_hex.inp  hex_well
Command: dumpavs Number of parameters:         3
 GEO>> WELLMESH>> The following files were created:
   h3_hex.inp
   h3.dat.sav
 GEO>> END WELLMESH.

example%
```

27.0 WELLMESH ERROR PROCESSING

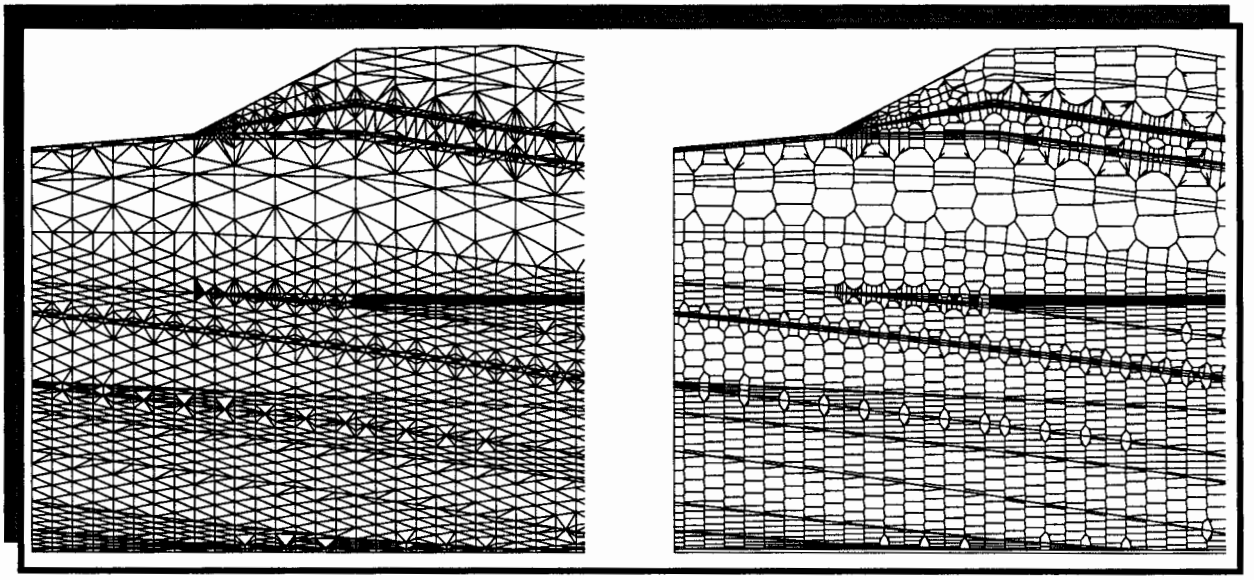
Two types of errors can occur. Those that are non-fatal will not terminate the program. Warnings will be printed to screen. The following are errors which can cause wellmesh to abort.

Table VIII. Error Conditions Resulting in Program Termination

GEO>> MREAD>> ERROR: Problem Reading Input File	Input file containing macro definitions has been opened but could not be read.
GEO WELLMESH>> FILES: "file" does not exist.	This error occurs if the user indicates that either a macro file or coordinate file is to be used, but the file is not found.
GEO>> ERROR: Exiting "routine()", program stopped.	If a fatal error occurs during a "routine()", the program will stop. Error messages will have been printed by the "routine()" that failed.
ERROR >> WELL2CMO: 'number' > 'constant' max	The number of nodes or coordinates or segments is larger than the number allocated by wellmesh.

LA-UR-95-4143

GEOMESH GRID GENERATION



GEOMESH PROJECT WORKLISTS

Los Alamos National Laboratory
Earth and Environmental Science Division
Geoanalysis Group, EES-5
Carl W. Gable
Terry Cherry
Harold Trease
George A. Zyvoloski

TABLE of CONTENTS

1.0 DOCUMENT DESCRIPTION

This Document has been written to aid the GEOMESH project team in setting priorities and goals. It provides the current project status by listing work items both done and to be done. Other documents available for this project include:

- *GEOMESH Project Description* - overview of capabilities, plans, and grid applications.
- *User's Manual for GEOMESH* - how to create computational grids.
- *GEOMESH Worklist* - running record of work done for the GEOMESH project.
- *X3D User's Manual* - how to run the command language for mesh object commands.
- *GEOMESH Handout* - a short, summarized version of the Project Description.
- *GEOMESH Programmer's Handbook* - code development details for GEOMESH.

2.0 DOCUMENT OVERVIEW

This document contains worklists for the following GEOMESH projects:

- GEOMESH - Major Tasks of Project
- COMMANDS for Geo Mesh Objects
- STRAT2MESH Geo Mesh Translator
- WELLMESH Geo Mesh Translator
- GRIDDER Geo Mesh Translator
- Grid Catalog for GEOMESH Applications
- Documentation and Administration
- Code Work for FEHM

3.0 DOCUMENT CONVENTIONS

These are on-going lists which began September of 1994. The list includes the current status of each work item, a priority rating, who is the primary worker for each item, and other work the item might depend on.

Completed work appears in the grey rows.

Priorities are indicated in the DO column. Priorities range from 10 being highest, to 1 being lowest (on hold).

The primary participants whose initials appear are:

C	Carl Gable	T	Terry Cherry	G	George Zyvoloski
H	Harold Trease	L	Lynn Trease	LL	Loraine Lundquist
		GT	Grid Team Members		

4.0 LANL EES-5 Code Development Deadlines

This list was constructed from documentation compiled by Bruce Robinson for the Yucca Mountain Project in May of 1995. The documentation listed deliverables and tasks for the EES-5 Code Development Team. This list includes details tasks relevant to the Grid Generation portion of the project.

YMP Deliverables FY 95

APR 3	I.	3471 Publish Final FEHM Theory and UM (PA)
MAY 31	II.	3470 Publish FEHM Verif. Problems (PA)
	III.	3468 Summary Rpt: Site-Scale Integrated Transport (RSA)
		A. GEOMESH Interface (STRAT, LYNX, ARCH/INFO)
		B. USGS and LBL Data Set - Stratamodel 3D of YM
		C. FEHM Grids - 3D and 2D subgrid extractions
SEP 29	IV.	4073 Ltr Report: Tech. Support to PA (PA)
	V.	4074 Ltr Rpt: Grid Generation Extension for FEHM (PA)
		A. FRACMAN/FEHM Linkage
		B. GEOMESH Interface (STRAT, LYNX, ARCH/INFO)
		C. USGS and LBL Data Set - Stratamodel 3D of YM
		D. FEHM Grids - 3D and 2D subgrid extractions of above
		E. GEOMESH Documentation - User's Manual (include FRACMAN)
	VI.	4075 Ltr Rpt: Model Implementation (PA)
SEP 30	VII.	3469 Summary of Near Rep. flow and Geochem (RSA)
		A. USGS and LBL Data Set - Stratamodel 3D of YM
		B. FEHM Grids - 3D and 2D subgrid extractions of above
	VIII.	3467 Report on Code Development (RSA)
FY 96	IX.	3051 Fracture-Matrix Coupling and P-Tunnel (RSA)
	X.	CST? Modeling Np Lab Data (RSA)

5.0 GEOMESH WORKLISTS

GEOMESH PROJECT - Major Tasks

Status	DO	Who	Tasks	Dependencies
Done 2/95	*	C, H	1) Design geomesh data objects and data structures - cell based?	harold's objects -> carl's structures
In progress	8	C	2) Design geomesh program flow - Diagram	conceptual diagram(done)
			3) Consider Geologic wrappers for some of the CMO commands	
5/95	10	T	4) Develop scripts for mergemesh and wellmesh and strat2mesh	
5/95	9	C, T	5) Decide file name conventions for I/O files (namelist, ctl_file ...)	
5/95	9	T	6) Develop simple interfaces for MERGEMESH, REFINE, WELLMESH ...	
			7) Design and implement user-friendly interface for GEOMESH	
	6		8) Generalize all I/O and error reporting levels. Implement use of x3d "lu", writlog for GEOMESH programs	
			9) Implement macro control of output level (0,1,2,3...)	
	5		10) Develop validation tests for different mesh scenarios	GEOMESH interface needed
6/95	10	T	11) Write NET2AVS, convert Earthvision ".net" files to AVS files. These are tiangle elements with zmax to zmin node ordering. So occasionaly the node odering will be counter-clockwise instead of clockwise.	
	4		12) Develop FAULTMESH strategy to include fault off-set.	
			13) Document Platform issues (Strat is SGI only)	
6/95 hold		H	14) multi-defined nodes not yet implemented by FEHM, turned off	
			15) FEHM info for MDN	
			16) FEHM info for neighbor connectivity	
			17) FEHM info for Aij	
2/95		C	18) material filter, in meshgen	
			19) Grid Coarsening	

CMO COMMANDS

Status	DO	Who	Tasks	Dependencies
9/95	1	C	20) Develop REFINE based on; location, interface, material, solution values, ...	
9/95		C	21) Debug north -south section extraction	
9/95		C	22) Given point distribution xyz, create tet grid to compare LBL and FEHM models.	
			23) Enable hextotet to allow some elements ot remain as hex.	
9/95	4	T,C	24) hex2cmo, init_geocmo, set_geocmo, geo_fdump	Need to generalize
7/95		GT	25) readavs, dumpavs	
8/95	1		26) readfehm .fin file, dumpfehm .ini resart file	
started	1	T	27) readtins - Earthvision format for usgs model of Yucca Mtn.	
9/95	2	C	28) output .area files, nodes outside zone, surface area for nodes	
8/95	2	C	29) output .stor, area ij coefficients	
9/95		GT,C	30) Interpolate (doping) one grid onto another, using fields	
			31) read Dynamic Graphics	
			32) read Lynx	
			33) read CPS3	
			34) read Intergraph	
			35) dumptough output grids compatiable with TOUGH2 code	

ADDMESH CMO COMMAND

Status	DO	Who	Tasks	Dependencies
Done 1/95	*	H	36) ADDMESH: Add a mesh to another	module from harold
5/95	10	T	37) Write script for interface, Integrate into GEOMESH as aux program, determine input and output options.	
5/95		T	38) Write macros for action, background, foreground, output.	
6/95		T	39) Implement action "add" and "meld".	
hold	9	T, H	40) addmeshAllow user to reset properties	need new "add" task
		T	41) Replace read avs with faster read.	new x3d libraries
In Progress		T, H	42) Test with well as foreground.	

ADDMESH CMO COMMAND

Status	DO	Who	Tasks	Dependencies
5/95		T	43) Macro "meld" works, fix "add" for well properties	new x3d libraries

MESHGEN - Geo Mesh Translator

Status	DO	Who	Tasks	Dependencies
Done 10/93		T	44) Add general conversion routine to include conversion between feet and meters	
Done 12/94		C	45) Entire code converted to double precision	
Done 12/94		C	46) Add capability for Repository Horizons for B. Robinson studies of transport of tracers released from the repository.	
Debug		H	47) Add option to divide hexahedra into 5 tetrahedra without adding any nodes. This keeps mesh size smaller then with the hex elements divided into 24 tetrahedra.	
1/95		C	48) OUTPUT: screen output level = 0,1,2,3,...3 and don't output intermediate mesh	
7/95	8	GT	49) Read & Output AVS UCD info using CMO	
			50) Implement batch/ interactive control of GEOMESH. User Control of program I/O - part of interface design	
8/95	8	T	51) Convert to CMO data structure before hex_to_tet	
9/95	7	T	52) Write Routine to check layer ordering and allow corrections. Give choice of file ordered or sort routine.	
	4	T, C, GT	53) Control brick resolution, Add user control of nx, ny, nz of brick mesh. Control layer refinement for individual layers. Present version only applies to all layers. Use RESAMPLE CMO commands.	allocate, write, deallocate
8/95	3	T	54) Deallocate brick grid arrays earlier - don't allocate if not used. Deallocate unneeded memory after hexcmo created.	
8/95		T	55) Use cmo command dumpfehm for zones lists and .stor files	

STRAT2MESH Geo Mesh Translator

Status	DO	Who	Tasks	Dependencies
Done 2/95	*	C	56) Write Stratamodel to CMO grid translator using OpenSGM library (opensgm is a library that can only be linked if you have a license. I think we cannot even export the binary since it checks for a license at run time).	
Done 3/95	9	C	57) Correct ZONE list output when mesh object is 2D & 3D	
Done 3/95		C	58) correct the way pinch out are handled, right now there are some cases that are not correct (i.e. C. Faunt model)	
6/95		C	59) Implement macro control of program for bactch mode.	
		C, T	60) Implement interactive mode of program.	user interface designed
6/95	10	C	61) Correct definition of itp1 and icr1 in hexmesh so that recon3d works.	
6/95	9	C	62) Include top/bottom/front/back/left/right as part of ZONE list output. - present version only outputs ZONE list of different materials - 2D and 3D	
5/95	8	C	63) Assign Properties so not hardwired. Implement user control of layer/sequence material property definition.	
6/95		C	64) Include char string variable for naming material layers.	
6/95		C	65) include char sting name of material as part of zone list output.	
5/95		C	66) Output Aij (Area I J) integration coefficients as part of mesh generation.	
5/95		C	67) Fix error in format of Zone list output files.	
5/95		C, H	68) Fix 'nibble' of bottom/side corners.	
	7	C	69) release memory of hexmesh object after it is copied into tet-mesh object in hex_to_tet	
	7	C	70) Check and repair choosing SFM based on coords., right now one mush specify row and column numbers to extract a subset of the SFM	
	6	C	71) Write script and Use CMO EXTRACT command rather than DECIMATE to pull out 2d sections from a 3D mesh object.	
works using X3D, turned off in meshgen	6	C	72) Write interface script to be able to turn multi-defined nodes at material interfaces on and off with a switch, right now it is hard wired	
	5	C, H	73) Define 'height in section' as a node attribute	
	5	C	74) Write Stratamodel cell attributes to AVS UCD file.	
			75) Plan to deal with new Stratamodel release. Data structure and framework model will change, as well as how pinched out layers are terminated.	

WELLMESH Geo Mesh Translator

Status	DO	Who	Tasks	Dependencies
Done 10/94		T	76) Create module to write wells in AVS format	
Done 12/94		T	77) Change tube interfaces so nodes are shared, not redundant, send tunnels to Harold for testing of his modules.	
1/95		T	78) Convert WELLMESH to Double precision	
5/95	10	T	79) Test Double precision, change auto-testing files	Conversion to dbl
5/95		T	80) Change wellmesh to use cmo and x3d libraries.	
5/95	6	T, C	81) Remove unneeded parameters, add more useful ones.	Finish script and interface
6/95		T	82) New macro interface for batch mode.	
6/95	10	T, C	83) New macro interface for interactive modes.	Design of GEOMESH flow and aux program flow
6/95	10	T	84) assign properties to well vs. buffer, enable well layers to be assigned integer values - macro (mat)erial	
Need header template		T	85) Add headers and put code into PVCS.	
		T	86) Change print statements to writlu	
		T	87) Add more user control to radius refinement, use radial lengths instead of level of refinement.	
Get formula from YUCCA?		T	88) Add radial corner (YUCCA) well description calculation to wellmesh.	
Ahmed may develop in Sept.	7	T, GT - Ahmed	89) Add spline or NURB interpretation of the well. Use cmo_well and spline routine.	
	7	T	90) Add output files with node information including center, exterior, all, and zone lists. Use cmo_zone_list.	
Need accurate data.		T,H	91) Test wellmesh with addmesh.	
this may be done in addmesh	6	T,H	92) PLAN: assignment of node properties after well made, user assignment i.e. define buffer on outside of well to refine. Resample or user assign radial characteristics. Use cmo_well to assign background vs. static well node properties.	
	3	T, C	93) PLAN: create 2-D well, possibly with EXTRACT command.	

WELLMESH Geo Mesh Translator

Status	DO	Who	Tasks	Dependencies
	1	T	94) Two possible solutions for center rotation in wellmesh 1) learn how to predict rotation around Z axis, then undo. 2) 2) make sure rotations around 3 axis done correctly (y,z ok). After rotation fixed, send knot well to harold	

GRIDDER regular grid generator

Status	DO	Who	Tasks	Dependencies
Done 9/94		LL	95) Create gridder - a regular grid generator, aux program.	
Done 2/95		T	96) Corrected segmentation fault, changed output for floats from %f to %14.8	
7/95	1	LL	97) DEBUG: problem with the way the materials are defined, There should only be 4 materials however we end up with something like 21	
7/95	1	LL	98) DEBUG: output file input.tmp s not the same as the input 4mat.inp The roundoff shows up. It looks like what goes to the input.tmpfile is x0 and x0 + nx*dx rather than x0 and x1.	

GEOMESH Grid Catalog (Applications)

Status	DO	Who	Tasks	Dependencies
Done 11/94		C	99) Create sample problems for GEOMESH testing & tutorial.	
Done 11/94		T	100) Create sample problems for WELLMESH testing and for Harold's routines, include spiral well.	
2/95	*	T	101) Create main and focus tunnels for Yucca Mtn., Data from J Wang's memo and Tom Dey.	
12/94		C	102) Import LBL Yucca Mtn. Data set to Stratamodel.	
6/95	9	C	103) USGS Data Set (Regional Saturated Zone Model) with Strata-model - Claudia	
7/95		T	104) Grids for UofTexas, Erik	
8/95	9	C	105) Site Flow with FEHM - George	
	5	T,C	106) C wells - Tracer test with Paul Reimus	Test wells/addmesh.
9/95			107) P-Tunnel - Experiment for YMP	
			108) Semi Conductor with Grid Team	
9/95		C	109) Calico Hills Test - Gilles Bussod	

GEOMESH Grid Catalog (Applications)

Status	DO	Who	Tasks	Dependencies
			110) Fracman Grids - fracture distribution for computational grids	
			111) PA - Performance Assesment for YMP	
			112) RSA - Retardation Sensitivity Analysis	
			113) Volcanic Subsurface Effects - Nina and Greg	
			114) Area G - Los Alamos Env. Restoration	
			115) Sandia Trans in heterogenous media - YMP	

Documentation & Code Administration

Status	DO	Who	Tasks	Dependencies
Done 7/94		C	116) WRITE FIRST DRAFT tutorial document	
Done 9/94		T,C	117) WRITE SECOND DRAFT: Add figures, create second draft of tutorial to put on ftp anonymous	terry & carl into frame4
Done 1/95	*	C,H,L	118) ADM: Put entire code under source control (PVCS) including Harold's librefine and memory management.	
Done 3/95		T	119) ADM: Reorganize docs into sections easily used separately, put into common work directory on vega.	
Done 1/95		T	120) ADM: Create project Makefile and directory structure	
Done 1/95		T	121) ADM: Write C code so portable for both cc and acc	Research methods
Done 12/94		T	122) WRITE WELLMESH documentation, module description	
Done 2/95		T, C	123) WRITE WHITE PAPER of mesh generation techniques and GEOMESH for proposals and documentation	
Design & add headers	8	T	124) ADM: Put wellmesh module in PVCS (GEOMESH and X3D)	test double precision
9/95	7	T	125) ADM: Create template for user's manual, programmers manual, overview, applications, worklists and reports	
9/95	6	T,C	126) WRITE Application examples (USGS, LNL, wells...)	design first
grid effects done	6		127) SUMMARIZE VALIDATIONS: mesh designs and use, document problems, grid effects	Validation tests done
In progress	5	T	128) ADM: Write documentation on how to use Makefile and setup GEOMESH directories for work.	Create Makefile and test
Draft done 9/95	5	T	129) WRITE SEPT. User's Manual. Make it useful for online help for Lynn	
9/95		T, G	130) WRITE SEPT. White Paper (geomesh.WP)	Reviews

Documentation & Code Administration

Status	DO	Who	Tasks	Dependencies
Draft 1 done		T	131) WRITE SEPT. Programmer's Manual (geomesh.PM)	
Sept Done		T	132) WRITE SEPT. Project Status (geomesh.STAT)	
9/95		G	133) WRITE SEPT. Example Problems (grid catalog)	
9/95	4	T	134) WRITE Overview part of User's Manual - use white paper	design first
In Progress			135) UPDATE and Keep Documents current	carl, harold, terry, george input needed

Code Work for FEHM

Status	DO	Who	Tasks	Dependencies
Done May, 94		T	136) AVS FIX liquid vs vapor output, add unit labels to avs output, document	
Done 1/95		T	137) ADM: Write C code so portable for both cc and acc - give lynn info on portable Makefiles and auto generating dependencies.	Research methods for compilers and Makefiles
4/95			138) AVS corrected to write 0.0 in Z coord for 2D problems	
2/95		G	139) CONNECTIVITY, how stored in FEHM, how output. Describe correctly in documentation	
4/95			140) DOUBLY DEFINED NODES: CHANGE FEHM DATA STRUCTURE to include material, cell based properties	consider harold's mesh structure
Done 12/94		L, T	141) PORTS to other platforms, create C standards	Lynn working on ports Terry start docs
Done 2/95		G	142) FEHM OUTPUTS: work on velocity outputs	George doing
Hold		T, kmb	143) AVS routine to write displacements for stress	
		G, T	144) AVS OUTPUT PERMS for unsaturated flow, need relative perm/saturation	how info stored in FEHM
			145) ELEMENT ID added to data structure	
		C	146) AVS2FEHM, FEHM2AVS	FREEFORM or script files
6/95			147) FEHM IN 2-D goes to 45 degrees	need initial output of values
Fill in heads		T,L	148) CHANGE FILENAMES, add headers to AVSIO source code	follow YMP standards
		G, T	149) 2ND VERSION AVSIO DOCS: update first version of AVS output for FEHM	create demo examples George input on velocities
		T	150) AVSIO:ADD TIME STEP to AVS output files	

Code Work for FEHM

Status	DO	Who	Tasks	Dependencies
Hold	*	T	151) ADD ABILITY TO READ AVS UCD FILES for FEHM input	
			152) CREATE BETTER FEHM OUTPUT for initial values, add better error checking	
6/95		L	153) UPDATE AND DOCUMENT HISPLT for FEHM post-processing (Lynn added to GUI browser)	
		G	154) Include null points for DMR "dudded points"	

Priority II Projects

Status	DO	Who	Tasks	Dependencies
	8	H,.C, GT	155) DYNAMIC ATTRIBUTES added to CMO for AVS I/O	design data objects & structures
	6	C & LL, G & T	156) TESTING & VALIDATION of wells in a mesh. Use runs with and without wells added, and with different refinements and materials.	seamless geomesh + well + add need problem definition (Zora?)need a student to do it?
9/95	3	C, H	157) AMR: Add Adaptive Mesh Refinement to GEOMESH to be used by FEHM or other transport programs. Read FE grids for refinements, flag chosen grids: layer, value, gradient, interface or properties	design data objects design program flow need ability to read FE grids need AMR module from H
	3	C, H	158) FAULTMESH: Add faults to geometry mesh	problem defined - C module from harold
	3		159) NURBS: Add capabilities to handle surfaces expressed as Non Uniform Rational B-Splines. i) convert grid to NURBS ii) NURBS to grid iii) wells as NURBS iv) faults as NURBS	Design NURBS into geomesh Research current methods of using NURBS
	1	H, T, L	160) GEOMESH: Write seamless version of wellmesh, addmesh, and geomesh as program or script. Define behavior of materials. Add all to PVCS.	
	1		161) FREEFORM read & write data added to Geomesh	Talk to NOAA/NGDC Need someone to program with the FREEFORM system

X3D Users Manual

by

Denise George

Harold Trease

X3D Users Manual

Table of Contents

I. Introduction

a. X3D

X3D is a library of user callable tools that provide mesh generation, mesh optimization and dynamic mesh maintenance in three dimensions for a variety of applications. Geometric regions within arbitrarily complicated geometries are defined as combinations of bounding surfaces, where the surfaces are described analytically or as collections of points in space. A variety of techniques for distributing points within these geometric regions are provided. Mesh generation uses a Delaunay tetrahedralization algorithm that respects material interfaces and assures that there are no negative coupling coefficients. The data structures created to implement this algorithm are compact and powerful and expandable to include hybrid meshes as well as tetrahedral meshes.

Mesh refinement and smoothing are available to modify the mesh to provide more resolution in areas of interest. Mesh refinement adds nodes to the mesh based on geometric criteria such as edge length or based on field variable criteria such change in field. Mesh smoothing moves nodes to adapt the mesh to field variable measures, and, at the same time, maintains quality element shape. Mesh elements may become distorted as mesh nodes move during a time dependent simulation or are added as a result of refinement operations. Mesh reconnection via a series of edge flips will maintain the non-negative coupling coefficient criterion of the mesh while eliminating highly distorted elements.

An additional requirement of time dependent simulations is that as interface surfaces move, the corresponding region definitions must respond dynamically accordingly. As surfaces collide, the mesh must respond by merging points and effectively squeezing out the material between the colliding surfaces. X3D provides the necessary tools for time dependent simulations.

b. Tutorial -- Generating Initial Grids Using the X3D Command Language:

The steps involved in generating three dimensional grids in the X3D command language are:

1. Define mesh objects.
2. Define an enclosing volume.
3. Define interior interfaces.
4. Divide the enclosing volume into regions.
5. Assign material types to the regions.
6. Distribute points within the volume.

7. Connect the points into tetrahedra

Detailed descriptions of the X3D commands are given in Section III. This tutorial covers just the commands needed to generate a simple grid. The tutorial will explain how to generate a grid in a unit cube containing two materials separated by a plane.

1. Define mesh objects

Define all Mesh Objects to be used in this problem using the **cmo/create** command. The **cmo/create** command establishes an empty Mesh Object data structure (see Section II.a for a description). For this example we will need only a single 3D Mesh Object:

* create a 3D tetrahedral mesh object and name it *3dmesh*

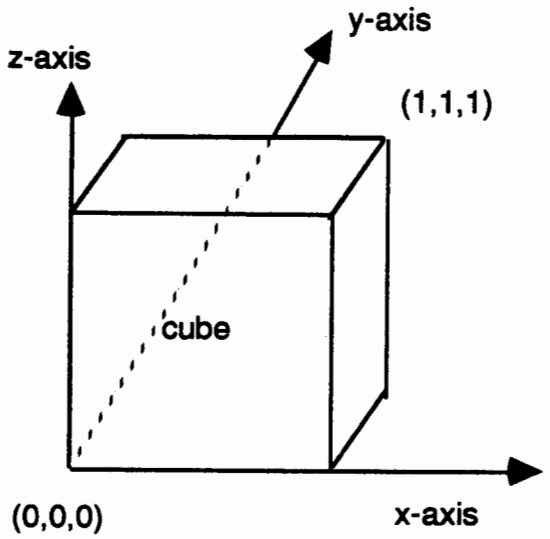
cmo/create/3dmesh/**tet/**

2. Define an enclosing volume

Define an enclosing volume using the **surface** command. Since we are defining an exterior boundary, the boundary type is **reflect**. The next item of information needed is the geometry of the volume; some common geometry types are **box**, **cylinder**, **sphere.** Geometry types, **box** and **sphere,** define closed volumes; whereas a **cylinder** is open on both ends and must be capped by planes. Along with the geometry type, the extent of the volume is defined by specifying for the box its corners, or for the cylinder its radius and end point of its axis of rotation. The enclosing volume must be convex. Complicated enclosing volumes can be described by their bounding surfaces including planes and sheets . Some simple examples of enclosing volumes are:

* unit cube

surface/cube/**reflect/box**/0.0,0.0,0.0/1.0,1.0,1.0/

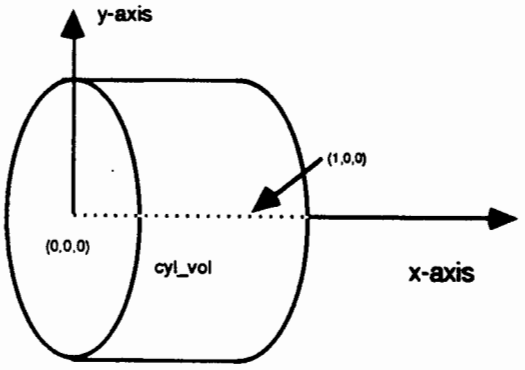


* cylinder whose axis is the x axis with radius 1 and height 1

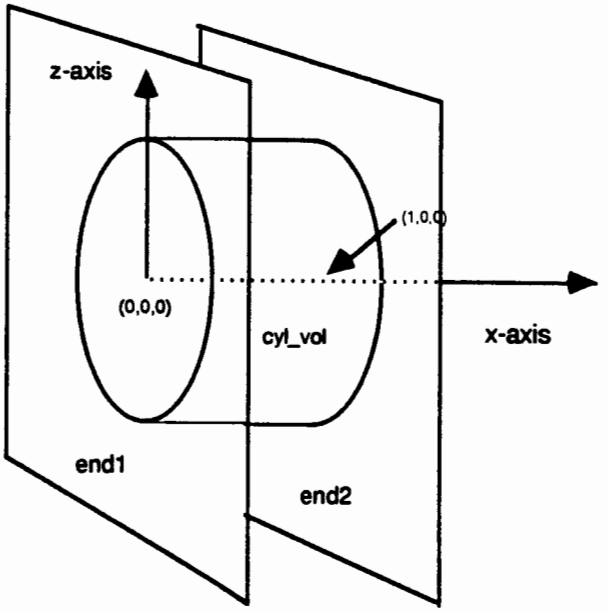
surface/cyl_vol/**reflect/cylinder**/0.,0.,0./1.,0.,0./1./

surface/end1/**reflect/plane**/0.,0.,0./0.,0.,1./0.,1.,1./

surface/end2/**reflect/plane**/1.,0.,0./1.,0.,1./1.,1.,1./



hollow cylinder



capped cylinder

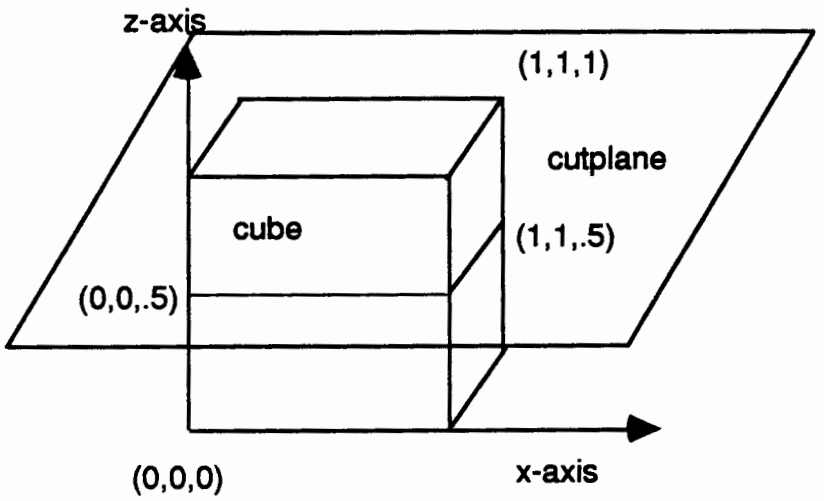
3. Define interior interfaces

Interfaces are defined with the **surface** command. In this case the boundary type is **intrface**. If the command defines a volume (e.g. box, cylinder) then the interface is the surface of the volume defined. If the command defines a plane or sheet then the

interface is the plane or sheet. It is important to remember that planes are infinite and that the order of points specifying the plane determines a normal to the plane in the usual right-hand-rule sense (see Section III.a.9). This direction is important in determining regions. In order to divide the unit cube defined above in half vertically, define a plane by:

surface/cutplane/**intrface/plane**/0.,0.,.5/1.,0.,.5/1.,1.,.5/

The normal to this plane points in the positive z direction.

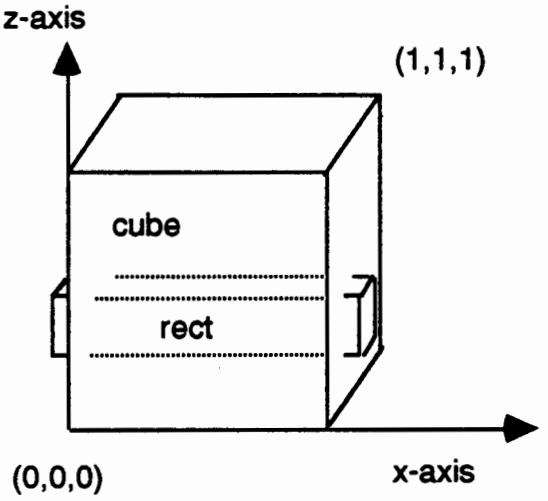


'cube' bisected by the infinite plane 'cutplane'

Interfaces may not be coincident with reflective boundaries. For example to embed a rectangle inside a cube, it is necessary to extend the ends of the rectangle beyond the cube to avoid coincident reflective and interface surfaces:

surface/cube/**reflect/box**/0.0,0.0,0.0/1.0,1.0,1.0/

surface/rect/**intrface/box**/-0.1,0.5,0.2/1.1,0.6,0.5/



'cube' with embedded 'rect', 'rect' extended beyond planar surfaces of 'cube' to avoid coincident interface and reflective surfaces

4. Divide the enclosing volumes into regions

The **region** command is used to divide the enclosing volume into regions. The operators **lt**, **le**, **gt**, and **ge** are applied to previously defined surfaces according to the following rules.

lt -- if the surface following is a volume then **lt** means inside not including the surface of the volume. If the surface is a plane or a sheet **lt** means the space on the side of the plane or sheet opposite to the normal not including the plane or sheet itself.

le -- if the surface following is a volume then **le** means inside including the surface of the volume. If the surface is a plane or a sheet **le** means the space on the side of the plane or sheet opposite to the normal including the plane or sheet itself.

gt -- if the surface following is a volume then **gt** means outside not including the surface of the volume. If the surface is a plane or a sheet **gt** means the space on the same side of the plane or sheet as the normal not including the plane or sheet itself.

ge -- if the surface following is a volume then **ge** means outside including the surface of the volume. If the surface is a plane or a sheet **ge** means the space

on the same side of the plane or sheet as the normal including the plane or sheet itself.

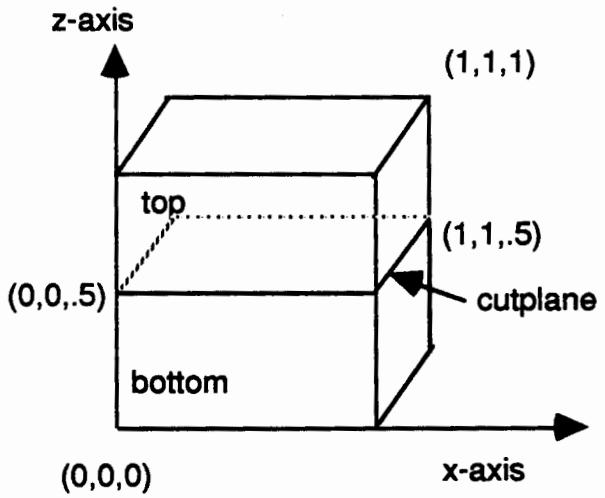
The operators **or**, **and**, and **not** applied to regions or surfaces mean union, intersection and complement respectively. The operators **or**, **and**, and **not** applied to relational operators are the normal logical operators. Parentheses are used for nesting. Spaces are required as delimiters to separate operators and operands. To define the two regions created by the plane bisecting the unit cube:

region/top/ **le** cube **and gt** cutplane /

region/bottom/ **le** cube **and le** cutplane /

The region *bottom* contains the interface *cutplane; top* contains none of the interface. Interior interfaces must be included in one and only one region.

If a region touches an external boundary, include the enclosing volume in **region** and **mregion** commands. For example, the regions *top* and *bottom* are enclosed in the volume *cube*



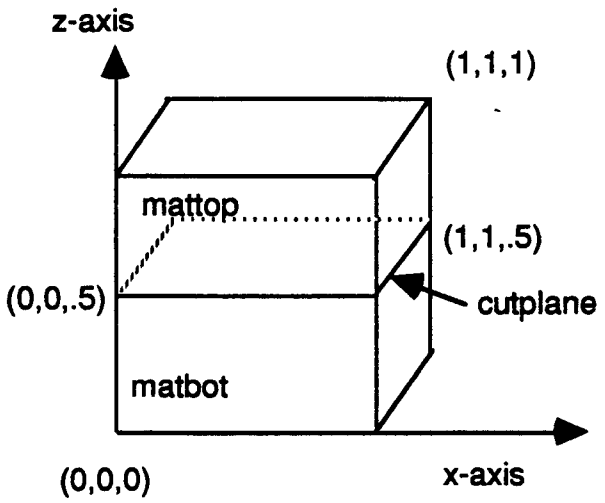
'cube' consisting of two geometric regions: 'top' and 'bottom'

5. Assign material types to the regions

Assign materials to regions using the **mregion** command. This command has similar syntax to the **region** command except that the interface should not be assigned to any material region. To assign two materials, *mattop* and *matbot*, to the regions *top* and *bottom:*

mregion/mattop/ **le** cube **and gt** cutplane /

mregion/matbot/ **le** cube **and lt** cutplane /



'cube' containing two materials: 'mattop' and 'matbot'

6. Distribute points within the volume

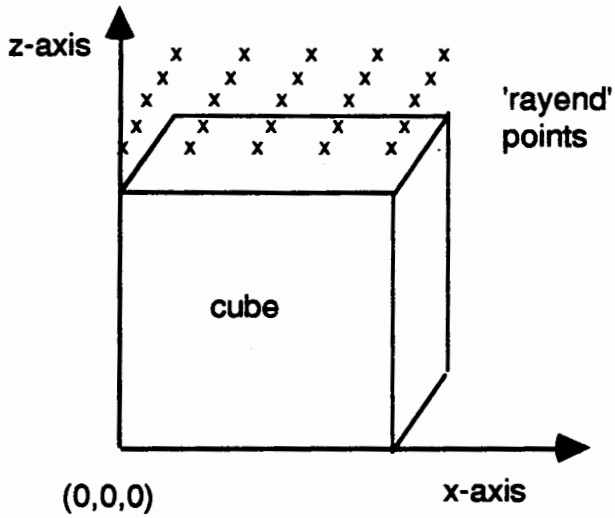
Points are distributed within regions using Cartesian, cylindrical or spherical coordinates by constructing rays that travel through regions and distributing points along these rays. For this example, points are distributed using Cartesian coordinates. The rays are specified by defining a set of points and a plane. For each point in the set, a ray is constructed normal to the plane passing through the point. In general rays are constructed in sets, each set is specified by a single plane and a set of points. The **rz** command is used to create the points. The **regnpts** command is used to specify the plane, to specify the region, and to specify the number of points to be distributed along the rays. The points and the plane should lie outside the enclosing volume and on opposite sides. The normal to the plane should point toward the point. As rays are created, if they do not pass through the specified region, no points are distributed. Points may be spaced evenly along the ray or they may be spaced according to a ratio. The following commands will place points in the unit cube.

* create 25 points (5x5x1) in a plane above the unit cube
* place points on the boundaries in the x and y directions (1,1,0)

rz/xyz/5,5,1/0.,0.,1.1/1.,1.,1.1/1,1,0/

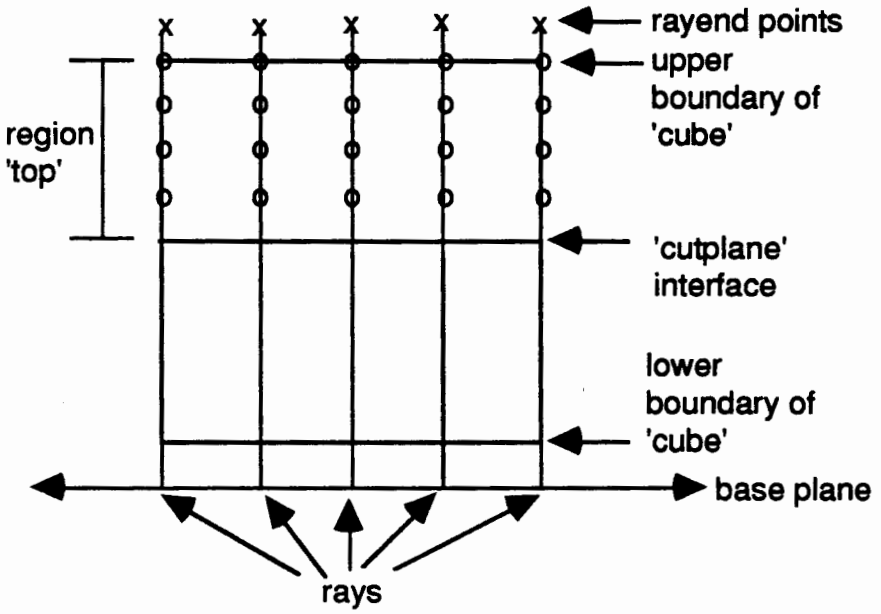
* give the points defined by the **rz** command the name, *rayend*

pset/rayend/**seq**/1,0,0/



* create rays between points in *rayend* and the plane below the cube
* distribute 3 points along these rays in the region *top*
* add one point at the upper external boundary for each ray
* will get 4 points total along each ray in region *top*
* "**pset**,**get**,rayend" refers to all the points named *rayend*
* the three points: (0.,0.,-.1), (0.,1.,-.1), (1.,1.,-.1)
* define a plane whose normal points toward the *rayend* points

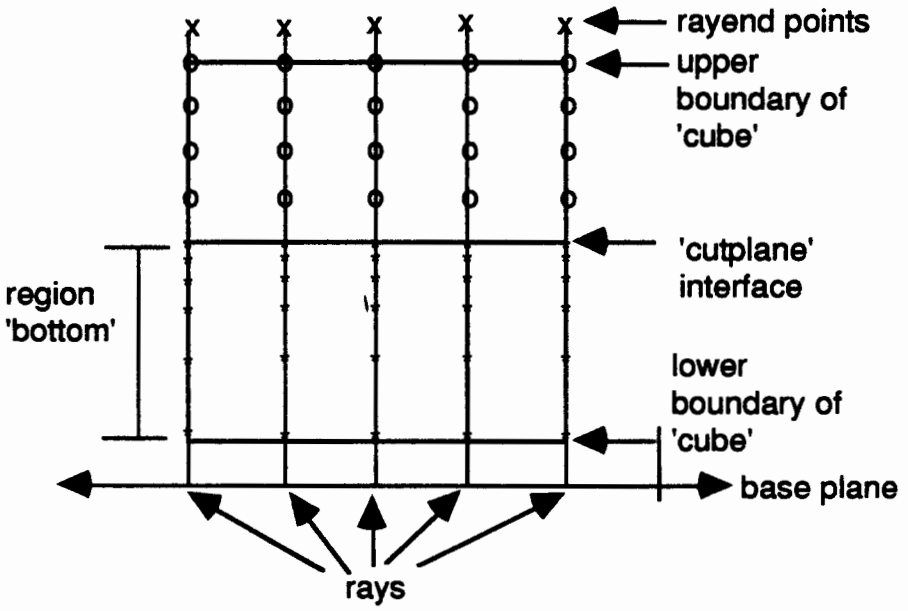
regnpts/top/3/**pset**,**get**,rayend/**xyz**/0.,0.,-.1/0.,1.,-.1/1.,1.,-.1/0,0/



front face of cube showing one row of 'rayend' points and one set of 5 rays. Points are distributed in the region 'top'.

* distribute 4 points along these rays in the region *bottom*
* add one point at the lower external boundary for each ray
* add one point at the material interface for each ray since
* *bottom* contains the interface - a total of 5 points for each ray.
* points will be distributed such that the ratio of distances between
* any two consecutive pairs of points is 0.6 traveling from the source
* of the ray (the plane) to the ray end.

regnpts/bottom/4/**pset**,**get**,rayend/**xyz**/0.,0.,-.1/0.,1.,-.1/1.,1.,-.1/1,.6/



front face of cube showing one row of 'rayend' points and one set of 5 rays. Points are distributed in the region 'bottom'.

Other versions of the **regnpts** are appropriate for cylindrical and spherical geometries. For cylindrical geometries the **rz** command specifies points in a cylindrical shell outside the volume. The **regnpts** command specifies a line (usually the cylinder axis), and the rays are constructed normal to this line and containing one of the **rz** points. For spherical geometries the **rz** command specifies points in a spherical shell outside the volume. The **regnpts** command specifies a point (usually the center of the sphere) from which rays are construted to the **rz** points.
If there are other regions that intrude on the region in which points are being distributed, then the effect is that of laying down a background distribution of points and erasing those that occur in the interior of the intruding regions.

7. Connect the points into tetrahedra

The mesh designer may use the following set of command to connect the points into a tetrahedral mesh:

* eliminate coincident or nearly coincident points
* 1,0,0 means consider all points

filter/1,0,0/

* *rayend* points are set to invisible (**dud** is the code for invisible)
* they were used as end points of the rays in the **regnpts** command

zq/ltp/pset,get,rayend/**dud/**

* assign material colors to the points
* identify points that are on material interfaces
* identify constrained points

setpts

* connect the points into a Delaunay tetrahedral mesh
* do not connect across material interfaces - add points if necessary to resolve material interfaces

search

* set element (tetrahedral) type
* spawn child points at material interfaces

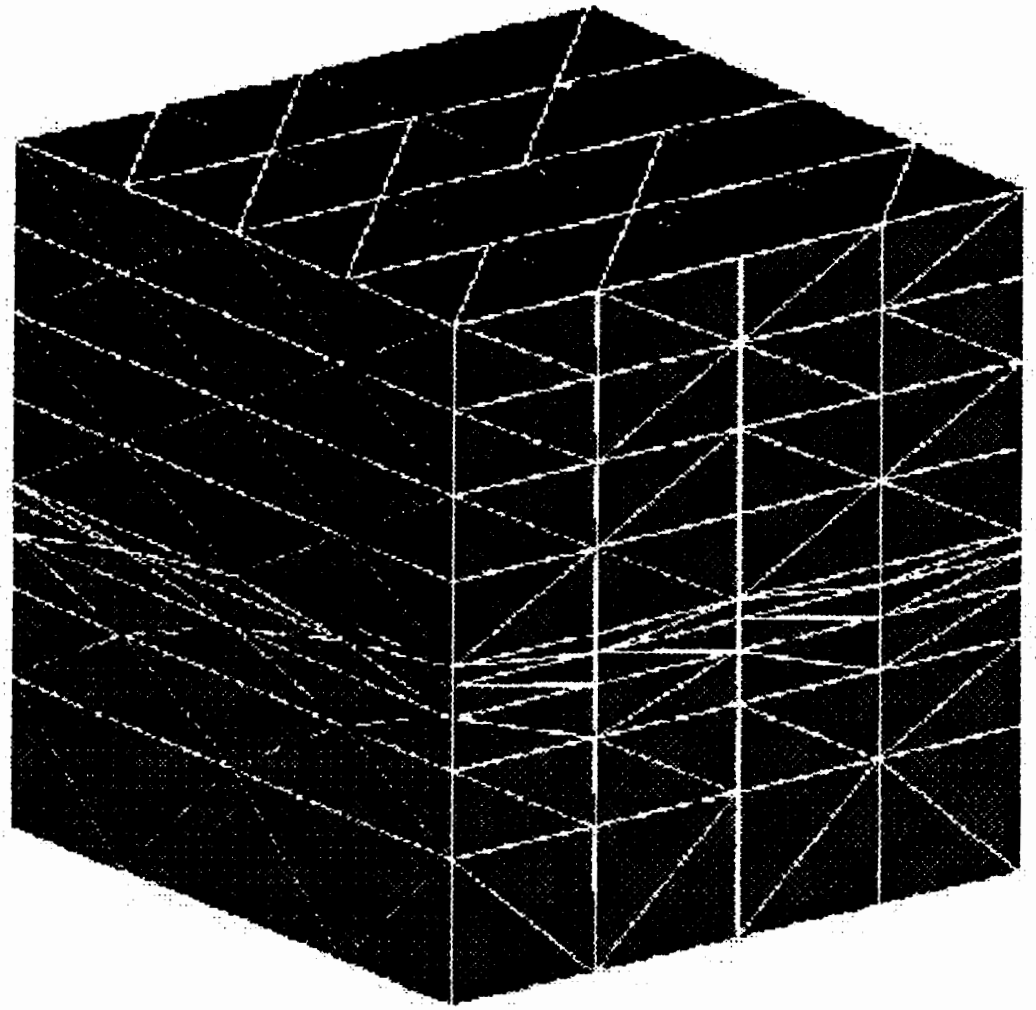
settets

* dump mesh to some output form

dump/?????/filename

* terminate processing

finish



The complete input for the tutorial is:

```
* create a 3D tetrahedral mesh object and name it 3dmesh
cmo/create/3dmesh/tet/
* unit cube
surface/cube/reflect/box/0.0,0.0,0.0/1.0,1.0,1.0/
* define z=.5 plane as interface
surface/cutplane/intrface/plane/0.,0.,.5/1.,0.,.5/1.,1.,.5/
*define geometric regions
region/top/ le cube and gt cutplane /
region/bottom/ le cube and le cutplane /
* define material regions
mregion/mattop/ le cube and gt cutplane /
mregion/matbot le cube and lt cutplane /
* create 25 points (5x5x1) in a plane above the unit cube
* place points on the boundaries in the x and y directions (1,1,0)
rz/xyz/5,5,1/0.,0.,1.1/1.,1.,1.1/1,1,0/
* give the points defined by the rz command the name, rayend
```

pset/rayend/**seq**/1,0,0/
* create rays between points in *rayend* and the plane below the cube
* distribute 3 points along these rays in the region *top*
* add one point at the upper external boundary for each ray
regnpts/top/3/**pset**,**get**,rayend/**xyz**/0.,0.,-.1/0.,1.,-.1/1.,1.,-.1/0,0/
* distribute 4 points along these rays in the region *bottom*
* add one point at the lower external boundary for each ray
* add one point at the material interface for each ray since
* *bottom* contains the interface - a total of 5 points for each ray.
* points will be distributed such that the ratio of distances between
* any two consecutive pairs of points is 0.6 traveling from the source
* of the ray (the plane) to the ray end.
regnpts/bottom/4/**pset**,**get**,rayend/**xyz**/0.,0.,-.1/0.,1.,-.1/1.,1.,-.1/1,.6/
* eliminate coincident or nearly coincident points
* 1,0,0 means consider all points
filter/1,0,0/
* *rayend* points are set to invisible (**dud** is the code for invisible)
* they were used as end points of the rays in the **regnpts** command
zq/**ltp**/**pset**,**get**,rayend/**dud**/
* assign material colors to the points
* identify points that are on material interfaces
* identify constrained points
setpts
* connect the points into a Delaunay tetrahedral mesh
* do not connect across material interfaces -
* add points if necessary to resolve material interfaces
search
* set element (tetrahedral) type
settets
* dump mesh to some output form
dump/?????/filename
* terminate processing
finish

II. Mesh Objects

a. Mesh Object Definition

The data structure which contains the information necessary to define a mesh is called a Mesh Object. There is a default template for a Mesh Object which consists of the following:

name (mesh object name)
nnodes (number of nodes in the mesh)
nelements (number of elements in the mesh, e.g. triangles, tetrahedra)
nfaces (number of uniques topological facets in the mesh, e.g number of edges in 2D or number of element faces in 3D) -- (not used)
nedges (number of uniques edges in mesh) -- (not used)
mbndry (value signifying that if the node number is greater that mbndry then the node is a boundary node)
ndimensions_topo (topological dimensionality,1, 2 or 3, i.e. a non-planar surface would have ndimensions_topo = 2 and ndimensions_geom = 3.)
ndimensions_geom (1, 2 or 3 for dimension of geometry)
nodes_per_element
edges_per_element
faces_per_element (topological number of facets per element (i.e. in 1D this number is always 2, for 2D use the number of edges of the element, for 3D use the number of faces of the element.)
isetwd (integer array containing pset membership information, see **pset** command definition)
ialias (integer array of alternate node numbers, i.e. for merged points)
imt1 (integer array of node material)
itp1 (integer array of node type - type ≥ 20 node will be invisible)

point type	name	description
0	int	interior
2	ini	Interface
10	rfl	Reflected boundary node
11	fre	Free boundary node
12	irb	Interface node on reflected boundary
13	ifb	Interface node on free boundary
20	mrg	Merged node
21	dud	Dudded node
41	par	Parent node

icr1 (integer array of constraint numbers for nodes)

isn1 (integer array of child, parent node correspondence)

Points on material interfaces are given point type 41 (parent). One child point is spawned for each material meeting at the parent point. The isn1 field of the parent point will contain the point number of the first child point. The isn1 field of the first child will contain the point number of the next child. The isn1 field of the last child will contain the point number of the parent. The point types of the child points will be 2, 12 or 13 depending on whether the interface point is also on an exterior boundary. This parent, child relationship is established by the **settets** command.

ign1 (integer array of generation numbers for nodes)

xic, yic, zic (real arrays of node coordinates)

itetclr (integer array of element material)

itettyp (geometry of element)

name	value	description
ifelmpnt	1	point
ifelmlin	2	line
ifelmtri	3	triangle
ifelmqud	4	quadrilateral
ifelmtet	5	tetrahedron
ifelmpyr	6	pyramid
ifelmpri	7	prism
ifelmhex	8	hexahedron

itetsetwd (integer array containing eset membership information, see **eset** command definition)

itetoff (index into itet array for an element)

jtetoff (index into jtet array for an element)

itet (integer array of node vertices for each element)

jtet (integer array of element connectivity)

b. Command Interface

The default Mesh Object is named *3dmesh* For simple problems the user must supply only a **cmo/create/**`mesh_object_name` command. There is no limit on the number of Mesh Objects that can be defined, but at any time there is only one 'current' or 'active' Mesh Object. For more advanced problems, such as those requiring more than one Mesh Object or requiring extensions to the basic Mesh Object template, the Mesh Object(s) is(are) manipulated via the **cmo** commands which are described in the next section. For example, additional

user defined components may be added to a Mesh Object by using the **cmo/addatt** command, or the 'active' Mesh Object can be changed using the **cmo/select** command..

c. FORTRAN Interface

Mesh Object data are accessed through a set of subroutines. An example of accessing an existing Mesh Object and creating a new mesh object is given in Section IV; that example should be used as a template when operating with Mesh Objects. The subroutine set includes:

`cmo_get_name`	retrieve active mesh object name
`cmo_set_name`	set active mesh object name
`cmo_get_info`	retrieve mesh object data
`cmo_set_info`	set mesh object data
`cmo_newlen`	adjust mesh object memory allocation

Only data from the active Mesh Object may be retrieved; calling `cmo_set_name` will make the Mesh Object active. Scalar quantities are retrieved and stored using `cmo_get_info` and `cmo_set_info`. Vector quantities are referred to by their pointers. The length of the vectors is calculated internal to X3D based on the scalar mesh object variables. Memory allocation for a new mesh object or for a mesh object which will grow in size is accommodated by setting the appropriate scalars for the Mesh Object and calling `cmo_newlen`. `cmo_set_info` with the new number of elements or nodes and `cmo_newlen` must be called before adding to the size of a Mesh Object.

III. X3D Commands:

a. Conventions

Following in Section III.b is list of the X3D commands. These commands are given in alphabetic order. Conventions that apply to all commands include:

1. Lines are a maximum of 80 characters long, identifiers are a maximum of 32 characters long.
2. Continuation lines are signaled by an "&" as the last character of a line to be continued. A command can be up to 1024 characters long.
3. Delimiters are comma, slash equal sign or blank. (',' '/' '=" ').

Blanks on either side of other delimiters are ignored. Leading blanks are ignored. Commas are usually used for parameters that belong to the same logical set such as `first point, last point, stride`. Slashes are usually used to separate sets of parameters.

4. The three parameters: `first point, last point, stride` can have integer values which refer to actual sequential point numbers or they can have the character-string values:

pset, get, `name` where `name` has been defined by a previous **pset** command. The triplet: 1, 0, 0 refers to all points. The triplet: 0, 0, 0 refers to the set of points defined in the last geometry command.

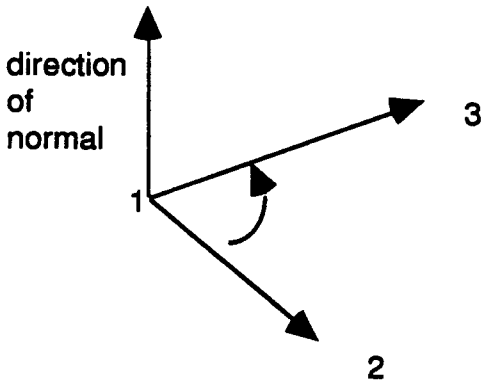
5. Commands should be typed in lower case, however names are case sensitive.

In the command description that follows certain symbols have special meaning.

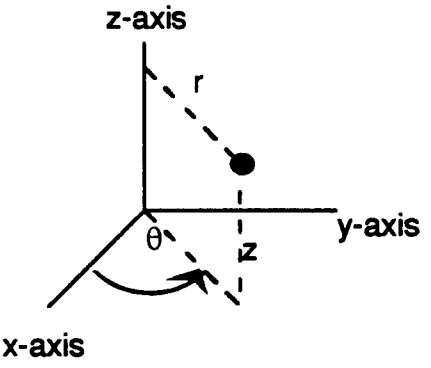
[]	surround optional parameters
|	signifies alternate choices
, or /	separates parameters

6. Courier font is used for variable names such as `ifirst`.
   bold is used for literals such as **xyz**.
7. Comments are identified by * in the first column.
8. All names (surface, region, pset,...) should be limited to 32 characters.
9. The right hand rule is used to determine normals to planes and to sheet surfaces.

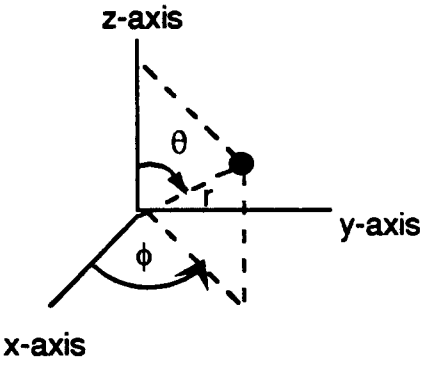
The first two points determine the first vector and the first and third point determine the second vector. By curling the fingers of the right hand from the first vector toward the second vector, the right thumb will point in the direction of the normal.



10. To separate commands on the same line use a semicolon (;).
11. Three coordinate systems are used.
    xyz refers to the standard Cartesian coordinate system
    rtz refers to a cylindrical coordinate system aligned along the z-axis, where **r** is the radius measured from the zaxis, **t** (theta) is the angle measured in the xy-plane from the positive x-axis toward the positive y-axis and **z** is the height measured from the xy-plane.
    rtp refers to a spherical coordinate system, where **r** is the radius measured from the origin, **p** (phi) is the angle in the xy-plane measured from the positive x-axis toward the positive y-axis, **t** (theta) is the angle measured from the positive z-axis to the positive y axis.



cylindrical coordinates
rtz
position of ●
determined by
r radius
t theta (angle from x-axis)
z height



spherical coordinates
rtp
position of ●
determined by
r radius measured from origin
t theta (measured from positive z-axis)
p phi (measured from positive x-axis)

b. Alphabetic Listing of X3D Commands

ADDMESH

This routine glues two meshes together at their common interface to produce a third mesh.

FORMAT:

addmesh / glue / mesh3 / mesh1 / mesh2 /

ASSIGN

Assign a value to a code variable.

FORMAT:

assign/category_name/column/variable_name/value.

EXAMPLES:

assign/-def-/-def-/epsilonl/1.e-4/

CMO

The **cmo** command operates on the `Mesh Object (MO)`. There can be many `Mesh Objects` in the code for a given problem. Only one of these `Mesh Objects` may by the `Current Mesh Object`. There is also one `Default Mesh Object` which is used as the template for generating new `Mesh Objects`.

FORMAT:

cmo/addatt /mo_name/att_name/type/rank/length/interpolation/
persistence/io/value/
compress/mo_name/
copy/mo_name/master_mo/
create/mo_name/npoints/nelements/mesh_type/
create/mo_name/npoints/nelements/ndimensions_geom/ndimensions_topo/
nodes_per_element/faces_per_element/edges_per_element/
delatt/mo_name/att_name/
derive/mo_name/master_mo/
length/mo_name/att_name/
list
memory/mo_name/number_nodes/number_elements/
modatt/mo_name/att_name/field_name/new_field/
move/mo_name/master_mo/

newlen/mo_name/
release/mo_name/
select/mo_name/
status/mo_name/
verify/mo_name/

CONVENTION: As a result of any command that generates a new Mesh Object, the newly generated Mesh Object becomes the Current Mesh Object.

RESERVED NAMES: The following names are reserved and may not be used for Mesh Object names:

-cmo-	the Current Mesh Object
-default-	the Default Mesh Object
-all-	all Mesh Objects or Attributes
-notset-	Null name for Mesh Object

TYPES ,DEFAULTS and POSSIBLE VALUES:

mo_name	is type character
att_name	is type character
mesh_type	is type character **(tet,hex,prl,pyr,trl,qua,hyb)**
type	is type character, default is **VDOUBLE** **(VDOUBLE, INT, VINT)**
rank	is type character, default is **scalar** **(scalar,vector,tensor)**
length	is type character, default is **nelements** **(nelements, nnodes)**
interpolate	is type character, default is **user** **(copy, sequence, linear, log, asinh, max, min, user)**
persistence	is type character, default is **temporary** **(permanent, temporary)**
io	is type character, default is **no** **(a, g, f, s, x, no** -- for avs,gmv,fehms,sgi,x3d)
value	is type real, default is 0.0
npoints	is type integer, default is Default MO

nelements	is type integer, default is Default MO
ndimensions_geom	is type integer, default is Default MO
ndimensions_topo	is type integer, default is Default MO
nodes_per_element	is type integer, default is Default MO
faces_per_element	is type integer, default is Default MO
edges_per_element	is type integer, default is Default MO

addatt/mo_name/att_name/type/rank/length/interpolate/persistence/io/value/

mo_name **required**

att_name **required**

Adds the Attribute, att_name, **to** Mesh Object, mo_name. **See the modatt command for more details on the field variables.**

Examples:

cmo/addatt/cmo1/boron1**/VDOUBLE/scalar/nnodes/asinh/permanent/gx/**1.0

cmo/addatt/-cmo-/boron2**/VDOUBLE/scalar/nnodesasinh/permanent/gx/**2.0

cmo/addatt/cmo1/boron3**/VDOUBLE/scalar/nnodes/user/temporary/gx/**3.0

cmo/addatt/-default-/boron**/VDOUBLE/scalar/nnodes/asinh/temporary/no/**3.

cmo/addatt/-default-/boron3

compress/mo_name/

mo_name **is type character, default is -cmo-**

Shortens all memory managed arrays for Mesh Object mo_name **to their actual lengths.**

Examples:

cmo/compress/mo_tet2

cmo/compress/-cmo-

cmo/compress

cmo/compress/-all-

copy/mo_name/master_mo/

mo_name **is type character, required.**

master_mo **is type character, default is '-cmo-'**

Makes an exact copy of `Mesh Object, master_mo,` including all data. The output `Mesh Object, mo_name,` will become the `Current Mesh Object.` If `mo_name` is the same as `master_mo` nothing happens.

If `mo_name` exists it is over written.

Examples:

cmo/copy/`mo_tet2/mo_tet1`

cmo/copy/-cmo-/`mo_tet1`

cmo/copy/`mo_tet2`

cmo/copy/`mo_tet2`**/-cmo-**

create/`mo_name/npoints/nelements/` **tet|hex|pri|pyr|tri|qua|hyb/**

or

create/`mo_name/npoints/nelements/ndimensions_geom/ndimensions_topo/`
`nodes_per_element/faces_per_element/edges_per_element/`

Creates a new Mesh Object '`mo_name`', which becomes the `Current Mesh Object.`

If a Mesh is created using the first (mesh_type) format, then values are supplied for the other parameters as follows:

mesh name	ndimension geom	ndimension topo	nodes_per element	faces_per element	edges_per element
tet	3	3	4	4	6
hex	3	3	8	6	12
pri(sm)	3	3	6	5	9
pyr(amid)	3	3	5	5	8
tri(angle)	3	2	3	3	3
qua(d)	3	2	4	4	4
hyb(rid)	3	3	8	6	12

If `mo_name` exists nothing happens.

`mo_name` required.

Examples

cmo/create/`mo_tet2`

cmo/create/`mo_tet2/0/0/`**hex**

delatt/`mo_name/att_name/`

Deletes the attribute `att_name` from `Mesh Object, mo_name.` Will not delete an attribute with a **`persistence`** of **permanent**

`mo_name` must be specified.

`att_name` must be specified.

Examples

cmo/delatt/mo_tet2/boron

cmo/delatt/-cmo-/boron

cmo/delatt/-default-/boron

cmo/delatt/-cmo-/-all- this will delete all attributes with **persistence** of **temporary**

derive/mo_name/master_mo/

mo_name is type character, required.

master_mo is type character, default is **-cmo-**

Uses Mesh Object, master_mo, as the template for deriving Mesh Object, mo_name. Mesh Object, mo_name, will be an image of master_mo but will contain no data. The output Mesh Object, mo_name, will become the Current Mesh Object. If mo_name is the same as master_mo nothing happens. If mo_name exists it is over written.

Examples:

cmo/derive/mo_tet2/mo_tet1

cmo/derive/-cmo-/mo_tet1

cmo/derive/mo_tet2

cmo/derive/mo_tet2/**-cmo-**

cmo/derive/-default-/-cmo-

cmo/derive/mo_tet2/**-default-**

cmo/derive/-default-/mo_tet1

length/mo_name/att_name/

mo_name is type character, default is **-all-**

att_name is type character, default is **-all-**

Returns the memory length of attribute att_name for Mesh Object, mo_name.

Examples:

cmo/length/mo_tet2/boron

cmo/length/-cmo-/boron

cmo/length/-default-/boron

cmo/length/-cmo-/-all-

cmo/length/mo_tet2/**-all-**

cmo/length
cmo/length/-all-/-all-
cmo/length/-all-/boron

list

Returns the name of the Current Mesh Object and a list of all defined Mesh Objects

Examples:

cmo/list

memory/mo_name/number_nodes/number_elements/

Allows the user to preset the size of the memory managed arrays for Mesh Object, mo_name.

mo_name required.

number_nodes required.

number_elements required.

Examples:

cmo/memory/mo_tet2/1000/10000

cmo/memory/-cmo-/1000/10000

modatt/mo_name/att_name/field_name/new_field/

Modifies the field field_name for attribute att_name in Mesh Object mo_name.

mo_name required.

att_name required.

field_name is type character, required

new_field is the type of the field, required

Field_names (may be lower or upper case):

name	- (character) Attribute name
type	- (character) Attribute type:
	INT- Integer
	VINT - Vector of integer
	VDOUBLE - Vector of real*8
rank	- (character) Attribute rank (must be an attribute for this Mesh object)
length	- (character) Attribute length (must be an attribute for this Mesh object)

interpolation - (character) Interpolation option:
- **constant** - Constant value
- **sequence** - Next is previous plus 1
- **copy** - Copy values
- **linear** - Linear interpolation
- **user** - User provides
- **log** - Logarithmic interpolation
- **asinh** - Asinh interpolation

persistence - (character) Attribute persistence:
- **permanent** - Can not be deleted
- **temporary** - Temporary attribute

io - (character) Attribute IO flag:
- **a** - Put this attribute on avs dumps
- **g** - Put this attribute on gmv dumps
- **f** - Put this attribute on fehms dumps
- **s** - Put this attribute on sgi dumps
- **x** - Put this attribute on x3d dumps
- **no** - Ignore this attribute

value (real) Attribute value

Examples:

cmo/modatt/mo_tet2/boron/**length/**nnodes
cmo/modatt/-cmo-/boron/**length/**nnodes
cmo/modatt/-cmo-/boron/**value/**10.0
cmo/modatt/-default-/boron/**value/**10.0

move/mo_name/master_mo/

mo_name **is type character, required.**

master_mo **is type character, default is -cmo-'**

Changes the name of Mesh Object, master_mo, **to** mo_name. **The output** Mesh Object, mo_name, **will become the** Current Mesh Object. **If** mo_name **is the same as** master_mo **nothing happens. If** mo_name **exists it is over written.**

Examples:

cmo/move/mo_tet2/mo_tet1
cmo/move/-cmo-/mo_tet1

cmo/move/mo_tet2
cmo/move/mo_tet2**/-cmo-**

newlen/mo_name**/**
mo_name **is type character, default is -cmo-**
Changes the length of all memory managed arrays for Mesh Object mo_name **to the proper length.**

Examples:
cmo/newlen/mo_tet2
cmo/newlen/-cmo-
cmo/newlen

release/mo_name**/**
mo_name **is type character, required.**
Deletes the Mesh Object mo_name.

Examples:
cmo/release/mo_tet2
cmo/release/-cmo-

select/mo_name**/**
mo_name **is type character, required.**
Selects Mesh Object mo_name **to be the Current** Mesh Object. **If** mo_name **does not exist a new** Mesh Object **will be created using the** Default Mesh Object **as the template.**

Examples:
cmo/select/mo_tet2

status/mo_name**/**
mo_name **is type character, default is '-all-'**
Prints the status of Mesh Objects.

Examples:
cmo/status/mo_tet2

cmo/status/-cmo-
cmo/status
cmo/status/-all-
cmo/status/-default-

verify/mo_name/

mo_name is type character, default is '**-all-**'

Verifies that Mesh Object mo_name is consistent.

Examples:

cmo/verify/mo_tet2
cmo/verify/-cmo-
cmo/verify
cmo/verify/-all-
cmo/verify/-default-

COPYPTS

Copy a point distribution. There are two distinct forms of this command. The first format is designed to copy points from one mesh object into another. In this form if the names of the source and sink mesh objects are are omitted, the current mesh object will be used. The copy may be restricted to a subset of points by including the source point information. Points in the sink mesh object will be overwritten if sink_stride is not zero. Attribute fields may be specified for both the source and sink mesh object. For example the x-coordinate field in the source mesh object (xic) may be placed in the y-coordinate field of the sink mesh object. Attribute values will be copied from the source mesh object to the sink mesh object. The user is warned that these values might not make sense in their new context.

The second form of this command is included for historic reasons: it duplicates points within a mesh object including all the attributes of the points. Also note that if no sink points or sink stride are specified, then the copied points are placed at the end of the data arrays (see third FORMAT) otherwise the copied points are written over the existing points starting at the 1st sink point. Note also that the first form of the command gives the arguments sink first then source whereas the second for give the source then the sink.

FORMAT:

copypts/sink_cmo/source_cmo/1st_sink_point/ sink_stride/
1st_source_point/last source point/source_stride
/sink_attribute/source_attribute

copypts/1st source point/last source point/stride/1st sink point/ sink stride /

copypts /1st point/last point/stride/

EXAMPLES:

copypts/3dmesh/2dmesh /

Copy all points in 2dmesh to the end of the 3dmesh point list.

copypts/3dmesh/2dmesh/0,0/**pset,get,**mypoints/

Copy the point set named *mypoints* from 2dmesh to the end of 3dmesh point list.

copypts/3dmesh/2dmesh/100,4/**pset,get,**mypoints/boron/arsenic/

Copy the arsenic field from the point set named mypoints from 2dmesh replacing the boron field at every fourth point beginning at point 100 in 3dmesh.

copypts/pset,get,mypoints/0,0/

Duplicate the point set named mypoints from the current mesh object and place the duplicated points at the end of the point list.

copypts///0,0/**pset,get,**mypoints/

Duplicate the point set named mypoints from the current mesh object and place the duplicated points at the end of the point list. Same effect as the example directly above. The current mesh object is used since the fields are blank on the command line

COORDSYS

This routine defines a local coordinate system to be in effect until another coordinate system is defined or the normal coordinate system is reset. The new coordinate system is defined by specifying an origin, a point on the new x-z plane and a point on the new z-axis. these points are specified in the normal coordinate system. the options available in iopt are:

define	define a new local coordinate system
normal	return to the normal coordinate system
save	save the current coordinate system for recall
restore	recall the last saved coordinate system

FORMAT:

coordsys/iopt/x0,y0,z0/xx,xy,xz/zx,zy,zz/

where x0,y0,z0 is the location of the new origin, xx,xy,xz is a point on the new x-z plane and zx,zy,zz is a point on the new z-axis. These points are defined with the normal coordinate system, and used only with the define option.

DOPING

Create doping profile for the grid.

A constant profile is invariant over the specified region with value xcon.

A gaussian profile contains a bounding box (x1,y1,z1) to (x2,y2,z2) where the peak concentration will be. z1=z2 in 2D. The doping varies according to the gaussian distribution:

doping = concentration * exp(-(L/std_dev)**2)

where L is the effective distance and can be represented as:

L = sqrt(dy**2 + (1/lateral_diffusion)*(dx**2 + dz**2))

where

dy = y-y1 (or y2 for that matter)

dx = x-x1 if x < x1 < x2

= 0 if x1 < x < x2

= x-x2 if x1 < x2 < x

dz similar to dx.

The table option reads in doping data that has been read into the mesh object, cmo_table_name, as the attribute, att_table_name. For example, the data may be read from a DATEX2.1 format file by a previously issued **read/datex** command. The fields cmo_geometry and table_geometry give the mapping from the domain on which the doping field was input to the active mesh object domain. The values of these fields may be **xy, yz, yx, xz, zx, zy** for 2D geometries and **xyz, xzy, yxz, yzx, zxy, zyx** for 3D geometries. The last example given below transforms 2D data input as **zy** data to the **yx** coordinate system.

In all cases, field specifies the name of a defined attribute field in the active mesh object.

FORMAT:

doping/constant/field/**set|add|sub/**ifirst,ilast,istride/xcon

doping/gaussian/field/**set|add|sub/**ifirst,ilast,istride/**xyz | rtz |rtp/**
x1,y1,z1/x2,y2,z2/lateral_diffusion/concentration/std_dev/

doping/table/field/**set|add|sub/**ifirst,ilast,istride/cmo_table_name/
att_table_name/**linear|log|asinh/**cmo_geometry/table_geometry/

EXAMPLE:

doping/constant/ric**/set/pset,get,**Silicon**/-1.0e+15/**

doping/gaussian/ric**/add/pset,get,**Silicon**/xyz/**

0.0,0.08,0.0/0.6,0.08,0.0/0.5/5.0e+18/0.225/

doping/table/pic**/set/**1,0,0/cmo1/pic**/linear/xyz/xyz/**

doping/table/Saturation **/set/**1,0,0/cmo_course/Saturation/

linear/zx/yx/

DUMP

This command produces an output file from a Mesh Object. If the option is **x3d**, a restart dump is made a subsequent **read/x3d** will restart the code at the state that the dump was taken.

FORMAT:

dump/file_type/file_name/[cmo_name]/

valid file_types are: **x3d**, **gmv**, **avs**, **chad**, **fehm**, and **datex**

EXAMPLE:

dump/gmv/gmv.out/3dmesh/

dump/x3d/x3d.out/

EXTRACT

This command produces a 2D mesh object from a 3D mesh object. A material interface, a plane or an iso-surface may be extracted. A plane may be defined by three points in the plane, by a vector normal to the plane, by three points on the axes of the space, or by the coefficients of the plane equation ax+by+cz=d. An isosurface is defined by the value of the surface and the mesh object field to test for this value. An interface is defined by the material(s) bounding the interface. region1, [region2] are the material numbers or the material region names whose interface is to be extracted. Use **-all-** to extract from all interfaces. All variations of the command can be limited by the usual pset syntax. The output 2D mesh object is cmoout, the input 3D mesh objects is cmoin.

The output MO will be oriented such that the outward normal of the plane that defines the surface will point in the same direction as the normals for the triangles in the output MO. If the command extracts on an isosurface, the output MO will be oriented such that the normals for the triangles point in the direction of increasing field. If the command extracts on an interface, the output MO triangles will be oriented the same as the triangles extracted from region1 of the input MO. In the case of a plane extracted along all or a

portion of a material interface, only those points that lie inside the material (i.e.: away from the direction of the normal) will be picked up. If the extraction is on a boundary, the normal to the extraction plane must point out of the material in order for points to be picked up.

FORMAT:

extract/plane/threepts/x1,y1,z1/x2,y2,z2/x3,y3,z3/
ifirst,ilast,istride/cmoout/cmoin
/ptnorm/x1,y1,z1/xn,yn,zn/ ifirst,ilast,istride/cmoout/cmoin
/axes/xv,yv,zv/ifirst,ilast,istride/cmoout/cmoin
/abcd/a,b,c,d/ ifirst,ilast,istride/cmoout/cmoin
/isosurf/var/value/ ifirst,ilast,istride/cmoout/cmoin
/intrface/region1/ ifirst,ilast,istride/cmoout/cmoin
/intrfac2/region1/region2/ifirst,ilast,istride/cmoout/cmoin

FIELD

The FIELD Command option manipulates one or more specified fields in the Current Mesh Object

- For all points in the specified point set, we **compose** the field value with the specified composition function. The composition functions allowed are currently **asinh** and **log**. So, for example, if 'i' is in the point set and **asinh** is the composition function, we have the assignment:

  field(i) = asinh(field(i)).

- The **field/mfedraw** command causes a binary dump of the specified fields to two files in the **mfedraw** input format. **mfedraw** is a graphics package for visualizing moving piecewise linear functions of two variables, such as those originally encountered in Moving Finite Elements. The files are named 'root1.bin' and 'root2.bin', where 'root' is the root file name argument. Because the graphics data are a function of two variables, you must supply two orthonormal vectors (x1,y1,z1) and (x2,y2,z2) which specify the graphics coordinate axes. More precisely, given 3D coordinates (x,y,z), the 2D graphic coordinates will then be (x*x1+y*y1+z*z1 , x*x2+y*y2+z*z2). So, for example, the choice:

  /x1,y1,z1/x2,y2,z2/ = /1.,0.,0./0.,1.,0./

causes the 'z' coordinate to be discarded while the 'x' and 'y' coordinates are unchanged.

- The **field/scale** option scales the field values of the specified points. scale option can take on the values **normalize**, **multiply**, and **divide**. If **normalize** is specified, we

multiply all the field values by **factor**/(fieldmax-fieldmin), where 'fieldmax' and 'fieldmin' are the maximum and minimum values taken over the point set. This has the effect of normalizing the field so that the new difference between the maximum and minimum values is equal to **factor**. If **multiply** is specified, we multiply all the field values in the point set by **factor**. If **divide** is specified, we divide all the field values in the point set by **factor**.

• The **field/volavg** option, for all the members of the point set and for all specified fields, replaces the point field values with values that represent the average of the field(s) over the control volumes associated with the points. The `averaging option` specifies what kind of control volume is to be used; the choices are **voronoi** and **median**. `iterations` is an integer that specifies a repeat count for how many times this procedure is to be performed on the field(s). The affect of this process is to broaden and smooth the features of the field(s), similar to the effect of a diffusion process. The **voronoi** choice, unlike the **median**. choice, produces a diffusive effect independent of mesh connectivity. However, again unlike the **median**. choice, it requires that the mesh be Delaunay, or incorrect results will occur.

FORMAT:

field/compose/`composition function/ifirst,ilast,istride/field list/`
field/mfedraw/`root file name/x1,y1,z1/x2,y2,z2/field list/`
field/scale /`scale option/factor/ifirst,ilast,istride/ field list/`
field/volavg/`averaging option/iterations/ifirst,ilast,istride/field list/`

EXAMPLE:

field/compose/asinh/`1,0,0/pressure/`
field/scale/factor/4.0/`pset,get,region1/boron/`
field/volavg/voronoi/`4/1,0,0/boron/`

FILTER

Used to filter (delete) points that are too close (default==> `tolerance`=5.0E-06), closer than the `tolerance` specified by the user, or duplicate points. This command records the deleted points as dudded out points (itp=21) and places their position at infinity but does not remove them from the point list. Note that at least one point must be specified in the point sequence numbers (`ifirst,ilast,istride`) in order for this command to work properly.

FORMAT:

filter / `ifirst,ilast,istride` / [`tolerance`]

FINISH

Terminate processing this set of command and return to the driver routine.

FORMAT:

finish

HELP

Access help package. **help**/`command` will return the command description. **help**/`code_variable` will return the variable definition. **help** with no arguments will return a list of commands and variables.

FORMAT:

help/[`variable_name|command_name`]

EXAMPLES:

help

help/surface

help/ipointi

INFILE

INPUT

These commands instruct X3D to begin processing commands from a file. The **infile** commands may be nested. Only the outermost set of commands should be terminated with a **finish** command

FORMAT:

infile/`file_name`

input/`file_name`

INTERSECT

Creates a new Mesh Object from the intersection of two existing Mesh Objects. The existing Mesh Objects have to be topologically 2D and geometrically 3D. The created Mesh Object will be topologically 1-D and geometrically 3D. Node quantities for the new Mesh Object will be create by interpolation on the corresponding node quantities of the first input Mesh Object, `cmo_1_in`.

FORMAT:

intersect/cmo_out/cmo_1_in/cmo_2_in

LOG

Turn the batch output file and tty output file **off** and **on**. The tty prints to and reads from the user's screen. The batch file is the output file called outx3dgen. Default is **on** for both files.

FORMAT:

log/bat|tty/on|off/

EXAMPLE:

log/tty/off

MEMORY

Initialize code and set aside memory. Use **cmo/memory** instead.

FORMAT:

memory/nnodes/ntets

nnodes and ntets are initial guesses for number of nodes and number of tetrahedra

MREGION

Define a material region from a set of surfaces by logically combining the surface names and region names. The operators **lt**, **le**, **gt**, and **ge** are applied to previously defined surfaces according to the following rules.

lt -- if the surface following is a volume then **lt** means inside not including the surface of the volume. If the surface is a plane or a sheet **lt** means the space on the side of the plane or sheet opposite to the normal not including the plane or sheet itself.

le -- if the surface following is a volume then **le** means inside including the surface of the volume. If the surface is a plane or a sheet **le** means the space on the side of the plane or sheet opposite to the normal including the plane or sheet itself.

gt -- if the surface following is a volume then **gt** means outside not including the surface of the volume. If the surface is a plane or a sheet **gt** means the space on the same side of the plane or sheet as the normal not including the plane or sheet itself.

ge -- if the surface following is a volume then **ge** means outside including the surface of the volume. If the surface is a plane or a sheet **ge** means the space on the same side of the plane or sheet as the normal including the plane or sheet itself.

The operators **or**, **and**, and **not** applied to regions or surfaces mean union, intersection and complement respectively. The operators **or**, **and**, and **not** applied to relational

operators are the normal logical operators. Parentheses are used for nesting. Spaces are required as delimiters to separate operators and operands. Internal interfaces should be excluded when defining material regions. (i.e. use **lt** and **gt**). External boundaries should be included when defining material regions.

FORMAT:

mregion/material_region_name/region definition

EXAMPLES.:

mregion/mat1/ **le** box1 **and** (**lt** sphere1 **and** (**lt** plane1 **or gt** plane2)) /

mregion/mat2/regiona **or** regionb

PSET

Give a name to a selected set of points.

union, **inter** and **not** are logical operations on previously defined psets.

list lists all psets

delete deletes a previously defined pset

zq forms a pset from all points in ifirst,ilast,istride which have value flag for the attribute point flag.

FORMAT:

pset/pset name/

seq/ifirst,ilast,istride

union|inter|not|delete/pset1[/pset2/.../psetn]

list

zq/point flag/ifirst,ilast,istride/flag

region/region name/ifirst,ilast,istride

geom/xyz/ifirst,ilast,istride/x1,y1,z1/xu,yu,zu/xc,yc,zc

READ

This command reads in data into the active Mesh Object, replacing whatever data might have been previously contained in the active Mesh Object. If the option is **x3d** the code reads in a restart dump. The **avs** option includes the choice of reading in nodes, elements and attributes by giving flags values of 1 (read) or 0 (skip) for the categories: node_ flag, element_flag and attribute_flag. This option requires that the mesh object name be specified.

FORMAT:

read/avs|dcm|datex|x3d/file_name

read/avs/file_name/cmo_name/node_ flag/element_flag/attribute_flag/

read/x3d/file_name/[dump_name/region_name/]
read/ngp/tet|hex|quad|tri/file_name/

EXAMPLE:
read/x3d/myfile

RECON

This command flips connections in the mesh to get restore the Delaunay criterion. The default is to add points on the boundaries if needed (**yes**). The option **no** specifies that no points are to be added on the boundaries.

FORMAT:
recon/[yes|no/]

REFINE

The **refine** command is used to create more elements when a criterion is specified.
The choice to refine is based on one of the following refine_criterion:

junction will refine object where field = 0.0
constant will refine object where field = xrefine
delta will refine object where Δ field = xrefine
lambda will refine object where λ (field) =xrefine
maxsize will refine object where object > xrefine
maxsize refers to volume for -- area for face, length for edges
minsize will derefine object where object < xrefine, (not implemented)
aspect will refine where as----- ratio > xrefine

The refine_type specifies what object will be refined and how that object will be refined:

tet will refine elements by placing a point in the center of the element.
face will refine facets by placing a point in the center of the facet.
edge will refine edges by placing a point on the midpoint of the edge.
faceedge will refine facets by refining all edges of the facet.
tetedge will refine elements by refining all edges of the element.

The field must refer to a previously defined attribute of the current Mesh Object.
The interpolation specifies how to interpolate the field to give field values to the new nodes created. The implemented values are.

linear
log

asinh

FORMAT:

refine/refine_criterion/field/interpolation/refine_type
/ifirst,ilast,istride /xrefine

REGION

Define a geometric region from the set of surfaces by logically combining the surface names. The operators **lt**, **le**, **gt**, and **ge** are applied to previously defined surfaces according to the following rules.

lt -- if the surface following is a volume then **lt** means inside not including the surface of the volume. If the surface is a plane or a sheet **lt** means the space on the side of the plane or sheet opposite to the normal not including the plane or sheet itself.

le -- if the surface following is a volume then **le** means inside including the surface of the volume. If the surface is a plane or a sheet **le** means the space on the side of the plane or sheet opposite to the normal including the plane or sheet itself.

gt -- if the surface following is a volume then **gt** means outside not including the surface of the volume. If the surface is a plane or a sheet **gt** means the space on the same side of the plane or sheet as the normal not including the plane or sheet itself.

ge -- if the surface following is a volume then **ge** means outside including the surface of the volume. If the surface is a plane or a sheet **ge** means the space on the same side of the plane or sheet as the normal including the plane or sheet itself.

The operators **or**, **and**, and **not** applied to surfaces mean union, intersection and complement respectively. The operators **or**, **and**, and **not** applied to relational operators are the normal logical operators. The parentheses operators, (and), are used for nesting. Spaces are required as delimiters to separate all operators and operands. Internal interfaces should be included in exactly one region.

FORMAT:

region/region_name/region definition

EXAMPLES:

region/reg1/**le** sphere1 **and** (**lt** plane1 **or gt** plane2)

region/reg2/**le** sphere1 **and** (**ge** plane1 **and le** plane2)

REGNPTS

Generates points in a region previously defined by the **region** command. The points are generated by shooting rays through a user specified set of points from an origin point, line, or plane and finding the intersection of each ray with the surfaces that define the region. The point distribution is determined by the data in `ptdist`. If `ptdist` is integer, then that many points are evenly distributed along the ray in the region. If `ptdist` is real, then points are distributed at that distance along the ray within the region. The variables `irratio` and `rrz` determine ratio zoning when `ptdist` is an integer. Ratio zoning is on when `irratio` is 1, then the distribution is adjusted by the value for `rrz`. When `irratio` is 2, the points are distributed by equal volumes depending on the geometry. When `irratio` is 3, ratio zoning is calculated on the longest ray then this length distribution is applied to all the rays. See the description of the command **surface** for a discussion of point distributions with respect to sheet surfaces.

FORMAT:

regnpts/`region name/ptdist/ifirst,ilast,istride/geom/`
`ray origin/irratio,rrz`

regnpts/`region name/ptdist`/**pset**,**get**,`setname/geom/ray origin`
`/irratio,rrz`

Where `ifirst,ilast,istride` or **pset**,**get**,`setname` define the set of points to shoot rays through.

EXAMPLES:

regnpts/`region name/ptdist/ifirst,ilast,istride/`**xyz**
`/x1,y1,z1/x2,y2,z2/x3,y3,z3/irratio,rrz/`

Where points 1, 2, 3 define the plane to shoot rays from that are normal to the plane.

regnpts/`region name/ptdist/ifirst,ilast,istride/`
rtz`/x1,y1,z1/x2,y2,z2/irratio,rrz/`

Where points 1, 2, define the line from which to shoot perpendicular rays

regnpts/`region name/ptdist/ifirst,ilast,istride/`
rtp`/xcen,ycen,zcen/irratio,rrz`

Where `xcen,ycen,zcen` define a point from which to shoot rays .

regnpts/`region name/ptdist/ifirst,ilast,istride/`**points**`/`
`iffirst,iflast,ifstride/`

Where `iffirst,iflast,ifstride` define a set of points from which to shoot rays

RM

Removes any points that are within the specified point range and specified piece of space. This is done in Cartesian (X, Y, Z), cylindrical (R, THETA, Z), or spherical (R, THETA, PHI) coordinates. It should be noted that in cylindrical coordinates, theta is the angle in the XY- plane with respect to the x-axis, while in spherical coordinates `theta` is the angle with respect to the Z-axis and `phi` is the angle in the XY-plane with respect to the X-axis. In cylindrical coordinates the cylinder always lines up along the z axis; use the **coordsys** command before issuing the **rm** command if the points to be removed are not aligned with the z-axis; then issue a final **coordsys** command to return to normal. Also note that the points that are removed become dudded out (point type set to 21) and are not removed from the data array.

The other options are:

`geometry` -- **xyz**, **rtz**, **rtp**

`ifirst,ilast,istride` -- point range to search

`xmin, ymin, zmin` -- minimums of `geometry` type coordinates

`xmax, ymax, zmax` -- maximums of `geometry` type coordinates

`xcen, ycen, zcen` -- center of removal space for `geometry`

`xscale, yscale, zscale` -- scaling factors for `geometry` limits

FORMAT:

rm / `geometry` /`ifirst,ilast,istride`/`xmin,ymin,zmin`/`xmax,ymax,zmax`/
`xcen,ycen,zcen` / [`xscale,yscale,zscale`]

EXAMPLE:

rm/xyz/`0,0,0/2.,2.,2./4.,4.,4./0.,0.,0./`

rm/rtz/`0,0,0/0.,0.,0./1.,360.,10./0.,0.,0./`

RMMAT

Removes all points of a specified material number. This command dudds out the points (sets ipt=21) but doesn't remove them from the data array. Use **edit/parts** to find the correct material numbers. To actually remove the points see the **rmpoints** command.

FORMAT:

rmmat / material number/

RMPOINTS

Removes a specified list of points (`ifirst,ilast,istride`) from a point distribution. The first format sets the point type flag [itp=**ifitpdud** (21)] to indicate that the set of points

should be removed, but does not actually remove the points. The second format, **/remove**, compresses and material-wise resequences all appropriately flagged points. The other options associated with the first format are: **nnlevels** (**nnlevels**=0, default), which specifies how many levels of nearest neighbors to be removed in addition to the flagged points, and **ioption** (**ioption**=0, default), which sets JTET values of tetrahedra next to those removed to either **mbndry** (**ioption**=0) or -(**mbndry**+|**ioption**|) when **ioption**≠0. This option is useful for imbedding another mesh inside an existing mesh, so that **ioption** can be used as an identifier for tetrahedron faces where a new mesh can be stitched. Clearly, these additional options could be used only if at least a part of the mesh exists. The third format, **/womesh**, duds all points not connected in the existing mesh (except, of course, for parent-interface points). The fourth format, **/zerotets**/volume, removes tetrahedra at or below the specified volume, changes the affected points to type **ifitpfre** (11), and sets JTETs to **mbndry** at appropriate tetrahedra. If the volume is not specified, it is computed internally by the code to be a small value relative to the problem. This format is useful for removing zero-volume tetrahedra at reflective boundaries. However, caution should be exercised, because the use of this option could create holes in the mesh. Also, remember to reset the point types through **setpts** or manually if you want the exposed point types to be other than type **ifitpfre** .

FORMAT:

rmpoint/ifirst,ilast,istride/nnlevels/ioption
rmpoint/remove/
rmpoint/womesh/
rmpoint/zerotets/tetrahedron_threshold_volume/

RMREGION

Removes points that lie within the specified region.

FORMAT:

rmregion/region_name/

RMSPHERE

Removes a sphere of points from a point distribution.

FORMAT:

rmsphere/inner_radius/outer_radius/xcen,ycen,zcen/

RMSURF

Removes points that lie in, on or in and on the specified surface. **loper** can be one of the following:

lt - only points in the surface are removed

eq - only points on the surface are removed

le - all points in or on the surface are removed

FORMAT:

rmsurf/region_name/**loper**

ROTATELN

Rotates a point distribution (specified by ifirst,ilast,istride) about a line. The **copy** option allows the user to make a copy of the original points as well as the rotated points, while **nocopy** just keeps the rotated points themselves. The line of rotation defined by X1 through Z2 needs to be defined such that the endpoints extend beyond the point distribution being rotated. theta (in degrees) is the angle of rotation whose positive direction is determined by the right-hand-rule, that is, if the thumb of your right hand points in the direction of the line (1 to 2), then your fingers will curl in the direction of rotation. xcen,ycen,zcen is the point where the line can be shifted to before rotation takes place.

FORMAT:

rotateln /ifirst,ilast,istride/ **[no] copy** / x1,y1,z1/x2,y2,z2/theta/
xcen,ycen,zcen/

ROTATEPT

Rotates a point distribution (defined by ifirst,ilast,istride) about a point xcen,ycen,zcen. phi (in degrees) is the angle of rotation of the XY plane around the Z-axis, where positive phi is measured from the positive x-axis toward the positive y-axis. theta (in degrees) is the angle of rotation toward the negative z-axis. The (**no**) **copy** options are as described in the **rotateln** command.

FORMAT:

rotatept /ifirst,ilast,istride/ **[no] copy** / xcen,ycen,zcen/theta/phi

RZ

This command adds points to the mesh. It can distribute points evenly or according to a ratio zoning method.

xyz specifies Cartesian coordinates.
rtz specifies cylindrical coordinates.
rtp specifies spherical coordinates.
When using the **rtz** or **rtp** coordinate systems the center is at (0,0,0). Use a **trans** command to move the center. For the **rtz** command, minimun and maximun coordinates are the triplets: radius from the cylinder's axis, angle in the xy-plane measured from the x-axis and height along the z-axis. For the **rtp** command minimun and maximun coordinates are the triplets: radius from the cylinder's axis, angle in the zy-plane measured from the positive z-axis and the angle in the xy-plane measured from the positive x-axis (see III.a.11). Note that the **rtz** always results in a (partial) cylinder of points centered around the z axis. Use the **rotateln** command to orient the cylinder. For example, to center the cylinder around the y axis, specify the x axis as the line of rotation in the **rotateln** command.
ni,nj,nk number of points to be created in each direction.
xmin,ymin,zmin minimums for coordinates.
xmax,ymax,zmax maximums for coordinates.
iiz,ijz,ikz if =0 then mins and maxs are used as cell centers
if =1 then mins and maxs are used as cell vertices
iirat,ijrat,ikrat ratio zoning switches (0=off,1=on)
xrz,yrz,zrz ratio zoning value - distance is multiplied by this value for each subsequent point.

FORMAT:
rz/xyz|rtz|rtp/ni,nj,nk/xmin,ymin,zmin/xmax,ymax,zmax/
iiz,ijz,ikz/[iirat,ijrat,ikrat/xrz,yrz,zrz/]
EXAMPLES:
rz/xyz/5,3,10/0.,2.,0./1.,.6,2./1,1,1/
This results in a set of 150 points, five across from x=0. to x=5., 3 deep from y=2. to y=6. and 10 high from z=0. to z=2.
rz/rtz/4,6,11/0.,0.,0./3.,360.,10./1,0,1/
This results in 264 points arranged around the z- axis. There are 3 rings of points at distances r=1., r=2. and r=3. from the z-axis. There are 11 sets of these three rings of points and heights z=0., z=1., z=2.,...,z=10. In each ring there are 6 points where each pair of points is separated by 60°; note that ijz=0 requests that points be placed at cell centers, hence the first point will be at 30° not at 0°. There will be 6 points identical points

at 11 intervals along the z-axis at heights z=0., z=1., z=2.,...z=10. **Filter** should be used to remove these duplicate points.

RZBRICK

Builds a brick mesh and generates a nearest neighbor connectivity matrix. This command is similar to the **rz** command format except here we have symmetry flags to input. A second format specifies that a mesh be created and connected.

xyz specifies Cartesian coordinates.

rtz specifies cylindrical coordinates.

rtp specifies spherical coordinates.

`ni,nj,nk` number of points to be created in each direction.

`xmin,ymin,zmin` minimums for coordinates.

`xmax,ymax,zmax` maximums for coordinates.

`iiz,ijz,ikz` if =0 then mins and maxs are used as cell centers
if =1 then mins and maxs are used as cell vertices

`iirat,ijrat,ikrat` ratio zoning switches (0=off,1=on)

`xrz,yrz,zrz` ratio zoning value - distance is multiplied by the value for each subsequent point.

`icount` starting point number

`isym,jsym,ksym`

FORMAT:

rzbrick/xyz|rtz|rtp/`ni,nj,nk/xmin,ymin,zmin/xmax,ymax,zmax/`
`iiz,ijz,ikz/[iirat,ijrat,ikrat/xrz,yrz,zrz/isym,jsym,ksym]`
or
rzbrick/xyz|rtz|rtp/`ni,nj,nk/icount/`**connect/**

RZS

Builds a sphere by generating coordinates of points and also modifies zoning by ratio-zoning point distributions. See the **rz** command for more details. The `itype` flag defines what type of sphere will be generated.

`itype`=1 generates a sphere by gridding the faces of a cube and then projecting the vertex onto a sphere.

`itype`=2 generates a sphere by subdividing an icosahedron placed on the surface of a sphere

`nr` is the number of radii

npt is the number of points total in the sphere

xirad, xorad are the inner and outer radii of the sphere

xcen, ycen, zcen are the coordinates of the center of the sphere

iz if =0 then mins and maxs are used as cell centers
if =1 then mins and maxs are used as cell vertices

irat is ratio zoning switch (0=off,1=on)

rz is ratio zoning value - distance is multiplied by the value for each subsequent point.

FORMAT:

rzs/itype/nr,npt,xirad,xorad/xcen,ycen,zcen/iz/irat,rz/

SCALE

Scale a point distribution specified by ifirst,ilast,istride according to the scale factors iscale, jscale, and kscale. The letters i, j, and k in the scale factors correspond to coordinates specified by one of the geometry types [**xyz** (Cartesian), **rtz** (cylindrical), **rtp** (spherical)]. For example, if geometry = rtz then iscale = rscale, jscale = tscale, and kscale = zscale. If the scaling option is **relative** then the scaling factors are unitless multipliers with reference to some geometric center (xcen,ycen,zcen). If the scaling option is **absolute** then the scaling factors are consistent units added on to the existing coordinates

FORMAT:

scale/ifirst,ilast,istride/**absolute|relative**/**xyz|rtz|rtp**/
iscale,jscale,kscale/xcen,ycen,zcen

SEARCH

This is the main command for generating the connectivity list.

isrchopt -

0 => Set up the mesh for specified points. If points are not specified, set up the mesh for the entire problem. Also, remove the enclosing tetrahedron after generating the mesh.

1 => Same as 0 except do not remove tetrahedra associated with the enclosing tetrahedron.

2 => Add specified points to the existing mesh and remove tetrahedra associated with the enclosing tetrahedron.

3 => Add specified points to the existing mesh and do not remove tetrahedra associated with the enclosing tetrahedron.

4 => Just remove tetrahedra associated with the enclosing tetrahedron.

FORMAT:

search/isrchopt/ifirst,ilast,istride/

SETPTS

Sets point types and material regions by calling **surfset** and **regset** routines. Generate constraint table.

FORMAT:

setpts

SETTETS

Set tetrahedra color (material type). Mark interface points; create child points at interior boundaries. Points on interior

If there are no parameters **settets** sets the color of all tets based on previous mregion specification

If there are parameters, tets whose face centroids all lie within the box specified by points 1 and 2 are colored to `color`. `Color` often represents material type.

FORMAT:

settets

settets/color/x1,y1,z1/x2,y2,z2/

SMOOTH

The SMOOTH Command smoothes 2D or 3D mesh objects. Adaptive smoothing (to values of specified fields) or non-adaptive smoothing is available. In the first form, we adapt the current mesh object to the specified field of the reference mesh object (`cmo_ref`). Although the x-y-z values of -cmo- are altered by adaption, `cmo_ref` should never change. Hence, to accomplish adaption using one or more fields in the current mesh object itself, one should let `cmo_ref` be a copy of the current mesh object. The user can specify one of two algorithm choices: Minimum Error Gradient Adaption (**mega**), or Elliptic Smoothing for Unstructured Grids (**esug**) The results of adaption of the grid to the field can be altered by using one or more **field** commands beforehand to modify the field of `cmo_ref` . For example, by increasing the scale of a field using **field/scale**, the **esug** algorithm option of **smooth** will produce grids with increased numbers of nodes in the regions where the field experiences relatively large gradients. By volume averaging a

field using **field/volavg, smooth** will cause a more gentle form of adaption with a better grading of elements. By composing the values of the field with **log** or **asinh** using **field/compose**, one can cause **smooth** to shift nodes to where the logarithm (or hyperbolic arcsine) of the field has interesting features, rather than where the field itself has interesting features. In the second form of the **smooth** command, we perform non-adaptive smoothing on the specified point set, using either **mega or esug**. You can specify an optional `control` value between zero and one. The default (`control`=0.) results in the standard smoothing scheme. Increasing `control towards` 1. causes the scheme to be progressively more controlled (moving the mesh less), until at control=1, there is no mesh movement whatsoever. Currently implemented is smoothing for 2D mesh objects only.

FORMAT:

smooth /mega|esug/`ifirst,ilast,istride/cmo_ref/field/`

smooth /mega|esug/`ifirst,ilast,istride/control/`

SURFACE

Defines a boundary surface of the type specified in `ibtype`.

`ibtype` can be **free**, **intrface**, or **reflect**. Use **reflect** or **free** for external boundaries, **intrface** for interior interfaces.

The surface is defined by `istype` and `X1` through `Z4`.

`istype` can be **plane**, **box, parallel**(piped), **sphere**, **cylinder**, **cone**, **ellipse**(oid), **tabular** (rotated tabular profile), or **sheet**.

`X1` through `Z4` are specified with the surface type in mind.

`isurname` is the name of the surface and must be unique for each surface defined by **surface**.

FORMAT:

surface/`isurname/ibtype/istype`**/x1/y1/z1/x2/y2/z2/x3/y3/z3/x4/y4/z4/**

EXAMPLES:

surface/`isurname/ibtype`**/box/**`xmin,ymin,zmin/xmax,ymax,zmax/`

surface/`isurname/ibtype`**/cone/**`x1,y1,z1/x2,y2,z2/radius/`

Where point 1 is the vertex and point 2 is the top center of the cone with radius from that point. A cone is finite but open. To create a closed cone cap the open end with a plane.

surface/`isurname/ibtype`**/cylinder/**`x1,y1,z1/x2,y2,z2/radius/`

Where point 1 is the bottom center and point 2 is the top center of the cylinder. Cylinders are open but finite To create a closed cylinder cap both ends with planes.

surface/isurname/ibtype/**ellipse**/x1,y1,z1/x2,y2,z2/x3,y3,z3/ar,br,cr/

Where point 1 is the center of the ellipsoid and point 2 is on the a semi-axis (new x), point 3 is on the b semi-axis (new y), and ar, br, cr are radii on their respective semi-axes.

surface/isurname/ibtype/**parallel**/x1,y1,z1/x2,y2,z2/x3,y3,z3/x4,y4,z4/

Where points 1, 2, 3 are the front left, front right and back left points of the base and point 4 is the upper left point of the front face.

surface/isurname/ibtype/**plane**/x1,y1,z1/x2,y2,z2/x3,y3,z3

surface/isurname/ibtype/**planexyz**/x1,y1,z1/x2,y2,z2/x3,y3,z3

the direction of the normal to the plane is determined by the order of the points according to the right hand rule.

surface/isurname/ibtype/**planertz**/radius1,theta1,z1,
radius2,theta2,z2, radius3,theta3,z3, xcen,ycen

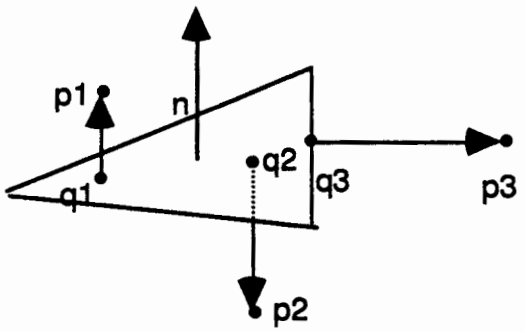
surface/isurname/ibtype/**planertp**/radius1,theta1,phi1,
radius2,theta2,phi2, radius3,theta3,phi3, xcen,ycen,zcen/

surface/isurname/ibtype/**sheet**/cmo_name/

Sheet surfaces may be input by specifying a cmo_name. The Mesh Object must be either a 2D quad Mesh Object or a 2D triangle Mesh Object.

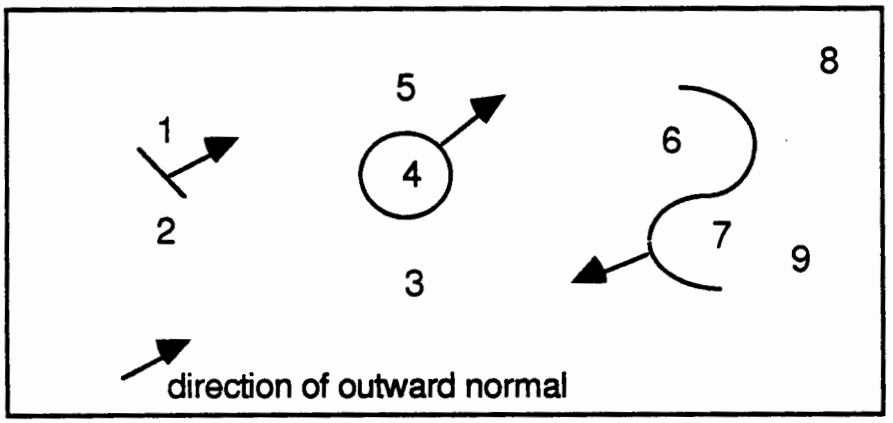
Inside/outside with respect to sheet surfaces will be determined by the following algorithm:

- For the point being considered, p, find the nearest sheet triangle and the closest point, q, to p that lies on that triangle.
- Construct the vector, $\vec{d}$, from q to p.
- Construct the outward normal to the triangle, $\vec{n}$. The outward normal is constructed using the right hand rule and the order of the points in the sheet. Sheets may be specified as quad Mesh Object (i.e. a 2 dimensional array of points containing the coordinates of the corners of each quad). Either two triangles (divide each quad in two using point (i,j) and (i+1,j+1)) or four triangles (add a point in the center of the quad) are generated by each quad. Applying the right hand rule to the points (i,j), (i+1,j), (i+1,j+1) gives the direction of the normal for all triangles created from the quad.
- If $\vec{d} \cdot \vec{n} < 0$ then the point is inside. If $\vec{d} \cdot \vec{n} > 0$ the point is outside. If $\vec{d}\,\vec{n} \cdot n = 0$, and if p is on the triangle then p=q and p in on the triangle.
- If $\vec{d} \cdot \vec{n} = 0$ and p is not on the triangle then p is outside.



p1 is outside
p2 is inside
p3 is outside

One implication of this definition is that the concept of shadows cast by open sheets no longer is valid. Sheets may be considered to extend to the boundary of the geometry.



points 1, 5, 3, 6 are outside
points 2, 4, 7, 8 and 9 are inside

surface/isurname/ibtype/**sphere**/xcen,ycen,zcen/radius/

surface/isurname/ibtype/**tabular**/x1,y1,z1/x2,y2,z2/geom/

r1,z1
r2,z2
r3,z3
....
rn,zn

```
end
```
or
```
r1,theta1
r2,theta2
r3,theta3
...
rn,thetan
end
```
Where point 1 and point 2 define the axis of rotation for the tabular profile with point 1 as the origin. This is followed by pairs of profile descriptors depending on the value of `geom`. If `geom` is set to **rz**, then the r value is a radius normal to the axis of rotation and z is the distance along the new axis of rotation. If geom is set to **rt** then `theta` is the angle from the axis of rotation at point 1 and `r` is the distance from point 1 along `theta`. The first pair must start on a new line and all lines must contain pairs of data. The last pair of data must be followed by end.

SURFPTS

Generates points on boundary surfaces previously defined by the surface or region command. The variable `itype` can be **surface** or **region** and `iname` is the name of the surface or region. The points are generated by shooting rays through a user specified set of points from an origin point, line or plane and finding the surface intersection of each ray. The point location for a region is determined by `iregpt` and can be on the **inside**, **outside** or **both** surfaces.

FORMAT:

surfpts/`itype/iname/iregpt/ifirst,ilast,istride/geom/ray_origin`
surfpts/`itype/iname/iregpt/`**pset,get,setname**/`geom/ray_origin`
Where `ifirst,ilast,istride` or **pset,get,setname** define the set of points to shoot rays through.
SPECIFICALLY FOR ALLOWABLE GEOMETRY TYPES:
surfpts/`itype/iname/iregpt/ifirst,ilast,istride/`
xyz/x1,y1,z1/x2,y2,z2/x3,y3,z3/
Where points 1, 2, 3 define the plane to shoot rays from that are normal to the plane.
surfpts/`itype/iname/iregpt/ifirst,ilast,istride/` **rtz**/x1,y1,z1/x2,y2,z2 /
Where points 1, 2, define the line to shoot rays from that are perpendicular to the line.

surfpts/itype/iname/iregpt/ifirst,ilast,istride/**rtp/xcen,ycen,zcen/**
surfpts/itype/iname/iregpt/ifirst,ilast,istride/**points/iffirst,iflast,ifstride/**
Where ifirst,ilast,istride define a set of points to shoot rays from.

TRANS

Translates a selected set of points (ifirst,ilast,istride) in X,Y,Z space by picking one specific point (xold,yold,zold) in the set of points and moving it to new coordinates (xnew,ynew,znew) with a linear translation. This will then cause the remaining points in the set to be moved by the same translation.

FORMAT:

trans/ifirst,ilast,istride/xold,yold,zold/xnew,ynew,znew

EXAMPLE:

trans/pset,get,mypoints/0.,0.,0./2.0,2.0,0./

The points in the **pset** mypoints will be moved 2 in the positive x direction and 2 in the positive y direction.

ZQ

Set or print node attribute values of a selected set of nodes.
For printing, attributes are grouped as follows:
Group1: isq,imt,itp (material type and point types)
Group2: x,y,z (coordinates)
To print, specify any one of a group and all will be printed.
To set an attribute value, set value and all selected nodes will be set to this value.

FORMAT:

zq/iattribute/ifirst,ilast,istride/value

EXAMPLE:

zq/imt/1,100,2/material1/

IV. Interfacing User Routines to X3D

a. Building an executable and running X3D.

The executable is built by linking a driver routine with the code and utility libraries. The driver routine must contain a call to `initx3d` and a call to `dotaskx3d` and must contain a subroutine called `user_sub`. A sample driver routine is listed:

```
      program adrivgen
C
C
#####################################################################
C
C     PURPOSE -X3D driver
C
#####################################################################
C
      implicit real*8 (a-h,o-z)
C
      call initx3d('generate')
C
      call dotaskx3d('interact',ierror_return)
C
      stop
      end
C
      subroutine user_sub(imsgin,xmsgin,cmsgin,msgtyp,nwds,
     x              ierr1)
C
#####################################################################
C
C     PURPOSE -
C
C        Process user supplied commands
C
C     INPUT ARGUMENTS -
C
C        imsgin - integer array of tokens returned by parser
C        xmsgin - real array of tokens returned by parser
C        cmsgin - character array of tokens returned by parser
C        msgtyp - int array of token types returned by parser
C
C     OUTPUT ARGUMENTS -
C
C        ierr1 - 0 for successful completion - -1 otherwise
C
#####################################################################
      character*32 cmsgin(nwds)
      integer imsgin(nwds),msgtyp(nwds)
      integer nwds,ierr1,lenc
      real*8 xmsgin(nwds)
C  get command length
```

```
      lenc=icharlnf(cmsgin(1))
C  Insert code here to handle user coded subroutines
C  For example
C   if(cmsgin(1)(1:lenc).eq.'my_cmnd') call my_rtn(imsgin,xmsgin
C         cmsgin,msgtyp,nwds,ierr1)
C
      ierr1=-1
      return
      end
```

Sample build scripts for the supported platform are:

Sun OS and Sun Solaris

```
f77 -g -o x3dgen adrivgen.f libx3d.a libutil.a
```

IBM RISC

```
f77 -g -o x3dgen -qintlog -brename:.fdate,.fdate_ adrivgen.f
 libx3d.a libutil.a
```

SGI

```
f77 -g -Nn10000 -o x3dgen adrivgen.f  libx3d.a libutil.a
```

HP

```
f77 -g +U77 -R8 -o x3dgen adrivgen.f libx3d.a libutil.a
```

Once the executable is built, the dictionary file must be installed. This file, `x3ddict`, is supplied with the libraries. It must either exist in the directory from which X3D will be run, or an environment variable may be set to give the directory path to its location. The format of the `setenv` command is:

```
setenv x3ddict full_directory_path_to_x3ddict.
```

To execute, use standard unix file redirection for standard input and output. X3D will produce two additional files, `outx3dgen` and `logx3dgen`. These contain detailed output information and the list of commands respectively. X3D may also be run interactively in which case the user will be prompted to enter commands from the workstation.

b. Issuing Commands from a user program.

Any X3D command can be issued by calling the subroutine `dotaskx3d`, for example:

```
call dotaskx3d('cmo/select/3dmesh', ier1)
```

will select the Mesh Object named *3dmesh.* ier1 will be zero if the commands are executed with no error, non-zero otherwise.

By using the X3D command **Infile**, a series of commands may be executed, for example

```
call dotaskx3d('infile/mydeck', ier1)
```

will execute all the X3D commands that are in the user's file named *mydeck*. The final command in the file *mydeck* should be **finish.**

c. Writing user commands

The access to user written subroutines is through the X3D subroutine, `user_sub`. It is passed the parsed command input line. The parser breaks up the input line into tokens and returns to X3D a count of number of tokens, an array containing the token types, and the tokens themselves. The parameters returned by the parser are:

`nwds` (number of tokens)
`msgtyp` (integer array of token types - 1 for integer, 2 for real, 3 for character)
`imsgin` (array of integer tokens, e.g. imsgin(i) is the ith token which is an integer if msgtyp(i)=1)
`xmsgin` (array of real tokens)
`cmsgin` (array of character tokens)

The parser is written in C, therefore character variables returned will be null terminated on some platforms. A FORTRAN function is supplied; `icharlnf` will return the length of the character string blank or null terminated, ignoring leading blanks

If the user has written a subroutine, `my_routine`, that responds to the command, `my_comnd`, the call from `user_sub` should look like:

```
      elseif (cmsgin(1)(1:lenc).eq. 'my_comnd')
x          call my_routine(nwds,imsgin,xmsgin,cmsgin,msgtyp.ierr1)
```

The subroutine `my_routine` should set `ierr1` to zero if the command is processed successfully and should use the `cmo` interface routines to access the components of the `Mesh Object` that it needs, for example:

```
      character*32 cmo
      pointer (ipimt1, imt1(*))
c        get the name of the current mesh object
      call cmo_get_name(cmo_name,ierror)
c        get the number of nodes and the material ids
      call cmo_get_info('nnodes',cmo_name,nnodes,ilen,ityp,ierr)
      call cmo_get_info('imt1',cmo_name,ipimt1,ilen,ityp,ierr)
```

The subroutine `user_sub` is supplied with the driver and is defaulted to print the error message: 'Illegal command' and return.

d. The following template is an example of using the an existing mesh object and of creating a new mesh object. The existing mesh object is a 3d object. The object to be created is a 2d object. It is first necessary to set up the pointer statements for both the existing and new mesh objects.

```
C   Definitions for incoming (existing) cmo
C
      pointer (ipimt1, imt1)
      pointer (ipitp1, itp1)
      pointer (ipint1, int1)
      pointer (ipicr1, icr1)
      pointer (ipisn1, isn1)
      pointer (ipicn1, icn1)
      integer imt1(1000000), itp1(1000000), int1(1000000),
     *        icr1(1000000), isn1(1000000), icn1(1000000)
      pointer (ipxic, xic)
      pointer (ipyic, yic)
      pointer (ipzic, zic)
      dimension xic(1000000), yic(1000000), zic(1000000)
      pointer (ipitetclr, itetclr)
      pointer (ipitettyp, itettyp)
      pointer (ipitetoff, itetoff)
      pointer (ipjtetoff, jtetoff)
      pointer (ipitet, itet)
      pointer (ipjtet, jtet)
      integer itetclr(1000000), itettyp(1000000),
     *        itetoff(1000000), jtetoff(1000000)
      integer itet(4,1000000) , jtet(4,1000000)
C
C   Definitions for cmo that is to be created
C
      pointer (ipimt1a, imt1a)
      pointer (ipitp1a, itp1a)
      pointer (ipint1a, int1a)
      pointer (ipicr1a, icr1a)
      pointer (ipisn1a, isn1a)
      pointer (ipicn1a, icn1a)
      integer imt1a(1000000), itp1a(1000000), int1a(1000000),
     *        icr1a(1000000), isn1a(1000000), icn1a(1000000)
      pointer (ipxica, xica)
      pointer (ipyica, yica)
      pointer (ipzica, zica)
      dimension xica(1000000), yica(1000000), zica(1000000)
      pointer (ipitetclra, itetclra)
      pointer (ipitettypa, itettypa)
      pointer (ipitetoffa, itetoffa)
      pointer (ipjtetoffa, jtetoffa)
      pointer (ipiteta, iteta)
      pointer (ipjteta, jteta)
      integer itetclra(1000000), itettypa(1000000),
     *        itetoffa(1000000), jtetoffa(1000000)
      integer iteta(3,1000000) , jteta(3,1000000)
C
      pointer (ipjtet2, jtet2)
      integer jtet2(3,1000000)

C    Get the existing cmo - its name is in the variable cmoin
C
```

```
      call cmo_get_name(cmoin,ier)
C
C     Get the scalar mesh variables
      call cmo_get_info('nnodes',cmoin,npoints,lencm,itypcm,ier)
      call cmo_get_info('nelements',cmoin,ntets,lencm,itypcm,ier)
      call cmo_get_info('ndimensions_topo',cmoin,ndt,lencm,itypcm,ier)
      call cmo_get_info('ndimensions_geom',cmoin,ndg,lencm,itypcm,ier)
      call cmo_get_info('nodes_per_element',cmoin,npe,lencm,itypcm,ier)
      call cmo_get_info('faces_per_element',cmoin,nfpe,lencm,itypcm,ier)
      call cmo_get_info('mbndry',cmoin,mbndry,lencm,itypcm,ier)
C
C     Get pointers to the vector variables
      call cmo_get_info('ialias',cmoin,ipialias,lenialias,ictype,ier)
      call cmo_get_info('imt1',cmoin,ipimt1,lenimt1,ictype,ier)
      call cmo_get_info('itp1',cmoin,ipitp1,lenitp1,ictype,ier)
      call cmo_get_info('int1',cmoin,ipint1,lenint1,ictype,ier)
      call cmo_get_info('icr1',cmoin,ipicr1,lenicr1,ictype,ier)
      call cmo_get_info('isn1',cmoin,ipisn1,lenisn1,ictype,ier)
      call cmo_get_info('icn1',cmoin,ipicn1,lenicn1,ictype,ier)
      call cmo_get_info('xic',cmoin,ipxic,lenxic,ictype,ier)
      call cmo_get_info('yic',cmoin,ipyic,lenyic,ictype,ier)
      call cmo_get_info('zic',cmoin,ipzic,lenzic,ictype,ier)
      call cmo_get_info('itetclr',cmoin,ipitetclr,lenitetclr,ictype,ier)
      call cmo_get_info('itettyp',cmoin,ipitettyp,lenitettyp,ictype,ier)
      call cmo_get_info('itetoff',cmoin,ipitetoff,lenitetoff,ictype,ier)
      call cmo_get_info('jtetoff',cmoin,ipjtetoff,lenjtetoff,ictype,ier)
      call cmo_get_info('itet',cmoin,ipitet,lenitet,ictype,ier)
      call cmo_get_info('jtet',cmoin,ipjtet,lenjtet,icmotype,ier)
C
C     Create the new 2d cmo - call it cmoout.
C
      call cmo_exist(cmoout,ier)
C
C     ier.eq.0 means that the cmo already exists - if so release it.
C
      if(ier.eq.0) call cmo_release(cmoout,idelete)
C
C     Set active cmo to cmoout
      call cmo_set_name(cmoout,ier)
C
C     set scalar mesh variables
C
      call cmo_set_info('nnodes',cmoout,npoints,1,1,ier)
      call cmo_set_info('nelements',cmoout,ntets,1,1,ier)
C
C     the following scalars need to be set for a 2d cmo
C
      call cmo_set_info('ndimensions_topo',cmoout,2,1,1,ier)
      call cmo_set_info('ndimensions_geom',cmoout,3,1,1,ier)
      call cmo_set_info('nodes_per_element',cmoout,3,1,1,ier)
      call cmo_set_info('faces_per_element',cmoout,3,1,1,ier)
C
C     allocate memory for vector variables
      call cmo_newlen(cmoout,ier)
```

```
C
C     now get the pointers to the allocated memory for the vector data
      call cmo_get_info('imt1',cmoout,ipimt1a,lenimt1a,icmotype,ier)
      call cmo_get_info('itp1',cmoout,ipitp1a,lenitp1a,icmotype,ier)
      call cmo_get_info('int1',cmoout,ipint1a,lenint1a,icmotype,ier)
      call cmo_get_info('icr1',cmoout,ipicr1a,lenicr1a,icmotype,ier)
      call cmo_get_info('isn1',cmoout,ipisn1a,lenisn1a,icmotype,ier)
      call cmo_get_info('icn1',cmoout,ipicn1a,lenicn1a,icmotype,ier)
      call cmo_get_info('xic',cmoout,ipxica,lenxica,icmotype,ier)
      call cmo_get_info('yic',cmoout,ipyica,lenyica,icmotype,ier)
      call cmo_get_info('zic',cmoout,ipzica,lenzica,icmotype,ier)
      call cmo_get_info('itetclr',cmoout,ipitetclra,lenclra,icmotype,ier)
      call cmo_get_info('itettyp',cmoout,ipitettypa,lentypa,icmotype,ier)
      call cmo_get_info('itetoff',cmoout,ipitetoffa,lenoffa,icmotype,ier)
      call cmo_get_info('jtetoff',cmoout,ipjtetoffa,lenoffa,icmotype,ier)
      call cmo_get_info('itet',cmoout,ipiteta,leniteta,icmotype,ier)
      call cmo_get_info('jtet',cmoout,ipjteta,lenjteta,icmotype,ier)
C
C     now the values for the vector components of the 2d mesh
C     object can be set.
```