SAND92-2290
Unlimited Release
Printed December 1992

Distribution
Category UC-705

GJOIN: A Program for Merging Two or More GENESIS Databases

Gregory D. Sjaardema
Solid and Structural Mechanics Department
Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

GJOIN is a two- or three-dimensional mesh combination program. GJOIN combines two or more meshes written in the GENESIS mesh database format into a single GENESIS mesh. Selected nodes in the two meshes that are closer than a specified distance can be combined The geometry of the mesh databases can be modified by scaling, offsetting, revolving, and mirroring. The combined meshes can be further modified by deleting, renaming, or combining material blocks, sideset identifications, or nodeset identifications. GJOIN is one of the mesh generation tools in the Sandia National Laboratories Engineering Analysis Code Access System (SEACAS). GJOIN is typically used with the other SEACAS mesh generation codes GEN3D, GENSHELL, GREPOS, and Aprepro.

Contents

Figures

1 Introduction

GJOIN is a two- or three-dimensional mesh combination program. GJOIN combines two or more meshes written in the GENESIS[11] mesh database format into a single GENESIS mesh. Selected nodes in the two meshes that are closer than a specified distance can be combined. The geometry of the mesh databases can be modified by scaling, offsetting, revolving, and mirroring. The combined mesh can be further modified by deleting, renaming, or combining material blocks, sideset identifications, or nodeset identifications.

1.1 Sandia Engineering Analysis Code Access System Mesh Generation Toolbox

GJOIN is one of the mesh generation tools in the Sandia National Laboratories Engineering Analysis Code Access System (SEACAS)[1]. GJOIN is typically used with the other SEACAS mesh generation codes FASTQ[6], GEN3D[2], GENSHELL[2], GREPOS[4], and Aprepro[7]. Figure 1 shows the structure of the SEACAS mesh generation toolbox. The basic premise underlying this toolbox is that complicated geometries can be generated using a set of small specialized codes.

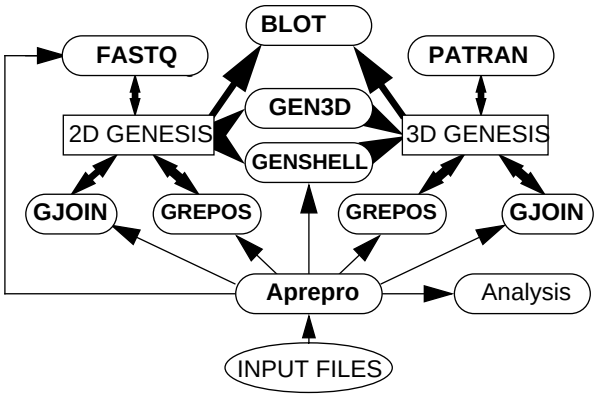

Figure 1.Structure of SEACAS Mesh Generation Toolbox

Each of these codes has a specialized purpose, a short synopsis of each code is given below, for more information consult the referenced documentation.

GEN3D — Transforms a two-dimensional GENESIS database into a three-dimensional GENESIS database. Several transformations are supported and additional transformations can be easily added.[2]

GREPOS — Transforms the geometry of a GENESIS database by scaling, mirroring, offsetting, or rotating. It can also modify the database by deleting or renaming material blocks, sideset identifications, or nodeset identifications.[4]

FASTQ	Interactive two-dimensional finite element mesh generation program. Includes several mesh generation options including paving.[6]
APREPRO	An algebraic preprocessing program which is used to parameterize finite element analyses. Includes a unit conversion system and material database access routines.[7]
GENSHELL	Transforms a two-dimensional GENESIS database into a three-dimensional shell GENESIS database. Several transformations are supported and additional transformations can be easily added.[2]
NUMBERS	Calculates several properties of an EXODUS file, including mass properties, timesteps, condition numbers, cavity volumes, and others.[9]
BLOT	The primary graphical two-dimensional and three-dimensional postprocessing code. It includes deformed mesh plots, contour plots, shaded fringe plots, variable versus variable and time history plots, and distance versus variable plots.[5]

The SEACAS mesh generation toolbox is a simpler approach to three-dimensional mesh generation than the automatic and general-purpose programs that are available from commercial vendors. Many complicated three-dimensional geometries are composed of several primitives that can be defined in terms of transformations of two-dimensional geometries. Each of the primitives can be meshed using FASTQ and GEN3D, and then joined together using GJOIN.

This approach does; however, have some inherent difficulties. The biggest being managing and synchronizing several related files. For example, the meshes form some large problems can require more than one hundred files containing FASTQ, GEN3D, GJOIN, and GREPOS input files; temporary GENESIS files; and parameter files. Manually building and modifying a mesh this complicated is obviously very difficult and time consuming. This problem has typically been minimized at SNL through the use of the UNIX* make† program and Aprepro. Make is used to build a set of dependencies between the various pieces of the finite element model. The analyst can then change files as needed and simply type `make mesh` to generate the mesh. If the dependencies have been entered correctly, make will rebuild only those portions of the mesh that are affected by the changed file. The synchronization problem (that is, ensuring that all of the separate pieces have compatible dimensions and discretizations) is typically solved by creating a few parameter files which contain key dimensions and discretization information. Aprepro is then used to preprocess the input files and insert the key dimensions and discretization information into the input files.

---

*UNIX is a registered trademark of UNIX Systems Laboratories Inc.

† See your UNIX documentation for more information on make. Typically this is done by entering the command `man 1 make`

1.2 Introduction to the GENESIS File Format

The GENESIS mesh database file format is the geometry definition portion of the EXODUS database file format used in the Engineering Sciences Center at Sandia National Laboratories. All of the mesh generation programs in the Engineering Sciences Center read and write files in the GENESIS format, which allows great flexibility in the choice of mesh generation, file translation, and graphical processing.

The GENESIS file contains the data to describe a finite element mesh including the location of the nodal points, the connectivity of the nodes that form each element, the material types of each element, and the boundary condition data which are used to specify load application points and nodal constraints. The two GENESIS entities that are primarily used in GJOIN are the Element Block ID and the Nodeset ID. These will be briefly described below; however, the reader is referred to Reference 10 for more information.

Element Block ID: In order to promote efficient storage and to enable efficient processing within codes, elements are grouped into element blocks. Within an element block, all elements must be of the same type and the same material. Each element block has an arbitrary unique number which identifies that particular block. This is called the Element Block ID.

Nodeset ID: Nodesets provide a means to reference a group of nodes with a single ID without requiring the user to know node numbers in the model. A particular node may be in multiple nodesets, but may be in a single set only once. GJOIN uses nodesets to specify which nodes are to be combined in the two meshes.

1.3 Organization of Report

The remainder of this document is organized as follows:

- Chapter 2 describes the commands recognized by GJOIN and the algorithms it uses to combine the mesh databases into a single mesh database, and
- Chapter 3 includes a short example problem which illustrates GJOIN use.

Four appendices are included in this document:

- Appendix A describes the specifics of GJOIN including executing it, obtaining it, compiling it, and quality assurance.
- Appendix B describes the original GJOIN input syntax which is still recognized, but is not recommended.
- Appendix C describes the GENESIS mesh database format.
- Appendix D presents the GJOIN output from the example problem

Intentionally Left Blank

2 Commands

GJOIN has three distinct input phases:

1. Filename prompting
2. Node combination and/or database modification prompting
3. General command processing

Initially, GJOIN prompts for the names of the two GENESIS files which are to be combined. Both files must have the same spatial dimension. GJOIN then enters the second phase in which the second database can be modified by scaling and or offsetting its coordinates and the combination of nodes in the two databases is specified. Following this, GJOIN enters the general command section in which the numbering of material blocks, sidesets, and nodesets can be modified. At this point, the resulting combined database can be saved, or additional databases can be combined with the current database. GJOIN creates a log file containing the commands entered during an interactive session. This file, typically named `gjoin.log`, can be used in subsequent invocations to run GJOIN in a batch mode.

2.1 Command Syntax

The user directs the processing by entering commands to set processing parameters.

The commands are in free-format and must adhere to the following syntax rules.

- Valid delimiters are a comma or one or more blanks.
- Either lowercase or uppercase letters are acceptable, but lowercase letters are converted to uppercase except for filenames, and database titles.
- A “$” character in any command line starts a comment. The “$” and any characters following it on the line are ignored.

Each command has an action keyword or “verb” followed by a variable number of parameters. The command verb is a character string. It may be abbreviated, as long as enough characters are given to distinguish it from other commands. The meaning and type of the parameters is dependent on the command verb. Most command parameters are optional. If an optional parameter field is blank, a command-dependent default value is supplied. Below is a description of the valid entries for parameters.

- A numeric parameter may be a real number or an integer. A real number may be in any legal FORTRAN numeric format (e.g., 1, 0.2, -1E-2). An integer parameter may be in any legal integer format.
- A string parameter is a literal character string. Most string parameters may be abbreviated.

The notation conventions used in the command descriptions are:

- The command verb is in all uppercase **bold SANSERIF** type.

- A literal string is in all uppercase SANSERIF type and should be entered as shown (or abbreviated).
- The value of a parameter is represented by the parameter name in *italics*.
- A literal string or parameter in square brackets (“[ ]”) represents a parameter option which is omitted entirely (including any following comma) if not appropriate.
- A series of literal strings separated by a vertical bar ("|") represents a list of valid options. Only one of the options is allowed. Unless the strings are in square brackets, one of the strings must be entered.
- A command description terminating with ellipses ("…") signifies that the data following the command verb can be repeated on the same command line.

2.2 Filename specification phase

Initially, GJOIN will prompt for the filenames of the first two mesh databases that are to be combined. If only one filename is specified, GJOIN will skip the node combination phase and go directly to the general command processing phase.

2.3 Node combination and database modification phase

In the node combination and database modification phase, nodes in the two files can be combined or the coordinates of the second database can be scaled, offset, and mirrored. GJOIN will prompt with the string:

```
"Combine or Convert (Enter HELP for info)> "
```

At this point, the valid responses are*:

- **COMBINE** [*nset1, nset2*] *tolerance* **[**CLOSEST**][**MATERIAL**]**
  combine all nodes in nodesets *nset1* and *nset2* that are within *tolerance* distance of each other.

  If CLOSEST is not specified, GJOIN combines the *first* node in the second database that is within *tolerance* distance of a node in the first database. If CLOSEST is specified, the closest node of all nodes in the second database that are within tolerance distance of a node in the first database is combined.

  If MATERIAL is specified, nodes are only combined if the material numbers in the first and second databases match.

  The keyword **EQUIVALENCE** can be used in place of **COMBINE**. Node combinations are performed on the modified geometry, therefore, all scaling, offsetting, and mirroring commands must be specified prior to specifying any combination command.

---

* An older syntax is also supported that requires several Y/N answers. See “Old Style Command Input Syntax” on page 23 for more information.

- **COMBINE** EXIT|END|NO
  exit the node combination phase, perform all of the specified offset, scale, and mirror commands, and combine the nodes.

- **OFFSET** X|Y|Z|ALL|RESET *offset, …*
  modify the coordinates of the second database by adding *offset* to its coordinates. Multiple offsets can be specified on a single command line; however, offsets are not cumulative. For example, if "**OFFSET X 5, Y 7.5, X 10**" is entered, the X-coordinate will be offset by 10 and the Y-coordinate by 7.5. The command **OFFSET RESET** nullifies previous offset commands. The keyword **SHIFT** is a synonym for **OFFSET**.

- **SCALE** X|Y|Z|ALL|RESET *scale,…*
  modify the coordinates of the second database by multiplying the specified coordinates by scale. Multiple scalings can be specified on a single command line; however, scalings are not cumulative. For example, if "**SCALE X 2, y 3, x 4**" is entered, the X-coordinate will be scaled by 4 and the Y-coordinate by 3. The command **SCALE RESET** nullifies previous scale commands.

- **REVOLVE** X|Y|Z|ALL *angle*,...
  modify the coordinates of the second database by revolving *angle* degrees about the specified axis. The center of rotation defaults to 0.0, 0.0, 0.0 unless set by the **REVCEN** command. Two-dimensional meshes can only be rotated about the Z axis. Multiple revolutions can be specified on a single command line and they are cumulative.

- **REVCEN** *xcen, ycen, zcen*
  set the center of rotation for the **REVOLVE** command to *xcen, ycen, zcen.* The center of rotation defaults to 0.0, 0.0, 0.0 if it is not specified.

- **MIRROR** X|Y|Z|ALL,...
  modify the coordinates of the second database by multiplying the specified coordinate by -1. Multiple mirrorings can be specified on a single command line; however they are not cumulative. The **MIRROR** command simply sets the scale factor to -1, so scaling and mirroring can not be used together. To mirror and scale at the same time, simply set the scale factor to a negative value.

- **LIST** list the nodesets that are in the first and second mesh databases.

2.3.1 Details of the Node Combination Algorithm

The GJOIN node combination algorithm is an efficient, but not overly complicated, process. The algorithm will be described by first describing the most simplistic algorithm and then adding additional refinements until the GJOIN algorithm results.

The most simplistic algorithm would be to determine the distance between every node in mesh 1 and every node in mesh 2. This would generate $num1 \times num2$ comparisons, where $num1$ and $num2$ are the number of nodes in mesh 1 and mesh 2, respectively. In addition to requiring excessive comparisons, this approach also has the disadvantage of possibly combining two or more nodes in one mesh with a single node in the other mesh.

The first refinement is to keep a list of the nodes for each mesh. Once a match has been determined, the nodes that participated in the match are removed from the list. This simple refinement will, on the average, reduce the number of comparisons by one-half. Another gain in performance can be realized by preprocessing the two lists. First the overlapping volume of the two lists is calculated, then the lists are reduced to include only the nodes that fall within the overlapping volume. This eliminates from the node lists all nodes that could not possibly match a node in the other list. For some geometries this results in a significant reduction in the list; however, for many geometries there is little savings.

The next refinement is to order the lists based on the nodal X-coordinate. The searching can then be terminated when the X-coordinate in the second list exceeds the X-coordinate in the first list by more than the tolerance. This is algorithm is implemented in GJOIN. Additional refinements could be made; however, the current algorithm has proven itself fast enough, even for meshes consisting of over 200,000 nodes. A pseudo-code representation of this algorithm is shown below:

```
generate node lists -- either all of the nodes, or if a nodeset
      match, all of the nodes in the nodesets.
determine overlapping volume of two node lists
remove nodes that do not fall within overlapping volume
sort lists on x-coordinate
---now we begin the node comparison function
jbeg = 1
for i = 1 to length(list1)
   node1 = list1(i)
   dmin = BIG_NUMBER
   for j = jbeg to length(list2)
      node2 = list2(j)
      if (x(node1) - eps > x(node2)) jbeg = j
      if (x(node1) + eps < x(node2)) --exit inner loop
      dist = distance(node1, node2)
      if (dist < tolerance) then
         if (closest match AND dist < dmin) then
            dmin = dist
            node_min = node2
         else (not closest match)
            ---Combine node1 and node2
            ---Remove node2 from list2
            ---Get a new node from list1 (goto next i)
          end if
       end if
    next j
    if (closest match AND dmin < tolerance) then
       ---Combine node1 and node_min
       ---Remove node_min from list2
    end if
 next i
```

If the by-material matching is being performed, the above process is repeated for each matching material block in the two meshes. At the end of this process, the number of combined nodes, the maximum distance between matched nodes, and the minimum distance

between non-matched nodes are summarized. If the minimum and maximum distances are relatively close, or if the number of combined nodes is less (or more) than expected, you should loosen (or tighten) the tolerance.

2.4 General Command Processing Phase

After the node combination section has been completed, GJOIN enters the general command processing mode in which several attributes of the mesh can be controlled. The prompt in this mode is: GJOIN>. Valid commands are:

TITLE	change the database title
BLOCKS	manipulate the element material blocks
MATERIAL	manipulate the element material blocks
NODESETS	manipulate the nodal point sets
NSETS	manipulate the nodal point sets
SIDESETS	manipulate the element side sets
SSETS	manipulate the element side sets
FINISH	end command input, write output file
ADD	end command input, add another mesh piece
EXIT, END	end command input, start processing
QUIT	abort processing, stop immediately
HELP	print this list

NOTE: **END** and **EXIT** are old commands that have been superseded by **ADD** and **FINISH**. See “Old Style Command Input Syntax” on page 23 for more information. Each of the above commands is described in more detail in the following sections. When **ADD** is entered, GJOIN begins again in the file name prompting mode and continues through each of the modes again. If **FINISH** is entered, GJOIN prompts for the output file name and writes the combined mesh to the specified file.

2.4.1 TITLE

TITLE enters the title manipulation routine in which you can change the title of the output database. By default, the title of the first input database is written to the output database. Valid commands in this section are:

1	copy title from first database
2	copy title from second database (if any)
CHANGE	change title to user-specified title
LIST	list database titles
UP	go up a command level (back to command mode)
EXIT	go up a command level (back to command mode)

If **CHANGE** is entered, GJOIN will prompt for the title on a separate line.

2.4.2 Blocks and Material

BLOCKS or **MATERIAL** enters the block manipulation routine in which you can modify the material blocks of the output database. Valid commands are:

ID *n newid*	change the block identification of block *n* to *id*
DELETE *id1, id2, …*	delete material blocks with identification numbers *id1*, *id2*, ….
COMBINE *id1 id2 …*	combine material blocks *id1, id2, …*, into a single material block with identification *id1*
RESET *id*	reset material block *id*
LIST	list information about the material blocks
UP/EXIT	go up a command level

2.4.3 Nodesets

NODESETS or **NSETS** enters the nodeset manipulation routine in which you can modify the nodesets of the output database. Valid commands are:

ID *n newid*	change the nodeset identification of nodeset number *n* to *newid*
DELETE *id1 id2 …*	delete nodesets with identification numbers *id1, id2, …*
COMBINE *id1 id2 …*	combine nodesets *id1*, *id2*, … into a single nodeset with identification number *id1*
RESET *id*	reset nodeset *id*
LIST	list information about the nodesets
UP/EXIT	go up a command level

2.4.4 Sidesets

SIDESETS or **SSETS** enters the sideset manipulation routine in which you can modify the sidesets of the output database. Valid commands are:

ID *n newid*	change the sideset identification of sideset number *n* to *newid*
DELETE *id1 id2 …*	delete sidesets with identification numbers *id1, id2, …*
COMBINE *id1 id2 …*	combine sidesets *id1, id2, …* into a single sideset with identification number *id1*
RESET *id*	reset sideset *id*
LIST	list information about the sidesets

UP/EXIT go up a command level

2.4.5 Finish

FINISH ends general command processing and terminates GJOIN. GJOIN will prompt for the filename for the output database, write the file, and terminate.

2.4.6 Add

ADD ends general command processing and prepares to add another piece to the combined mesh. GJOIN will prompt for the filename for the next input database and begin again at the node combination phase.

2.4.7 Exit and End

EXIT and **END** are old style commands which terminate general command processing. GJOIN will then ask if you want to add another piece or write the combined mesh.

2.4.8 Quit

QUIT abandons all processing and terminates GJOIN. No files are written.

2.4.9 Help

HELP provides a system-dependent help message.

Intentionally Left Blank

3 GJOIN Example Problem

The following example illustrates most of the commands in GJOIN. Although this geometry could easily be generated without GJOIN, it is easier to illustrate GJOIN usage with a two-dimensional mesh. In this example, we want to generate the geometry shown at the left side of Figure 2. The first step in performing the mesh generation is to decompose

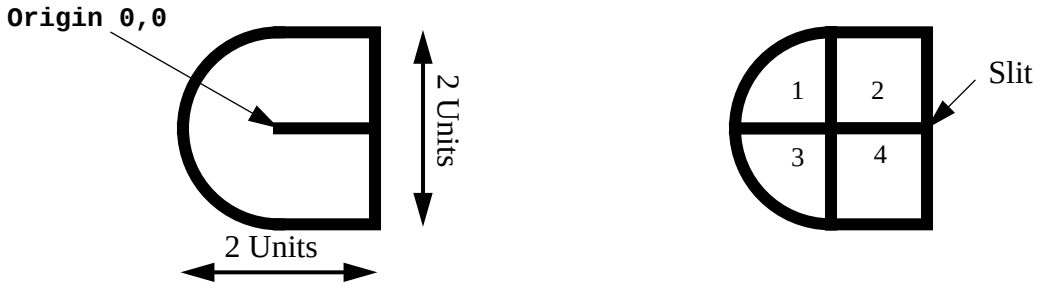

Figure 2. Geometry Definition for GJOIN Example Problem

the geometry into primitives. In this case, the entire geometry can be built with a mesh of a square and a quadrant of a circle as shown in the right side of Figure 2. The next step is to generate the primitive meshes which are schematically shown in Figure 3. Now, we must

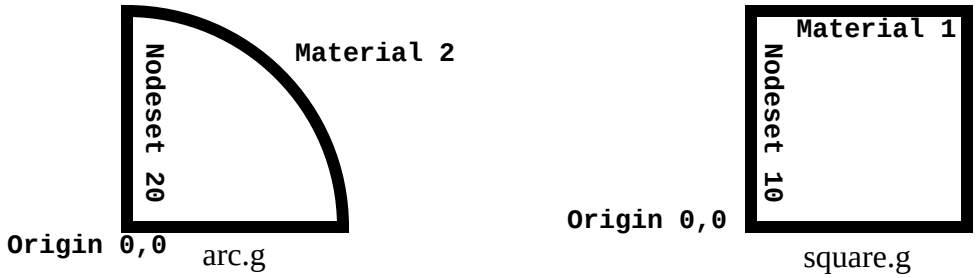

Figure 3. Schematics of Mesh Primitives for GJOIN Example Problem

determine the order to join the pieces together. For this case, and for many practical cases, there are many options. In this case we can simply join all of the pieces separately with no intermediate meshes generated; however, in practical cases it is sometimes advantageous to generate, for example, half of the mesh and then mirror it and join it to itself. For the sample problem, we could do this by first combining pieces 1 and 2, write a temporary mesh, and then reenter GJOIN and read in the temporary mesh, mirror it about the vertical axis and join it to itself.

Next, we need to create the GJOIN input file. In our case, the input file will look like:

square.g	Initial Input File
arc.g	Second Input File
mirror x	Mirror it horizontally
combine 1.0e-4	Equivalence all nodes closer than 1.0e-4
combine end	No more combination
add	Add another piece
arc.g	Filename of added piece
mirror x, y	Mirror it horizontally and vertically
combine 1.0e-4	Equivalence all nodes closer than 1.0e-4
combine end	No more combination
add	Add another piece
square.g	Filename of added piece
offset y -1.0	Offset vertically 1 unit
combine 20 10 1.0e-4	Equivalence nodes in nodeset 20 in main piece and nodeset 10 in added piece.
combine end	No more combination
nodeset	Modify the nodesets
delete 10, 20	Delete both nodesets, used only for mesh generation, not needed for analysis
up	Finished with nodeset modification
material	Modify the material blocks
combine 1 2	Combine all materials to material 1
up	Finished with material modification
title	Modify the title
change	Want a completely new title
Example Problem for GJoin	The new title
up	Finished with title modification
finish	Finished with mesh combination
example.g	Name the resultant file

This is then run by typing `gjoin < input_file.` The output from this execution is reproduced in Reference D, and the resulting mesh is shown in Figure 4.

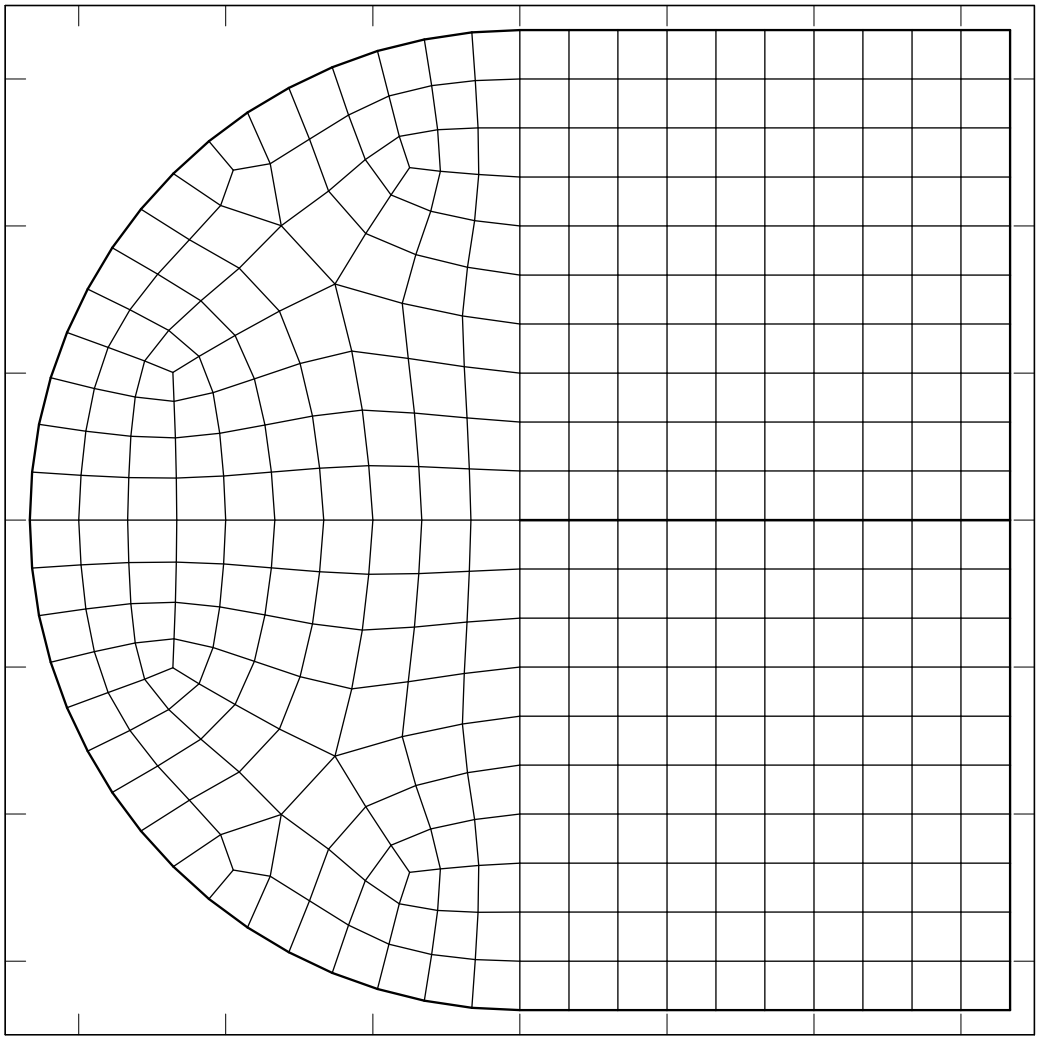

Figure 4. Mesh Resulting from GJOIN Example Problem

Intentionally Left Blank

4 References

[1]G. D. Sjaardema, “Overview of the Sandia National Laboratories Engineering Analysis Code Access System,” SAND92-2292, Sandia National Laboratories, Albuquerque, NM, January 1993.

[2]A. P. Gilkey and G. D. Sjaardema, “GEN3D: A GENESIS Database 2D to 3D Transformation Program,” SAND89-0485, Sandia National Laboratories, Albuquerque, New Mexico, March 1989.

[3]G. D. Sjaardema, “GENSHELL: A GENESIS Database 2D to 3D Shell Transformation Program,” In preparation.

[4]G. D. Sjaardema, “GREPOS: A GENESIS Database Repositioning Program,” SAND90-0566, Sandia National Laboratories, Albuquerque, NM, April 1990.

[5]A. P. Gilkey and J. H. Glick, “BLOT-A Mesh and Curve Plot Program for the Output of a Finite Element Analysis,” SAND88-1432, Sandia National Laboratories, Albuquerque, New Mexico, August 1991.

[6]T. D. Blacker, “FASTQ Users Manual, Version 2.1,” SAND88-1326, Sandia National Laboratories, Albuquerque, NM, July 1988.

[7]G. D. Sjaardema, “Aprepro: An Algebraic Preprocessor for Parameterizing Finite Element Analyses,” SAND92-2291, Sandia National Laboratories, Albuquerque, New Mexico, December1992.

[8]J. H. Red-Horse, W. C. Mills-Curran, D. P. Flanagan, “SUPES Version 2.1, A Software Utilities Package for the Engineering Sciences,” SAND90-0247, Sandia National Laboratories, Albuquerque, New Mexico, May 1990.

[9]G. D. Sjaardema, “NUMBERS: A Collection of Utilities for Pre- and Postprocessing Two- and Three-Dimensional EXODUS Finite Element Models,” SAND88-0737, Sandia National Laboratories, Albuquerque, New Mexico, March 1989.

[10]W. C. Mills-Curran, A. P. Gilkey, and D. P. Flanagan, “EXODUS: A Finite Element File Format for Pre- and Post-processing,” SAND87-2977, Sandia National Laboratories, Albuquerque, New Mexico, September 1988.

[11]L. M. Taylor and D. P. Flanagan and W. C. Mills-Curran, “The GENESIS Finite Element Mesh File Format,” SAND86-0910, Sandia National Laboratories, Albuquerque, NM, May 1986.

[12]*American National Standard Programming Language FORTRAN*, American National Standards Institute, Inc., ANSI X3.9-1978, New York, 1978.

[13]B. Berliner, “CVS II: Parallelizing Software Development,” Paper presented at the Winter 1990 USENIX Conference, Washington, D.C., 1990.

Intentionally Left Blank

A Old Style Command Input Syntax

The initial version of GJOIN was primarily a request-driven program. GJOIN would ask a question and the user would respond with **Yes** or **No**. This syntax is still recognized to maintain backward compatibility; however, its use is not recommended. The following questions would be asked:

`First input file>` *Enter the file name of the first piece*

`Next input file>` *Enter the file name of the second piece*

`Combine or Convert (Enter HELP for info)>`
Enter **Yes** *or* **No**

The following two prompts only appear if each piece has nodesets.

`Should a nodal point set match be done?`

Enter **YES** *if you want to match based on nodesets,*
Enter **NO** *if you want to match on geometry only.*

`Enter set ID of first set, second set>`
Enter nodeset ids

`Enter new value for tolerance (<ret> for default):`
Enter the tolerance

The above three lines will be repeated until **NO** *is entered*
The GJOIN> prompt will now appear and you will be in general command mode
Enter **EXIT** *to end command mode, GJOIN will then ask:*

`Is there another database?`

Enter **YES** *if you want to add another database to the generated database, the* `Next input file>` *prompt (line 2 above) will then appear and the above process will repeat.*
Enter **NO** *to end processing and write out the final database*

`Output file>` *Enter the output filename*

Intentionally Left Blank

B GJOIN Details

Execution:

To execute GJOIN on a UNIX* system (with SEACAS), type:

```
gjoin [<input_file]
```

where `input_file` is an optional input file containing commands. Commands are read from the standard input if `input_file` is not specified.

GJOIN reads and writes mesh database files written in the GENESIS format. A code segment illustrating the GENESIS database is given in Appendix C.

Source Code:

The GJOIN source code is maintained in the SEACAS system which is managed by the Concurrent Version System (cvs)[13]. GJOIN is written in ANSI standard FORTRAN-77[12]. It must be linked with the SUPES[8] and suplib libraries which are also part of SEACAS.

Availability:

GJOIN and all other SEACAS codes are available on a licensed basis. The license agreements for these codes stipulate that (1) the software is to be used solely for internal purposes, (2) the codes are not to be distributed or transferred to any person without written permission, (3) the codes are to be used at a single site and should be copied only for necessary maintenance, development, or backup purposes, and (4) there should be a procedure, or site plan, in place for protecting the provisions of the license agreements.

For more information on obtaining GJOIN or other SEACAS codes, contact:

Marilyn K. Smith
Division 1425
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-5800
(505) 844-3082, FAX: (505) 844-9297

*UNIX is a registered trademark of UNIX Systems Laboratories Inc.

Intentionally Left Blank

C The GENESIS Database Format

The following code segment reads a GENESIS database.

```
C   --Open the GENESIS database file

      NDB = 9
      OPEN (UNIT=NDB, ..., STATUS='OLD', FORM='UNFORMATTED')

C   --Read the title

      READ (NDB) TITLE
C       --TITLE - the title of the database (CHARACTER*80)

C   --Read the database sizing parameters

      READ (NDB) NUMNP, NDIM, NUMEL, NELBLK,
     &   NUMNPS, LNPSNL, NUMESS, LESSEL, LESSNL
C       --NUMNP - the number of nodes
C       --NDIM - the number of coordinates per node
C       --NUMEL - the number of elements
C       --NELBLK - the number of element blocks
C       --NUMNPS - the number of node sets
C       --LNPSNL - the length of the node sets node list
C       --NUMESS - the number of side sets
C       --LESSEL - the length of the side sets element list
C       --LESSNL - the length of the side sets node list

C   --Read the nodal coordinates

      READ (NDB) ((CORD(INP,I), INP=1,NUMNP), I=1,NDIM)

C   --Read the element order map (each element must be listed once)

      READ (NDB) (MAPEL(IEL), IEL=1,NUMEL)

C   --Read the element blocks

      DO 100 IEB = 1, NELBLK

C       --Read the sizing parameters for this element block

          READ (NDB) IDELB, NUMELB, NUMLNK, NATRIB
C           --IDELB - the element block identification (must be unique)
C           --NUMELB - the number of elements in this block
C           --   (the sum of NUMELB for all blocks must equal NUMEL)
C           --NUMLNK - the number of nodes defining the connectivity
C           --   for an element in this block
C           --NATRIB - the number of element attributes for an element
C           --   in this block

C       --Read the connectivity for all elements in this block
```

```
          READ (NDB) ((LINK(J,I), J=1,NUMLNK, I=1,NUMELB)

C       --Read the attributes for all elements in this block

          READ (NDB) ((ATRIB(J,I), J=1,NATRIB, I=1,NUMELB)

  100 CONTINUE

C    --Read the node sets

       READ (NDB) (IDNPS(I), I=1,NUMNPS)
C       --IDNPS - the ID of each node set
       READ (NDB) (NNNPS(I), I=1,NUMNPS)
C       --NNNPS - the number of nodes in each node set
       READ (NDB) (IXNNPS(I), I=1,NUMNPS)
C       --IXNNPS - the index of the first node in each node set
C       --   (in LTNNPS and FACNPS)

       READ (NDB) (LTNNPS(I), I=1,LNPSNL)
C       --LTNNPS - the nodes in all the node sets
       READ (NDB) (FACNPS(I), I=1,LNPSNL)
C       --FACNPS - the factor for each node in LTNNPS

C    --Read the side sets

       READ (NDB) (IDESS(I), I=1,NUMESS)
C       --IDESS - the ID of each side set
       READ (NDB) (NEESS(I), I=1,NUMESS)
C       --NEESS - the number of elements in each side set
       READ (NDB) (NNESS(I), I=1,NUMESS)
C       --NNESS - the number of nodes in each side set
       READ (NDB) (IXEESS(I), I=1,NUMESS)
C       --IXEESS - the index of the first element in each side set
C       --   (in LTEESS)
       READ (NDB) (IXNESS(I), I=1,NUMESS)
C       --IXNESS - the index of the first node in each side set
C       --   (in LTNESS and FACESS)

       READ (NDB) (LTEESS(I), I=1,LESSEL)
C       --LTEESS - the elements in all the side sets
       READ (NDB) (LTNESS(I), I=1,LESSNL)
C       --LTNESS - the nodes in all the side sets
       READ (NDB) (FACESS(I), I=1,LESSNL)
C       --FACESS - the factor for each node in LTNESS

C ...A valid GENESIS database may end at this point or after any point
C ...described below.

C    --Read the QA header information

       READ (NDB, END=...) NQAREC
C       --NQAREC - the number of QA records (must be at least 1)

       DO 110 IQA = 1, MAX(1,NQAREC)
```

```
         READ (NDB) (QATITL(I,IQA), I=1,4)
C         --QATITL - the QA title records; each record contains:
C         --   1) analysis code name (CHARACTER*8)
C         --   2) analysis code qa descriptor (CHARACTER*8)
C         --   3) analysis date (CHARACTER*8)
C         --   4) analysis time (CHARACTER*8)
  110 CONTINUE

C   --Read the optional header text

      READ (NDB, END=...) NINFO
C      --NINFO - the number of information records

      DO 120 I = 1, NINFO
         READ (NDB) INFO(I)
C         --INFO - extra information records (optional) that contain
C         --   any supportive documentation that the analysis code
C         --   developer wishes (CHARACTER*80)
  120 CONTINUE

C   --Read the coordinate names

      READ (NDB, END=...) (NAMECO(I), I=1,NDIM)
C      --NAMECO - the coordinate names (CHARACTER*8)

C   --Read the element type names

      READ (NDB, END=...) (NAMELB(I), I=1,NELBLK)
C      --NAMELB - the element type names (CHARACTER*8)
```

Intentionally Left Blank

D GJOIN Example Problem Output

```
                    *** GJoin Test Version 1.9 ***
                          Revised 92/11/11

                A GENESIS DATABASE COMBINATION PROGRAM

                     Run on 11/11/92 at 15:48:19


Database:  square.g

Square mesh for gjoin example

Number of coordinates per node       =     2
Number of nodes                      =   121
Number of elements                   =   100
Number of element blocks             =     1

Number of node sets                  =     1
   Length of node list               =    11
Number of side sets                  =     0


Database:  arc.g

Arc mesh for gjoin example

Number of coordinates per node       =     2
Number of nodes                      =   101
Number of elements                   =    82
Number of element blocks             =     1

Number of node sets                  =     1
   Length of node list               =    11
Number of side sets                  =     0


*** Xnew = -1.000E+00 * Xold + 0.000E+00
*** Entering Sorting Phase
*** Entering Comparison Phase


Number of equivalence comparisons =                  66
Tolerance used for matching =                  1.000E-04
Maximum distance between matched nodes    =    0.000E+00
Minimum distance between nonmatched nodes =    1.000E-01

   11 nodes matched
Cpu Time =  0.000E+00, comparison/sec = Infinite
```

```
Database:  %gjoin

Square mesh for gjoin example

Number of coordinates per node       =     2
Number of nodes                      =   211
Number of elements                   =   182
Number of element blocks             =     2

Number of node sets                  =     2
   Length of node list               =    22
Number of side sets                  =     0


Database:  arc.g

Arc mesh for gjoin example

Number of coordinates per node       =     2
Number of nodes                      =   101
Number of elements                   =    82
Number of element blocks             =     1

Number of node sets                  =     1
   Length of node list               =    11
Number of side sets                  =     0


*** Xnew = -1.000E+00 * Xold + 0.000E+00
*** Ynew = -1.000E+00 * Yold + 0.000E+00
*** Entering Sorting Phase
*** Entering Comparison Phase


Number of equivalence comparisons =                   66
Tolerance used for matching =                  1.000E-04
Maximum distance between matched nodes    =    0.000E+00

   11 nodes matched
Cpu Time =  0.000E+00, comparison/sec = Infinite


*** WARNING - Duplicate IDs in element blocks - combined unless changed
*** WARNING - Duplicate IDs in nodal point sets -
combined unless changed


Database:  %gjoin

Square mesh for gjoin example

Number of coordinates per node       =     2
```

```
Number of nodes                      =   301
Number of elements                   =   264
Number of element blocks             =     2

Number of node sets                  =     2
   Length of node list               =    32
Number of side sets                  =     0


Database:  square.g

Square mesh for gjoin example

Number of coordinates per node       =     2
Number of nodes                      =   121
Number of elements                   =   100
Number of element blocks             =     1

Number of node sets                  =     1
   Length of node list               =    11
Number of side sets                  =     0


Nodal point sets:
   Set    10 (#1):    11 nodes
   Set    20 (#2):    21 nodes
 * Set    10 (#3):    11 nodes
*** Ynew = 1.000E+00 * Yold + -1.000E+00
*** Entering Sorting Phase
*** Entering Comparison Phase


Number of equivalence comparisons =                  176
Tolerance used for matching =                    1.000E-04
Maximum distance between matched nodes    =      2.086E-07
Minimum distance between nonmatched nodes =      1.000E-01

   11 nodes matched
Cpu Time =  0.000E+00, comparison/sec = Infinite


*** WARNING - Duplicate IDs in element blocks - combined unless changed
*** WARNING - Duplicate IDs in nodal point sets -
combined unless changed


Nodal point sets:
   Set    10 (#1):    11 nodes   \    combined into ID 10
 * Set    10 (#3):    11 nodes   /    combined into ID 10
   Set    20 (#2):    21 nodes


Nodal point sets:
   Set    10 (#1):    11 nodes   <deleted>
```

```
   Set    20 (#2):    21 nodes   <deleted>
 * Set    10 (#3):    11 nodes   <deleted>


Element blocks:
   Block     1 (#1):   100 elements   4-node   \    combined into ID 1
 * Block     1 (#3):   100 elements   4-node   /    combined into ID 1
   Block     2 (#2):   164 elements   4-node


Element blocks:
   Block     1 (#1):   100 elements   4-node   \    combined into ID 1
   Block     2 (#2):   164 elements   4-node    |   combined into ID 1
 * Block     1 (#3):   100 elements   4-node   /    combined into ID 1


Database title:
Square mesh for gjoin example
Output database title:
Square mesh for gjoin example


Database title:
Square mesh for gjoin example
Output database title:
Example Problem for GJoin


Database:  example.g

Example Problem for GJoin

Number of coordinates per node       =     2
Number of nodes                      =   411
Number of elements                   =   364
Number of element blocks             =     1

Number of node sets                  =     0
Number of side sets                  =     0

GJoin used .27 seconds of CPU time
```

Distribution

1	1400	E. J. Barsis
1	1401	J. R. Asay
1	1402	S. S. Dosanjh
1	1403	G. S. Davidson
1	1404	J. A. Ang
13	1425	J. H. Biffle & staff
50	1425	M. K. Smith
1	1431	J. M. McGlaun
1	1431	K. G. Budge
1	1431	J. S. Peery
1	1432	W. T. Brown
1	1433	J. W. Swegle
15	1434	D. R. Martinez & staff
1	1500	D. J. McCloskey
1	1501	C. W. Peterson
1	1502	P. J. Hommert
1	1503	L. W. Davison
1	1504	D. J. McCloskey, actg
1	1511	J. S. Rottler
1	1511	D. K. Gartling
1	1511	M. W. Glass
1	1511	P. L. Hopkins
1	1511	M. J. Martinez
1	1511	P. A. Sackinger
1	1511	P. R. Schunk
1	1511	J. D. Zepper
1	1512	A. C. Ratzel
1	1513	R. D. Skocypec
1	1513	R. G. Baca
1	1513	B. L. Bainbridge
1	1513	R. E. Hogan, Jr.
1	1513	J. L. Moya
1	1551	W. P. Wolfe
1	1552	C. E. Hailey
1	1553	W. L. Hermina
1	1554	W. H. Rutledge
15	1561	H. S. Morgan & staff
13	1562	R. K. Thomas & staff
10	1562	G. D. Sjaardema
1	1832	J. M. Ramage
1	2565	S. T. Montgomery
1	6313	J. Jung
1	6411	A. S. Benjamin
1	6423	J. F. Dempsey
1	6513	D. S. Oscar
1	6522	J. D. Miller
5	7141	Technical Library
1	7151	Technical Publications
10	7613-2	Document Processing for DOE/OSTI
1	8523-2	Central Technical Files
6	8741	G. A. Benedetti & staff
1	8742	M. R. Birnbaum
1	8742	J. J. Dike
1	8742	L. I. Weingarten
5	8743	M. L. Callabresi & staff