PLACING THREE-DIMENSIONAL ISOPARAMETRIC ELEMENTS INTO NASTRAN

By M. B. Newman and A. W. Filstrup
Westinghouse Research and Development Center

SUMMARY

Linear (8 node), parabolic (20 node), cubic (32 node) and mixed (some edges linear, some parabolic and some cubic) have been inserted into NASTRAN, Level 15.1. First the dummy element feature was used to check out the stiffness matrix generation routines for the linear element in NASTRAN. Then, the necessary modules of NASTRAN were modified to include the new family of elements. The matrix assembly was changed so that the stiffness matrix of each isoparametric element is only generated once as the time to generate these higher order elements tends to be much longer than the other elements in NASTRAN. This paper presents some of the experiences and difficulties of inserting a new element or family of elements into NASTRAN.

INTRODUCTION

In solving many structural problems at Westinghouse, it has become apparent that in order to obtain the accuracy required, three-dimensional finite elements would be required. It also became apparent that three-dimensional finite elements based on constant strain tetrahedra like the CTETRA, CWEDGE, CHEXA1 and CHEXA2 elements in NASTRAN are too stiff to give accurate results at a reasonable cost for many problems.

Because of this, a Westinghouse proprietary program, WISEC, was developed for heat conduction and static linear elastic analysis using three dimensional isoparametric elements. Because of the large general purpose capability of NASTRAN, both for types of problems solved and for types of elements used, it was decided to place these elements into NASTRAN.

Even though three-dimensional isoparametric elements were then to be and now have been placed into NASTRAN by Dr. E. I. Field and Mr. S. E. Johnson of Universal Analytics (see Ref. 1), and are to be included in Level 16 NASTRAN now scheduled to be released in 1974, it was decided to place three-dimensional isoparametric elements into Level 15.1 NASTRAN. First we would have use of this element in NASTRAN at an earlier date than we would if we waited until Level 16 was released. Second, we would gain experience and familiarity with NASTRAN which would enable us to more easily make any future modifications which we would desire. A third benefit, which we didn't realize at the time, is the
fact that the family of elements we added can have different number of grid or nodal points on the various edges as shown in figure 1. As we understand, Level 16 NASTRAN will have elements which are either linear (2 points on each edge), parabolic (3 points on each edge) or cubic (4 points on each edge). Mixed elements, like that shown in figure 1, can be used to reduce the number of degrees of freedom in portions of the structure not requiring the higher order elements without introducing incompatibilities between adjacent elements. The order of an element is taken to be that of its highest ordered edge.

As the theory of three-dimensional isoparametric elements is explained elsewhere, for instance in Refs. 1 to 4, it will not be repeated here.

At the present time, the stiffness and mass matrices have been successfully inserted and tested. The differential stiffness matrix is due to be added shortly.

The work described in this report was performed with Level 15.1 NASTRAN on an IBM 370-165. It is planned to insert the changes into Level 15.5 NASTRAN on a CDC 6600.

RECOMMENDATIONS

1. For anyone making changes in NASTRAN, an up-to-date Programmer's Manual is of great aid. Unfortunately, the latest available Programmer's Manual is not always for the latest available level of NASTRAN.

2. Many of the tables present in Level 15.1 NASTRAN are too short to permit elements with as many degrees of freedom as the isoparametric elements. These tables should be increased in length to permit easier insertion of new elements.

METHOD AND EXPERIENCES

NASTRAN is an extremely large system comprised of fifteen super links with approximately 850 subprograms whose source statements are on over 200,000 card images. NASTRAN is indeed a very large and complex program and, at first glance, a dense forest that seems too difficult to enter. As one starts to review the large NASTRAN Programmer's Manual (approximately 15 centimeters thick) and examine the materials the authors of NASTRAN have distributed, the forest does not seem as dense. This section of the paper describes our experiences in adding new elements to the NASTRAN system.

The three-dimensional isoparametric elements added presented many problems that the usual NASTRAN elements did not encounter. The tables were much larger, for example. The number of nodes that described our cubic isoparametric element varies from ten to thirty-two nodes. This number forced us to expand the Element Connection and Properties Tables and other array sizes that dealt with nodes. The concept of a variable number of nodes per element was also a departure from the usual NASTRAN practice of a constant number of nodes per element type.
For these higher order elements, the computer time necessary to create the element matrices was quite large; hence, a procedure to create the element matrices once and to save them had to be incorporated into the element level subprograms.

The release we used to incorporate the new elements was level 15.1. The computer used was an IBM 370-165 operating under the ASP system. The Programmer's Manual we had was for Level 12 which caused some difficulty but not too much. We will outline the procedure we used in adding the new elements.

First one should review the materials distributed with the Level 15 system. Figure 2 is a VTOC (Volume Table of Contents) of the distributed system. Pages 5.3-13 and 5.3-14 of the Programmer's Manual (ref. 5) describes each of the data sets of the distributed system. The data sets which are most useful to us are SOU1, the partitioned data set containing the FORTRAN source programs, SUBSYS, also a partitioned data set containing the linkage editor control cards for the fifteen super links of NASTRAN; the partitioned data set OBJ, which contains all the load modules of each individual subroutine of the system; the partitioned data set NSTNLMOD, which contain the fifteen link-edited super links which constitute the NASTRAN executable set.

The next step of the procedure should be to set up a development disk with at least two data sets which we named NEWOBJ and NADEV. NEWOBJ corresponds to OBJ, and NADEV corresponds to NSTNLMOD. It would be advisable to set up a data set corresponding to SOU1 but we elected to keep all of our new source programs in card-deck form. The IBM 370 utility program IEBMOVE or IEBCOPY can be used to move the fifteen link edited links from NSTNLMOD to the development pack. NADEV's initial allocation should be as large as possible as this data set will be modified frequently. An alternate approach, which we did not use but one that could have saved us some grief would be to set up fifteen different data sets rather than one partitioned data set with fifteen members. Then each time we needed to link-edit, we would scratch the particular data set and recreate the new link edited data set (instead of member). This procedure would keep us from using up all the extents of a partitioned data set and not having to compress the partitioned set which we had to do approximately every twenty to twenty-five re-link edits. Figure 3 is the VTOC of our development disk. The other utility that we made quite frequent use of was IEBPTPCH. With the use of this utility we can either list or punch a member of SOU1 or any of the other partitioned data sets. The JCL for PUNCHIT is given in Figure 4 and for PRINIT in Figure 5. With these two decks we can list or punch subroutines from SOU1. The punched routine could then be modified for our new element. Another utility which could have been used for modifying source decks is IEBUPDATE which we did not use. The next step in the process is to compile either a modified subprogram of the NASTRAN system or to compile one of our new subprograms. The compiled program is placed into our partitioned data set NEWOBJ. The JCL for this procedure is shown in Figure 6. When all the decks for one of the links has been compiled, the next step is to link edit this link.

The link editor allows one to specify a group of libraries of programs via the LKED.xxx DD cards. In our case, we described two libraries, OBJ which
contained all the original unmodified or distributed load modules and NEWOBJ, the modified and new load modules. Each library is given a DD name, for the partitioned set OBJ the name LKED.LIB is used and for NEWOBJ we chose to use the name LKED.LIP. The overlay control cards can be punched and listed from the data set SUBSYS for this link. The control card deck is then modified to reflect the modifications made to the link. See Figure 7 for the JCL and modified control deck for the Link Edit step. The SYSLMOD DD card defines the output load module for the linkage editor and places the load module in the data set NADEV.

The next step is to run a NASTRAN problem to test the procedure implemented. Alters can be made to the DMAP program to extract contents of tables or of generated matrices. In addition, print statements can be made within the modified programs to print out calculated results. If these print statements are used, they should be activated by a specific DIAG that is not in use by the NASTRAN system. See pages 3.3-15 and 3.3-15a of the NASTRAN Programmer’s Manual for DIAGS not in use by the system. Figure 8 gives an example of the use of alter statements and demonstrates the use of DIAG setting for controlling debug printing. In debugging a modified link, a dump is quite helpful on the occurrence of a system fatal error. The most important part of the dump is the save area trace which lists the routines last used when the error occurred. Usually this is sufficient and a full dump is not necessary. NASTRAN has built into the system a use of the SNAP macro which dumps only the save areas. Use of DIAG 1 will produce a full dump.

The link edit procedure for NASTRAN links is rather costly on the IBM 370-165 because of the extremely large number of segments in each of the links. Hence, whenever possible, we did as much checking of a modified module with a nonoverlaid FORTRAN run. For example, in checking out the stiffness matrix routines for the isoparametric elements we ran a simple model in NASTRAN, and with the Alter statements we printed the content of the ECPT (Element Connection and Properties Tables). A main program which simulates SMAL was written to supply the proper ECPT to the element stiffness matrix routines and the element matrices were generated and printed out. When we were satisfied that the routines operated properly we modified our link-edit control deck and link edited the new element stiffness matrix routines into our data set NADEV. A run of the same model would then produce the element stiffness matrices, displacements, and stresses. Figure 7 is our JCL and control deck for the insertion of stiffness matrix routines into link 3.

The procedure for the variable number of grid points for mixed elements (one that is not full) was implemented in the following manner. The connection cards for the element were left blank at positions where grid points were missing from the full element. A modification of TAIA and TALB was made to enter a zero as the grid point number for the missing grid points. For the grid points present, the degree of freedom for that grid point (a nonzero value) was entered as the grid point number. All tables such as ECPT, EST (Element Summary Table) have nonzero values for grid points present in the element and zeros for missing grid points. The length of the grid point table is fixed for each element type, for example, twenty for CSOLID2, the quadratic isoparametric. This table is then used as a guide to all processing of the mixed element. The modifications to TAIA and TALB were supplied to us by Carl Hennrich of MacNeal-Schwendler Company.
The procedure used for saving the element matrices and not recreating them each time they are needed was as follows: A scratch tape was assigned to be used in the element matrix subroutine. This tape had to be assigned a GINO buffer at a level where all buffers are assigned for this module. Also an array had to be assigned for record keeping of saved elements. Initialization of counters had to be done at the level where the buffer was assigned. At the element level the routine would first ask if this element had previously been encountered. This is done by a search through the table of all elements that have been saved. If found, the tape record number is extracted from the table and the tape is positioned by GINO commands to the proper record. The record is read into core and the sub matrices needed for this call are assembled from the total element matrix and given to the subroutine which is assembling the total mass or stiffness matrix. If this element has not been encountered, the element matrix is calculated, and the tape is positioned to the end of last element written, and the new element matrix is written on the scratch tape. The element number and record number is entered into the table. This procedure was suggested in the Programmers Manual, Page 4.87-1, last two sentences of paragraph 3. GINO proved very useful here in that the records saved were of variable length because of the three types of elements and because of the use of mixed elements within a type. The variable length could be stored in the record and using GINO's capability of reading and writing segmented or partial records we could read the number of words for the variable length record. To add scratch tapes to an existing module the MPL (Module Property List) had to be modified by recompiling the block data program XMPLBD, see pages 2.4-21 and 2.4-22 of the Programmers Manual.

DISCUSSION

The new Programmers Manual for version 15.1 has an excellent chapter on adding a structural element. This was an update of a NASA Fourteenth Quarterly Report for NASTRAN, January 1970. This chapter gives a step-by-step procedure of all routines and tables that have to be modified to accommodate a new element. The Fourteenth Quarterly Report aided us greatly in getting through most of the input problems of NASTRAN.

From this step-by-step procedure, one can see that adding a new element to NASTRAN is not that difficult because of the excellent documentation and supplies that have been distributed.

ACKNOWLEDGMENTS

The authors would like to thank S-Y Lien and W. VanBuren for providing the isoparametric routines and C. W. Haanrich, R. Gillian, and M. M. Hurwitz for NASTRAN consultation.
REFERENCES


4. MacNeal, R. H., ed.: The NASTRAN Theoretical Manual. NASA SP-221(01), 1

Figure 1.- Mixed 3-D isoparametric element.
Variable number of grid points.
Figure 2.- VTOS distributed system V15.
Figure 2.- Concluded.
Figure 3.- VTG of developmental system.
//PUNCH JOB (ROxxxxx,ROO) *MBREWMAN* REGION=100K, TIME=1:39 JOB CARD
//VOLREG ID: (NASTRAIN15) DISTRIBUTED SYSTEM V.15 EXEC PGM=EBPUPCH
//SYSPRINT DD SYSPRINT A
//SYSUT2 DD UNIT=SYSUTA,VOL=SER=VOLNUM
//DCE=(RECFM:FB,LRECL=80, BLKSIZE=7230)
//SYSUT2 DD SYSOUT=3
//SYSIN DD *
//PUNCH TYPORG=PO, MAXNAME=65, MAXFDCS=60
// MEMBER NAME=DS1 RECORD FIELD=(8D)
// MEMBER NAME=DS1A RECORD FIELD=(8D)
// MEMBER NAME=DS1AD RECORD FIELD=(8D)

/*
NOTE: SOU1 IS DATA SET NAME, VOLNUM IS DISK VOLUME SERIAL NUMBER
*/

Figure 4.- Punch source from SOU1.

//PRINT JOB (ROxxxxx,ROO) *MBREWMAN* REGION=100K, TIME=1:39 JOB CARD
//VOLREG ID: (NASTRAIN15) DISTRIBUTED SYSTEM V.15
//FORMAT PR:DDNAME=SYSUT2,TRAIN:HN EXEC PGM=EBPUPCH
//SYSPRINT DD SYSPRINT A
//SYSUT1 DD UNIT=SYSUTA,VOLUME=SER=VOLNUM,DISP=OLD
//DCE=(RECFM:FB,LRECL=80, BLKSIZE=7230,DSN=SOU1)
//SYSUT1 DD SYSOUT=A
//SYSIN DD *
//PRINT TYPORG=PO, MAXFDCS=80, MAXNAME=80, MAXLINES=85
// MEMBER NAME=READI RECORD FIELD=(8D)
// MEMBER NAME=READ2 RECORD FIELD=(8D)
/*
Figure 5.- Prints source from SOU1.
*/
Figure 6.- Compiles and puts object into development data set NEWOBJ.
Figure 7.- Link edit and puts execution module into NADEV development set.
Figure 7.- Concluded.
Figure 8.- VTOC example and use of ALTER for table and matrix printouts.
**Figure 8.** Concluded.