Geometry import to LS-DYNA

Livermore Software Technology Corporation

1 Introduction

The support of novel computer-aided geometric descriptions forming a potential future basis of isogeometric analysis in LS-DYNA is discussed in the present document. In particular, the design of a new keyword as well as the structure of the geometry input file meant to replace the current method using the *INCLUDE_TRANSFORM keyword is outlined. It is also aimed to generalize the geometric description as well as to focus on compressed storage in order to enable the run of larger and more complex examples.

The remaining part of the document is structured as follows. The new LS-DYNA keyword is introduced in section 2. Supported geometry file formats are discussed in section 3 in greater depth.

2 LS-DYNA keyword

The *IGA_INCLUDE_{*OPTION1*}_{*OPTION2*} keyword is introduced to import geometry files to LS-DYNA with NURBS or BEZIER as supported first optional arguments and blank or TRANSFORM as second optional argument, i.e.

Card 1	1	2	3	4	5	6	7	8
Variable	FILENAME							
Type	C							

Card 2	1	2	3	4	5	6	7	8
Variable	FTYP	PID	FORM	MASS	INT	NISR	NISS	NIST
Type	I	I	I	I	I	I	I	I
Default	none	none	0	0	0	none	none	none

Card 3	1	2	3	4	5	6	7	8
Variable	ID-NOFF	IDE-OFF	ID-POFF	FC-TLEN	TRA-NID			
Type	I	I	I	F	I			

Variable	Description
FILENAME	Name of file to be included.
FTYPE	File type. 1 - ASCII 2 - LSDA
PID	Part ID.
FORM	Element formulation type.
MASS	Mass matrix lumping scheme.
INT	Lamina integration rule.
NISR	Number of interpolation elements in the r-direction.
NISS	Number of interpolation elements in the s-direction.
NIST	Number of interpolation elements in the t-direction.
IDNOFF	Offset node ID.
IDEOFF	Offset element ID.
IDPOFF	Offset part ID.
FCTLEN	Length transformation factor.
TRANID	Transformation ID.

Remarks:

(1) The above keyword structure is valid for any of the options, however, the use of *IGA_INCLUDE_NURBS keyword is not detailed further in this document.

(2) One file per *IGA_INCLUDE keyword. The file, however, may contain multiple patches with the same part ID, element formulation, mass lumping scheme, as well as lamina integration rule as defined on Card 2.

(3) Card 2 is meant to be used with both options. This card contains non-geometrical information also used in other, e.g. *ELEMENT_SHELL_NURBS_PATCH or *ELEMENT_SOLID_NURBS_PATCH, keywords.

(4) The optional card 3 contains all fields from the *INCLUDE_TRANSFORM keyword relevant for geometric entities. Offsetting element, node, or patch IDs will only make a difference, i.e. be useful, if Bézier and standard finite element meshes are combined in a model.

3 Geometry file formats

The key differences with respect to the previous format are as follows:

(1) *Reduced storage.* Bézier extraction operators are not stored in their full form henceforth. In what follows, we distinguish between tensor product, non-tensor product, and mixed elements. A d-dimensional tensor product element is defined by d local knot vectors. A non-tensor product element is defined by a set of coefficient

vectors essentially representing a row in the Bézier extraction operator. Elements with mixed tensor and non-tensor product structure, e.g. prism cf. section 3.1, may be defined using a combination of local knot and coefficient vectors.

(2) *Sorted input.* Local knot and coefficient vectors are collected into sorted blocks comprised in a library. Element definitions use local knot and/or coefficient vector identifiers, i.e. pointers, to the entries of the library. Furthermore, a coefficient vector may be stored using either dense or sparse storage format. Noting that the latter may be beneficial as the element dimension increases but complicate the export of the data, the choice to invoke different storage formats is left to the pre-processor. Assuming for instance that tensor product elements are used to define most part of the discretization and non-tensor product elements occur in the vicinity of a few extraordinary points only, it might be easier to export the data in dense format only.

(3) *Format and precision.* In order to ensure consistency with the binary input, cf. section 3.2, a fixed input format is proposed using (due to the relative indexing) short integers, i.e. `i8`, and double precision reals of the form `1pe24.16`. Consequently, each line may contain up to ten integers or five reals yielding lines of up to 80 or 120 character long, respectively.

For brevity, local knot vectors are also referred to as coefficient vectors henceforth.

3.1 ASCII format

The following structured input has to be written for each patch separately, i.e.

BLOCK 1 - PATCH
PAID, PADIM, NN, NE, NCV, WFL
Total number of lines: 1.

BLOCK 2 - NODES
For each node $i = 1, \ldots,$NN:
Xi, Yi, Zi, Wi
Total number of lines: NN.

BLOCK 3 - ELEMENTS
NEB
For each element sub-block $j = 1, \ldots,$NEB:
NEj, NNj, NCVj
For each element in sub-block j:
ETYP, PR, PS, PT
N1, N2, ... , Nk (as many lines needed)
CVID1, CVID2, ..., CVIDl (as many lines needed)
Total number of lines: $1+\text{NEB}+\sum_{j=1}^{\text{NEB}}[1+\texttt{ceil}(\text{NN}j/10)+\texttt{ceil}(\text{NCV}j/10)]$.

BLOCK 4 - COEFFICIENT VECTORS
NDCVB, NSCVB
For each dense sub-block $d = 1, \ldots,$NDCVB:
NCVd, NCVCd
For each sparse sub-block $s = 1, \ldots,$NSCVB:
NCVs, NCVCs
For each coefficient vector in dense sub-block d:
CVID
CVC1, CVC2, ..., CVCd (as many lines needed)
For each coefficient vector in sparse sub-block s:
CVID
CVI1, CVI2, ..., CVIs (as many lines needed)
CVC1, CVC2, ..., CVCs (as many lines needed)
Total number of lines: 1+NDCVB+NSCVB+
$\sum_{d=1}^{\text{NDCVB}}[1+\texttt{ceil}(\text{NCVC}d)/5]+\sum_{s=1}^{\text{NSCVB}}[1+\texttt{ceil}(\text{NCVC}s)/10+\texttt{ceil}(\text{NCVC}s)/5]$.

where

Variable	Description
PAID	Patch ID.
PADIM	Patch dimension, i.e. 1 - Beam. 2 - Shell. 3 - Solid.
NN	Number of nodes/control points.
NE	Number of elements.
NCV	Number of coefficient vectors.
WFL	Control weight flag.
Xi, Yi, Zi	Nodal coordinates of the ith node.
Wi	Nodal weights of the ith node.
NEB	Number of *sorted* element sub-blocks, i.e. based on the number of nodes and coefficient vectors used in their definition elements are sorted into $j = 1, \ldots,$NEB sub-blocks.
NEj	Number of elements in the jth sub-block.
NNj	Number of nodes defining an element in the jth sub-block.
NCVj	Number of coefficient vectors defining an element in the jth sub-block.
ETYP	Element type. 0 - Cube (tensor product) 1 - Cube (non-tensor product)

	2 - Simplex (non-tensor product) 3 - Prism (tensor product in one direction only)
PR	Polynomial degree in the r-direction.
PS	Polynomial degree in the s-direction.
PT	Polynomial degree in the t-direction.
Nk	Node IDs defining the element connectivity, $k = 1, \ldots,$NNj in the jth sub-block.
CVIDl	Coefficient vector IDs defining the element, $l = 1, \ldots,$NCVj in the jth sub-block.
NDCVB	Number of *sorted* coefficient vector blocks using *dense* storage format, i.e. the coefficient vectors are stored into $d = 1, \ldots,$NDCVB sub-blocks based on their length.
NSCVB	Number of *sorted* coefficient vector blocks using the *sparse* storage format, i.e. the coefficient vectors are stored into $s = 1, \ldots,$NSCVB sub-blocks based on their length.
NCVd (NCVs)	Number of dense (sparse) coefficient vectors in the dth (sth) sub-block.
NCVCd (NCVCs)	Number of dense (sparse) coefficient vector components in the dth (sth) sub-block.
CVID	Coefficient vector ID.
CVCm (CVCn)	Coefficient vector components using the dense (sparse) storage format, $m = 1, \ldots,$NCVCd ($n = 1, \ldots,$NCVCs) in the dth (sth) sub-block.
CVIn	Coefficient vector index, $n = 1, \ldots,$NCVCs (sparse format only).

Remarks:

(1) Beam elements, i.e. PADIM=1, are currently not supported in LS-DYNA.

(2) The element dimension is the same as the part dimension, i.e. a shell patch should not contain any three-dimensional or solid elements and vice versa.

(3) Element and node IDs are local/relative to the patch and therefore are not defined at input.

(4) For the sake of generality, a non-tensor product cube (ELTYP=1) may also be used to define tensor product cubes (ELTYP=0). This may be useful in case local knot vectors can not be retrieved from Bézier extraction operators in higher dimensions.

3.2 Binary format

In addition to the ASCII format outlined in the previous section, we intend and in most industrial cases prefer to support binary storage of the geometry using the open LSDA format and API developed and maintained by LSTC. Invoking the binary format will further reduce storage requirements, speed up I/O. The data should be written using the following path `keyword/[option]/patch[i8.8]/` where `[option]` is either `isoshell` or `isosolid` for two and three-dimensional patches, respectively.

> Bézier extraction was originally introduced to enable the analysis of unstructured spline discretizations in existing finite element codes. As a consequence, topological information relevant for models defined fully or partially by virtue of Bézier extraction, e.g. element adjacency for a single patch or interface between multiple patches including for instance a hat stiffened shell, is lost. The present document wasn't aimed to address the storage of topological information and the authors are not even convinced that such data should be part of the data structure at all.