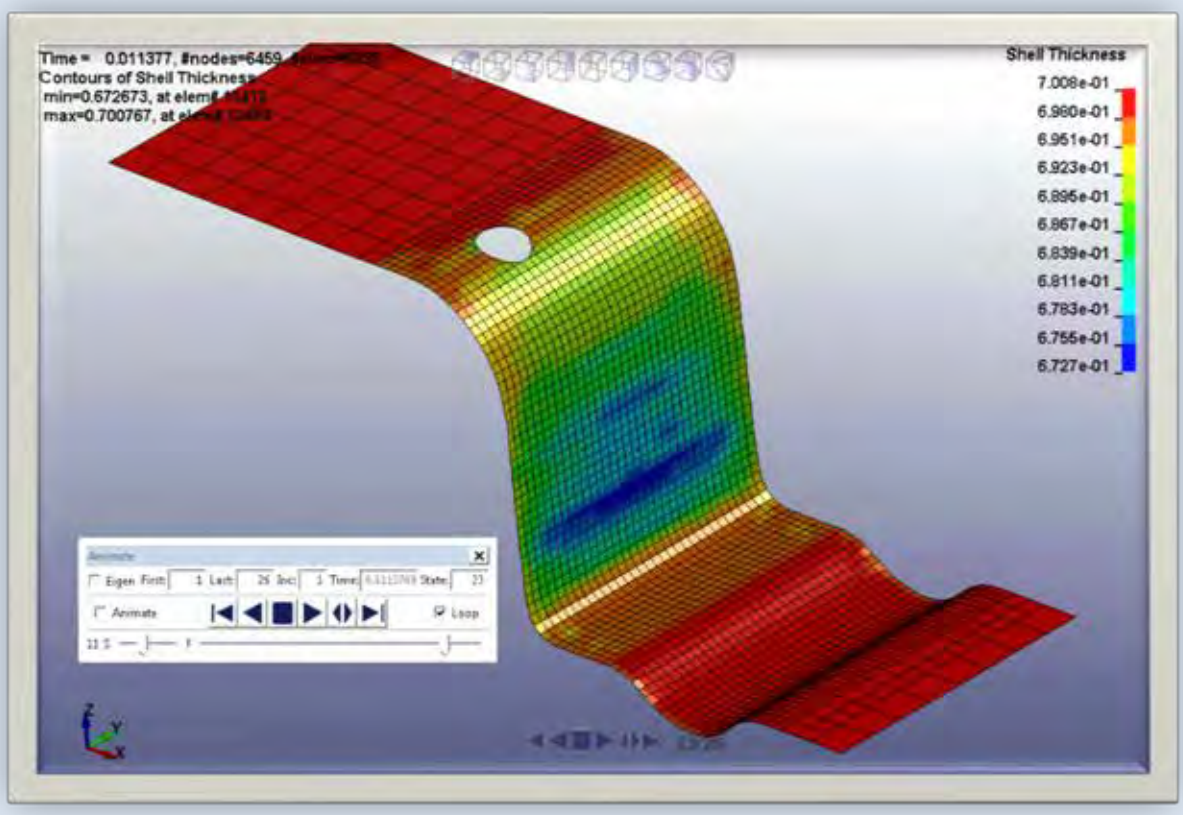
ISSN 2167-1273
Volume 6, Issue Q1, February 2017

FEA Information Engineering Journal



Instant lancing with trimming
LSTC Technical Articles/Short Subject

Aim and Scope

FEA Information Engineering Journal (FEAIEJ™) is a quarterly published online journal to cover the latest Finite Element Analysis Technologies. The journal aims to cover previous noteworthy published papers and original development updates. All published papers are peer reviewed in the respective FEA engineering fields.

Consideration is given to all aspects of technically excellent written information without limitation on length. All submissions must follow guidelines for publishing a paper, or periodical. If a paper has been previously published, FEAIEJ requires written permission to reprint, with the proper acknowledgement give to the publisher of the published work.

Monthly publication is limited to no more then five papers or informational articles of interest, either reprint, or original. Papers will be archived on www.feaiej.com

For information on publishing a paper original or reprint contact editor@feaiej.com Subject line: Journal Publication

Cover:

Fig. 3 the instant lancing with trimming

Article: 01_Lancing features in LS-DYNA

Quanqing Yan, Li Zhang, Yuzhong Xiao, Xinhai Zhu, Philip Ho

Livermore Software Technology Corporation

FEA Information Engineering Journal

TABLE OF CONTENTS

Publications are © to the authors and are informational short subject publishings

Lancing features in LS-DYNA

Quanqing Yan, Li Zhang, Yuzhong Xiao, Xinhai Zhu, Philip Ho
LSTC

Introduction

Lancing the blank in the sheet metal forming can be strategically used under controlled conditions to alleviate the thinning and necking of the sheet metal panels. A brief introduction of the lancing features (instant lancing, progressive lancing and the lancing with trimming) in LS-DYNA will be presented as below, along with the Graphical User Interface for the lancing definition in LS-PrePost EZSetup.

Keywords for the lancing feature in LS-DYNA

To use the lancing feature in LS-DYNA, *ELEMENT_LANCING and *DEFINE_CURVE_TRIM_3D would be needed. In *ELEMENT_LANCING, the blank to be lanced (IDPT), the lancing path (IDCV) and the staring time (AT) can be defined. The lancing path can be specified in *DEFINE_CURVE_TRIM_3D. An excerpt of the keyword is shown as below:

```
*ELEMENT_LANCING
$      IDPT       IDCV    IREFINE       SMIN         AT       ENDT     NTIMES
  &blk1pid      90953          1                    &at      &endt    &ntimes
*DEFINE_CURVE_TRIM_3D
$#     tcid     tctype       tflg       tdir      tctol       toln     nseed1     nseed2
      90953          1          1          0
$#                 cx                    cy                    cz
       -3.253909e+001        8.811000e+000        3.907000e+001
       -3.288404e+001        8.831327e+000        3.907000e+001
◦ ◦ ◦ ◦ ◦ ◦
```

If the ending time (ET) is left blank, the instant lancing will be performed; otherwise the progressive lancing will be performed. The cutting timesin this process can be further defined by specifying the NTIMES.
If the lancing is followed by the trimming process, the remaining zone in the blank can be specified in *DEFINE_LANCE_SEED_POINT_COORDINATE, as show below:

```
*DEFINE_LANCE_SEED_POINT_COORDINATES
$#    NSEED         X1         Y1         Z1         X2         Y2         Z2
          1   -50.6727      19.58      39.07
```

More details and the examples of the lancing feature can be found in the LS-DYNA Keyword User's Manual Vol.1.
It should be emphasized that the adaptivity (*CONTROL_ADAPTIVE) of the blank mesh should always be enabled in the lancing simulation.

The Graphical User Interface for the lancing setup

To simplify setting up the lancing process in LS-DYNA, the graphical user interface (GUI) is implemented in LS-PrePost EZSetup. The lancing process can be defined in the Punch page of the Forming in the EZSetup, as shown in Fig. 1.

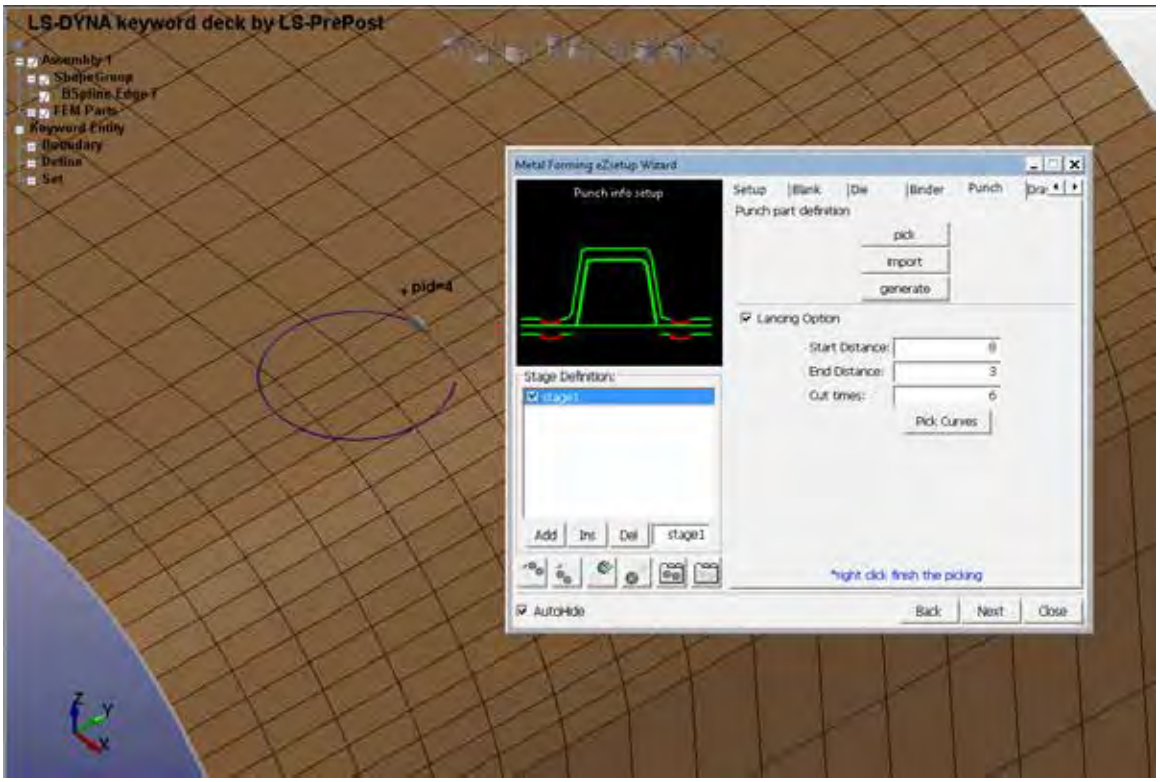


Fig. 1 the lancing setup in LS-PrePost EZSetup

For the instant lancing, the path can be defined by picking the curve in the IGES format in LS-PrePost. Instead of specifying the starting time, the distance to the tool home position is used in EZSetup to specify the starting of the lancing process. The corresponding keywords will be generated by EZSetup in the inputdeck of the forming process..

For the progressive lancing, the End Distance and the Cut Times would be further needed. As shown in Fig. 1, the direction of the lancing path can also be previewed in the main window. If necessary, the lancing direction can be reversed via LS-PrePost: GeoTol->Rever. Setting up the lancing with trimming would be supported in the later version of LS-PrePost EZSetup.

Currently for simulation of the lancing with trimming, manual modification of the generated keyword file would be necessary to include the following keyword as mentioned above: *DEFINE_LANCE_SEED_POINT_COORDINATES.

Examples of the lancing simulations

An example of the instant lancing with trimming is given in Fig.3. The blank is lanced and trimmed when the upper tool is at 5mm to the home position,. Fig.4 shows a progressive lancing process, in which the blank is lanced when the upper tool is moving from 8mm to 2mm until its home position.

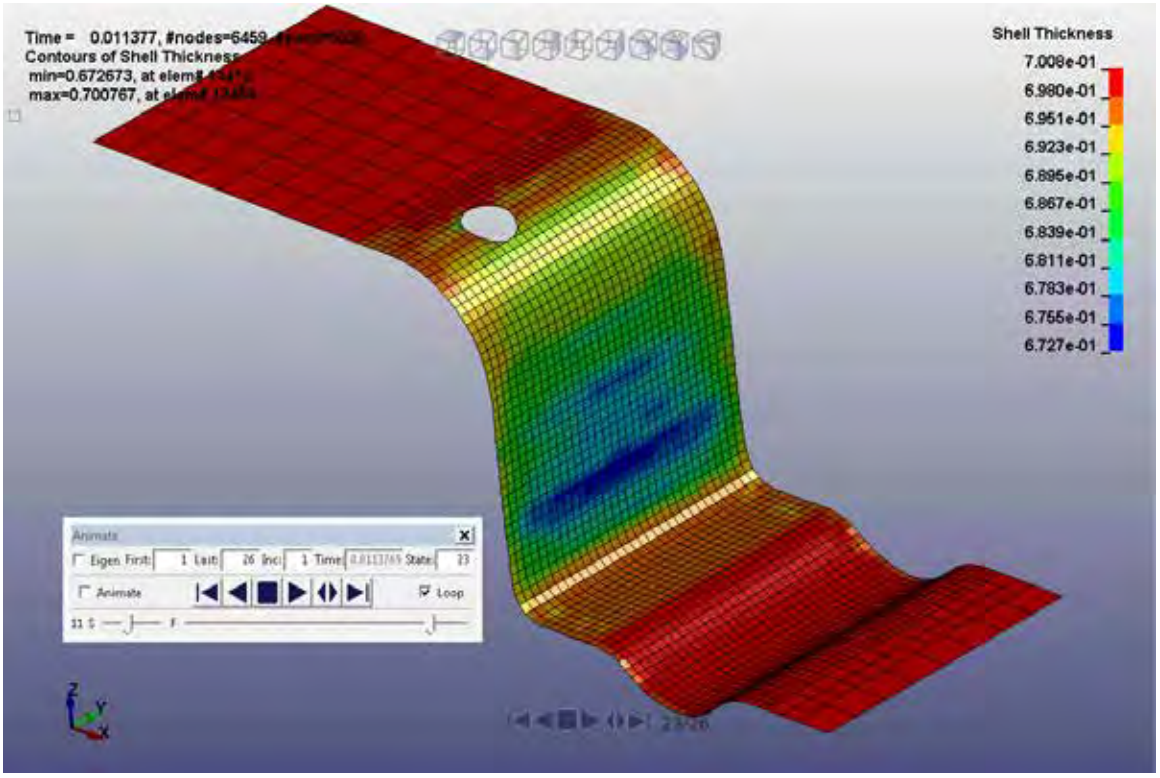


Fig. 3 the instant lancing with trimming

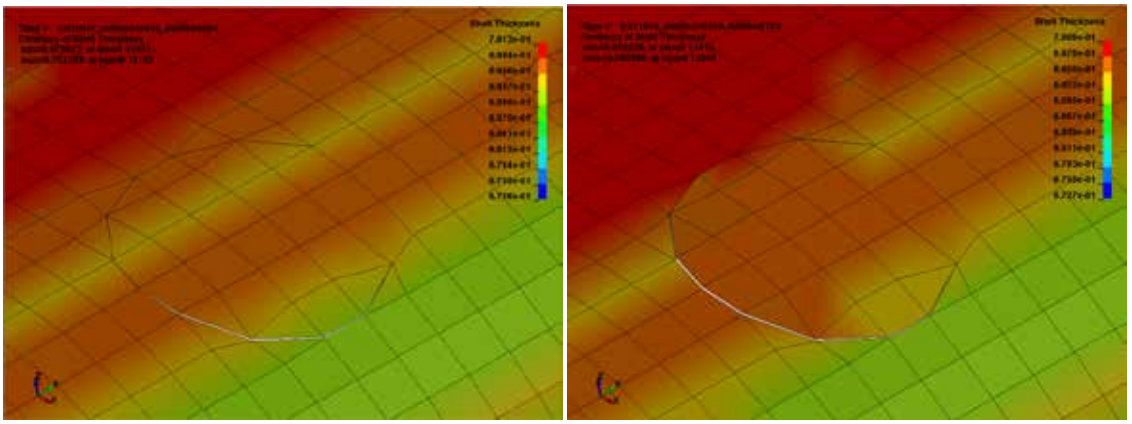


Fig.4 the progressive lancing

Revisions

The lancing feature in LS-DYNA is available starting in Revision 83562 for SMP, and Revision 94383 for MPP. For lancing with trimming, Revision 107262 or later is required. The newest beta solver is recommended due to the continuous improvements.
The GUI of lancing setup can be found in LS-PrePost 4.3 -> Applications -> Metal Forming: EZSetup

Summary

The keywords for the lancing feature of LS-DYNA, and the setting up via the GUI in LS-PrePost EZSetup, were briefly presented. With the lancing feature, the capability of the LS-DYNA in simulating the metal forming is further enhanced. The GUI in LS-PrePost EZSetup would efficiently help the users set up the lancing process in a straightforward manner.

LS-DYNA Smooth Particle Galerkin (SPG) Method

C.T. Wu, Y. Guo, W. Hu
LSTC

Element-free Galerkin (EFG) meshless method was introduced into LS-DYNA more than 10 years ago, and has been widely used in the solid and structure analyses. Compared to the conventional FEM, EFG is a better alternative in terms of numerical accuracy and capability for handling large material deformation. However, because of the constraint in using background mesh for numerical integration, EFG faces the similar difficulty as FEM in the application involving extreme large deformation and material failure. On the other hand, meshless method using nodal integration has been a highly active research area in the past decade. In recent years, we have been developing a new meshless method, called Smooth Particle Galerkin (SPG) [1,2,3], and improving its capability for industrial applications. SPG is a true meshless method using nodal integration under Galerkin framework, where a special smoothing scheme in displacement field is introduced to stabilize the numerical solution. Meanwhile, for large material deformation in explicit analysis, we are able to maintain the time step size by combining this smoothing scheme with kernel update, which helps to improve the overall computational performance. In this paper, we are going to briefly introduce the latest development of SPG and its keywords. Some numerical examples are presented to demonstrate its capability in manufacturing analysis involving large deformation and material failure.

SPG is currently implemented in LS-DYNA for solid analysis with element formation ELFORM=47 in the keyword *SECTION_SOLID_SPG. The FEM mesh (4/6/8-noded solid element) is automatically converted to SPG particles in LS-DYNA. The following is a snapshot of SPG keyword cards：

Card 2	**DX**	**DY**	**DZ**	**ISPLINE**	**KERNEL**	**LSCALE**	**SMSTE**	**SUKTIME**
Default	1.5	1.5	1.5	0	0		15	
Card 3	**IDAM**	**SF**						
Default	0							

(1) Nodal support size：DX, DY, DZ

Like many other meshless method, the approximation function in SPG is constructed based on discrete nodes, which, by default, are from FEM model. The support size of a given node is determined by the size of surrounding element edges with the scaling parameters DX, DY and DZ. For non-uniform mesh, the absolute nodal support sizes vary across the computational domain due to the variation of element size. The recommended range of scaling parameters in SPG is 1.4~1.8, and the default value, 1.5, is good for most of applications.

(2) Kernel types：KERNEL

SPG currently has two different kernels: updated Lagrangian kernel and Eulerian kernel with KERNEL=0 and 1, respectively. L-kernel is suitable for large deformation analysis without material failure, for example, rubber-like and foam materials, while E-kernel can be widely used in the application involving extreme large deformation and failure of ductile, EOS and solid fluid materials. Standard Eulerian kernel has tensile instability issue, which leads to numerical failure (different from the true physics-based material failure). The E-kernel in SPG is constantly updated according to the material deformation in order to avoid the tensile instability issue.

(3) The frequency of numerical smoothing in the displacement field：SMSTE

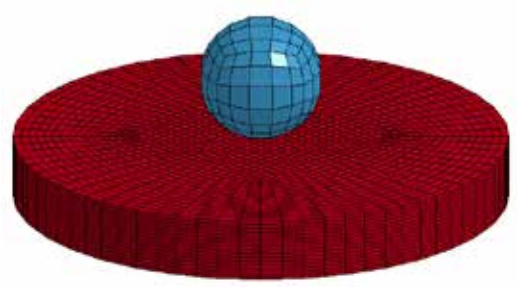
The smoothing scheme is introduced to stabilize the numerical solution. SMSTE defines the frequency of smoothing by the number of time steps. Note that over smoothing will significantly increase CPU time and lower the solution accuracy, while insufficient smoothing often results in numerical oscillation and instability. The default value applies to most of solid and structure analyses. In practice, SMSTE is related to the scaling parameter TSSFAC (*CONTROL_TIMESTEP). The recommended range of TSSFAC for SPG is 0.1~0.3. The larger TSSFAC is set, the smaller SMSTE is needed, or vice versa.

(4) Failure criteria：IDAM & SF

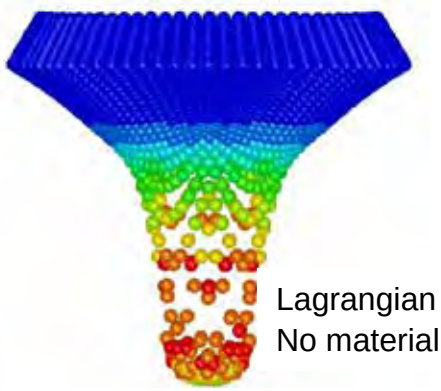
Material failure is a very complicated process in physics. LS-DYNA has a large material library where most of failure models are mainly empirical based on parameters calibrated by experiments. By setting IDAM=0, SPG supports these material models. According to the type of application and users' preference, the failed nodes can be either eroded (*MAT_ADD_EROSION) or treated as discrete ones interacting through contact. It is known

that material failure is, by nature, a multiscale problem. The development of material failure model and corresponding numerical tools requires a lot of fundamental studies, which has been one of very important R&D directions in LSTC. We are looking forward to the support from both academia and industry. In SPG, we developed a bond-based failure criteria (IDAM=1), where the average effective plastic strain (EPS) of paired nodes in support zone is examined and compared to the user input value SF. On top of that, we considered the bond stretching as well. In the following numerical examples, we demonstrate that SPG (IDAM=1) works pretty well in various applications.

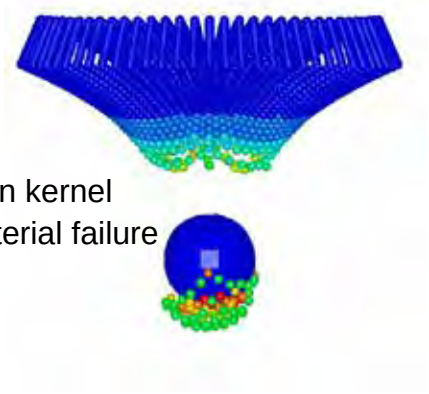
The first example is an impact analysis with rigid ball and metal plate. We tested the updated Lagrangian kernel (L), Eulerian kernel (E) and failure criteria IDAM=1. Note that the time step size keeps the same level through the analyses in all three cases. SPG (L) is able to deal with extreme large material deformation (no material failure). The numerical failure in case 2 using standard E-kernel can be corrected by SPG(E) with failure criteria IDAM=1 in case 3, where SPG well predicts the cracking behavior of metal plate under impact.



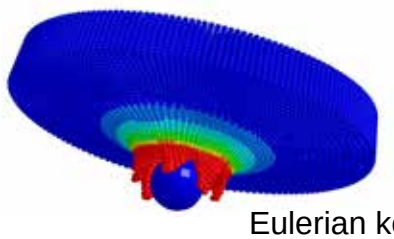
Computational model



Lagrangian kernel
No material failure

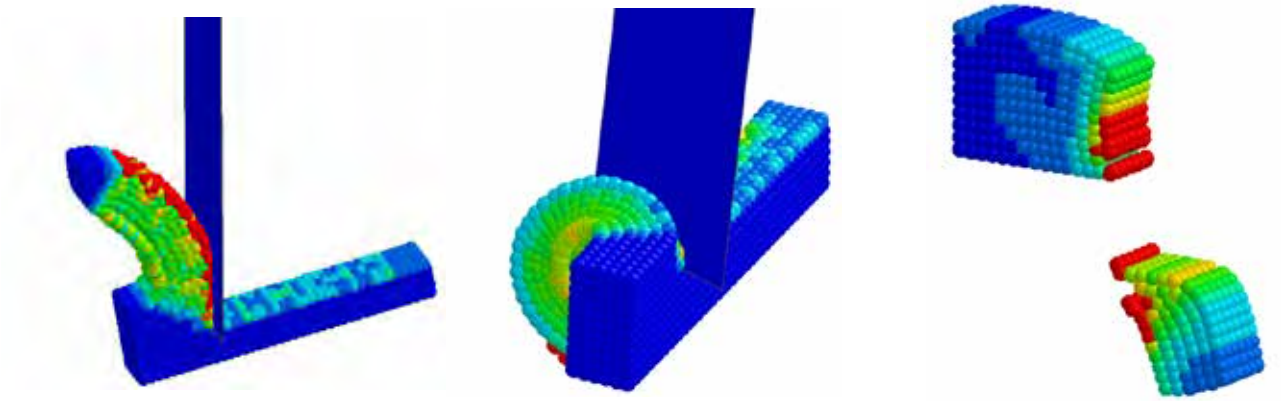


Eulerian kernel
No material failure

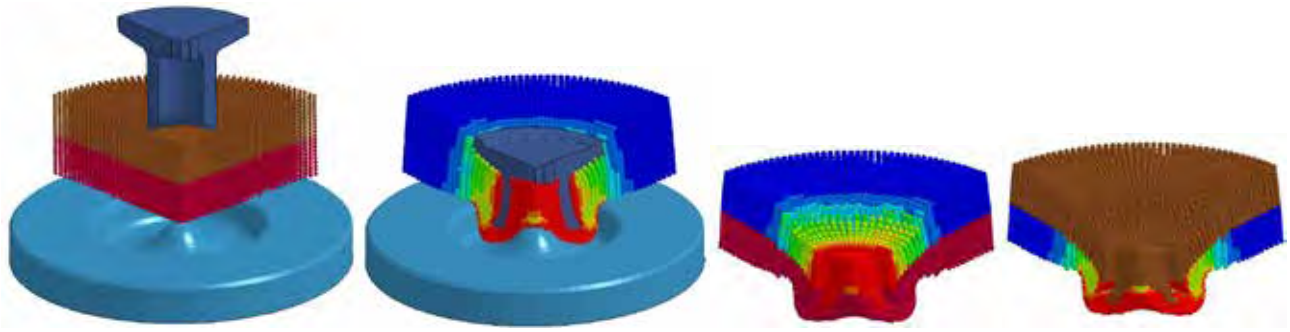


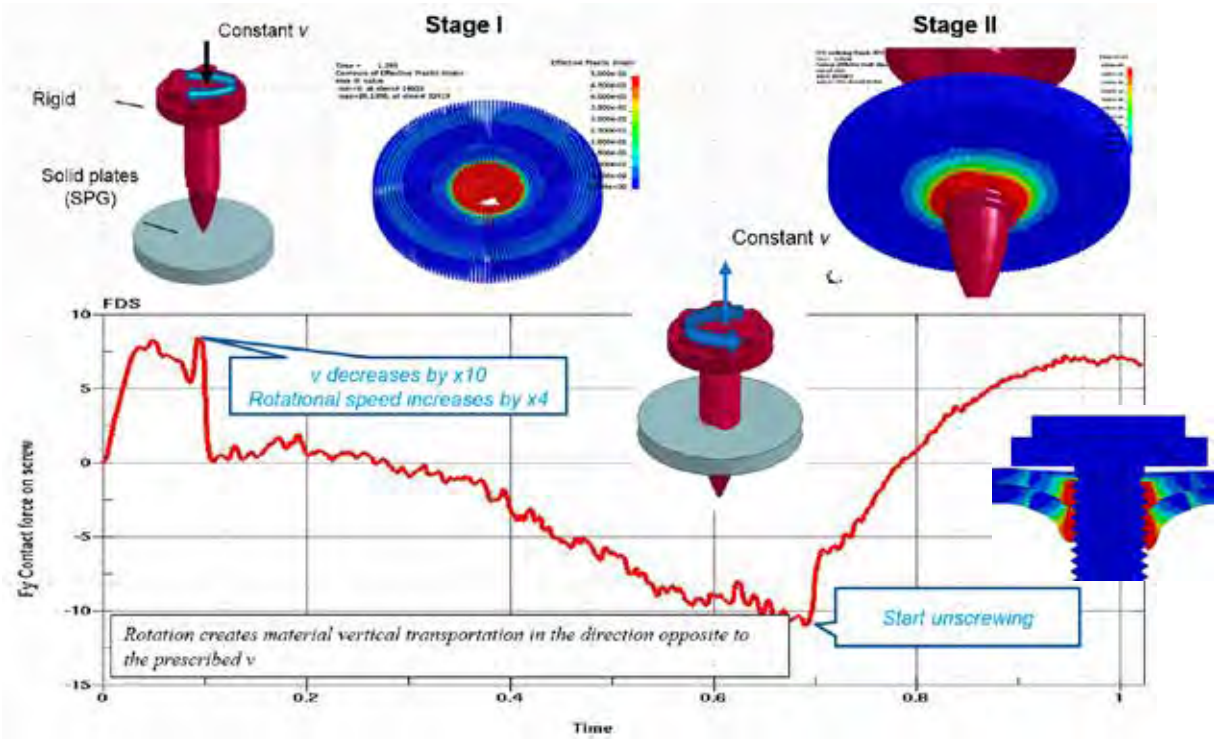
Eulerian kernel
w/ Material failure
IDAM=1, SF=1.0

The following shows analysis results of metal cutting and shearing problems using SPG.



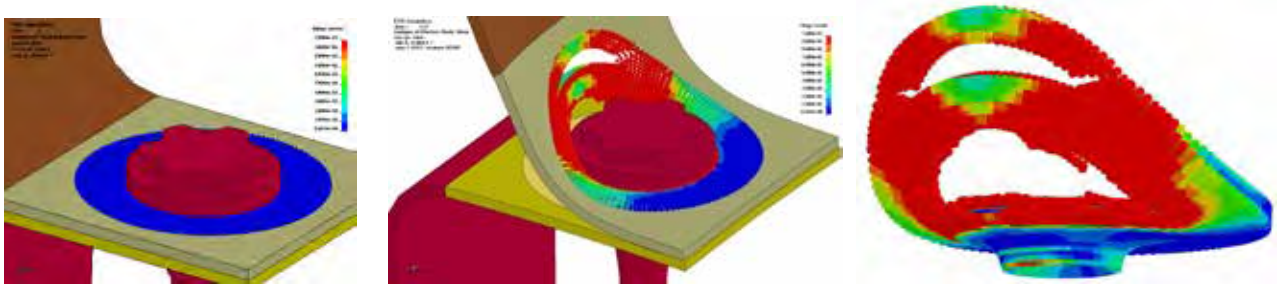
The next plot shows the riveting analysis using SPG, where both plates are modeled by SPG particles.





Friction Stir Drilling (FDS) is a very challenging problem for numerical simulation. The conventional methods such as element erosion have difficulty to capture the drilling threads, while adaptive re-meshing is very expensive especially when material failure is taken into account. The above figure shows that, by using SPG (IDAM=1), we can not only well capture the formation of the drilling threads but also successfully simulate the pull-out process.

In general, SPG costs 2~3 times more CPU time compared to FEM. In practice, it is recommended to apply SPG only in the area with large deformation and material failure. In the following example, SPG particles are used in the surrounding area of a FDS joint, where the interaction to the rest of the model is through sharing nodes with FEM along the interface.



As a new element formulation in LS-DYNA, SPG has been continuously improved and becoming more and more mature over the past few years. The SPG thermal-mechanical coupling solver and particle-to-particle contact will soon be released in LS-DYNA. For industrial applications, composite material, mesoscale modeling and high-velocity impact analysis will be potentially the new area for us to explore with SPG technology.

[1] Wu C.T., Hu W. and Koishi M., *International Journal of Computational Methods*, 2015

[2] Wu C.T., Koishi M. and Hu W., *Computational Mechanics*, 2015

[3] Wu C.T., Chi S.W., Koishi M. and Wu Y., *International Journal for Numerical Methods in Engineering*, 2016

* Dr. C.T.Wu graduated from the department of mechanical engineering in University of Iowa in 1999. His expertise is in advanced FEM and meshless methods. He joined LSTC in 2001, and has been working on the research and development for the solid and structure

analysis.

* Dr. Y. Guo graduated from the department of civil engineering in Northwestern University in 1999, and then joined LSTC. His expertise is in element technologies, meshless methods and fracture analysis.

* Dr. W. Hu graduated from the department of civil engineering in UCLA in 2007, and joined LSTC in 2009. He has been working on the research and development of adaptivity and meshless methods.

An Introduction to *CONSTRAINED_BEAM_IN_SOLID

Hao Chen
Livermore Software Technology Corp

Background

Rebar reinforced concrete is commonly used in construction industries. Its mechanical properties are of interest to people working in various engineering fields. While experimental, theoretical studies provided us essential guidelines to utilize this material efficiently and effectively, numerical simulations also showed their usefulness in predicting the overall structure behavior.

There are different techniques to simulate rebar reinforced concrete. One is to construct an inhomogeneous material model in which the concrete and the rebars inside were treated as a whole. This way, there is not explicit modeling of rebars. Instead they are assumed to be aligned along some specific directions inside the concrete solid elements.

Another way is to discretize rebars as beams and concrete as solids and make them share the same sets of nodes. Of course this requires extra efforts in mesh generations. It is not always doable if not cumbersome enough.

So an alternative technique becomes appealing to our users. It is to apply constraints between two set of nodes. One is for beams and another for solids. This way we avoid the meshing difficulties in "shared nodes" technique. Also, we don't need to construct complicated material models with the "composite material" approach.

Motivation

The rebar-concrete constraint coupling was done through a legacy keyword called *CONSTRAINED_LAGRANGE_IN_SOLID. This keyword is shared by two totally different applications. CTYPE=2 is used to model rebar coupling while CTYPE=4/5 is used to perform ALE fluid structure interactions. We will refer its rebar coupling function as "CLIS CTYPE 2" in the discussion below.

The CLIS CTYPE 2 had been widely used and proved being quite helpful in solving our users' problems. However, there are several flaws and shortcomings found by both our users and the author. Efforts were made to fix and enhance this function. But later the author found it was not possible to solve the fundamental error without overhauling it coding structure. He also found its implementation made it is very hard to add in new features requested by users.

In early 2015, the author started to develop *CONSTRAINED_BEAM_IN_SOLID to perform rebar constraint coupling. A new keyword was introduced for two reasons. First, the fix for CLIS CTYPE 2 was designed for beam only. However the "slave" in the legacy CLIS could also be other Lagrange entities such as node set, segment set and parts other than beams. The second reason is to be user-friendly. Too many times, the author witnessed users' confusion caused by the dual functionalities of the CLIS card. Also it has become a heavy, lengthy one with way too many flags so that even the most experienced user would frequently make input mistakes. So the author thought it would be the best to separate these two functionalities by giving rebar coupling

a new, dedicated keyword contains minimum input fields. This way, the author could also secure the input fields needed for new features.

Constraint coupling

Rebars and concrete are modeled by beams and solids, respectively. Beam mesh is submerged in solid mesh. Each of them has its own independent motion. Without some kind of coupling algorithm, they will move freely as if the other one doesn't exist at all. The way we couple these two is called "constraint method". There are always two parties involved in a constraint coupling. One is "master" and the other "slave". The slave contributes to the master and the master constraints the salve.

Typically both velocity and acceleration need to be constraint. The first is to ensure momentum conservation and the second force balance. The algorithm is exactly the same for both velocity and acceleration. For simplicity, we will limit the discussion below to velocity only.

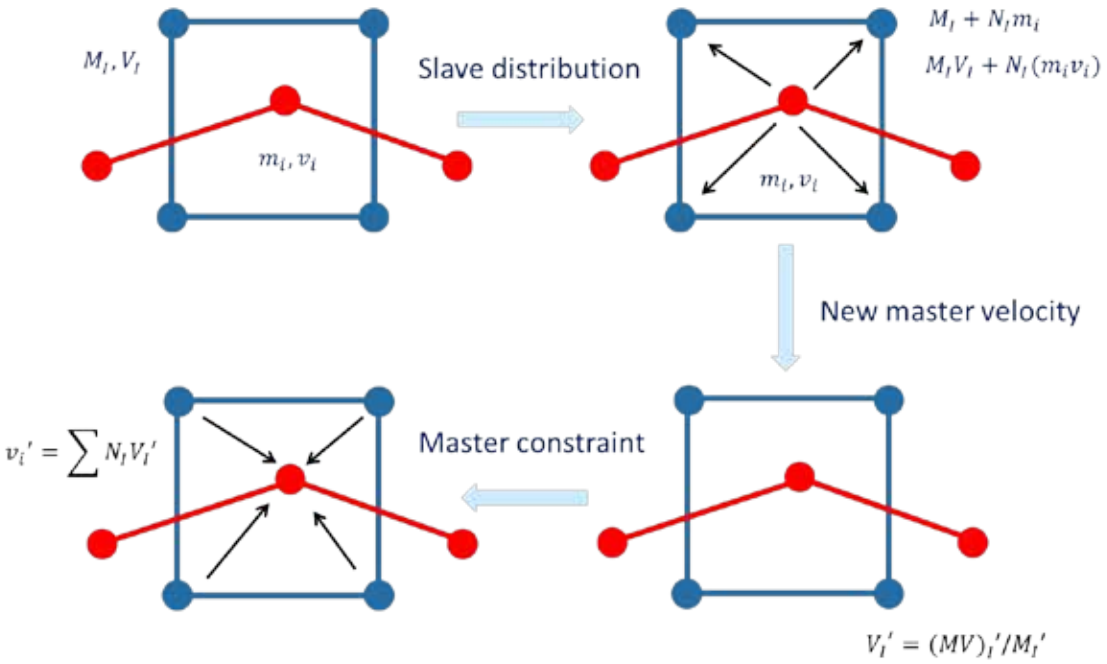
We start with an incompatible velocity field. Beam nodes are slave and denoted by lower case characters; solid nodes are master and upper case ones.

The first step is for slave beam nodes to distribute their nodal mass and momentum to master solid nodes.

Next we update the master nodal velocity by dividing the new momentum by new mass.

Finally we assign the interpolated velocity back to slave nodes.

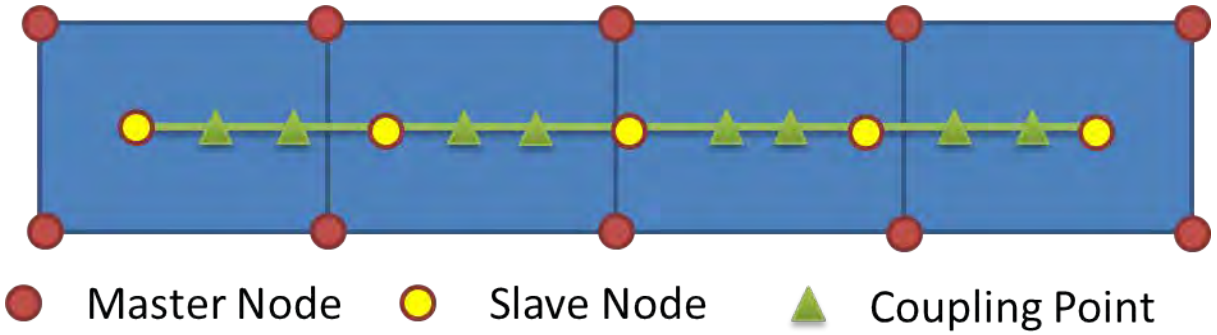
Now we have slave nodes moving exactly the same way as master nodes. The process is shown in the figure below.



Problems

The theory is straightforward. However, in real cases, the beam nodes are not always placed that well so that all solid elements contain at least one beam node. If a beam crossed certain solid element but its nodes did not fall in that solid, this solid won't get any distribution from this beam and the algorithm simply would fail.

So in both CLIS CTYPE 2 and CBIS we have an option to put extra "coupling points" in between the two end nodes of a beam element. This way, solid nodes get distributions either from beam nodes or these coupling points. This field is referred as "NQUAD" in CLIS or "NCOUP" in CBIS.

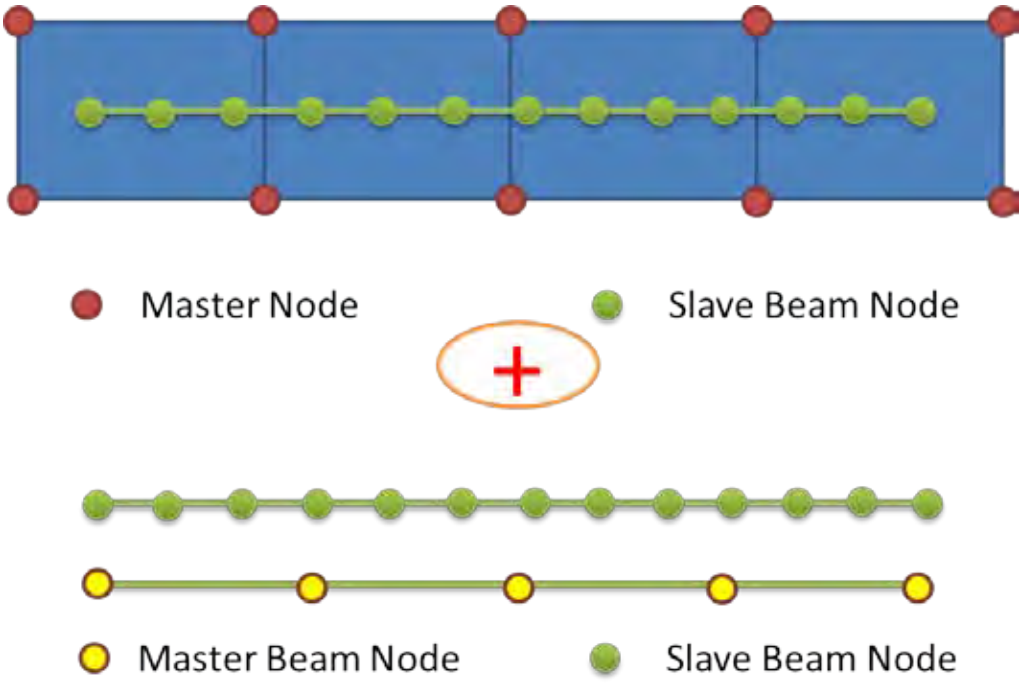


Now comes the puzzle, we all know each beam node has its nodal mass. This mass comes out naturally from discretization. Also it has its nodal velocity. These two entities are "physical". But for these artificially generated coupling points, there are no such properties. For velocity, it is pretty straightforward. We simply assume the velocity at a coupling point should be interpolated from the beam end nodes. How about mass?

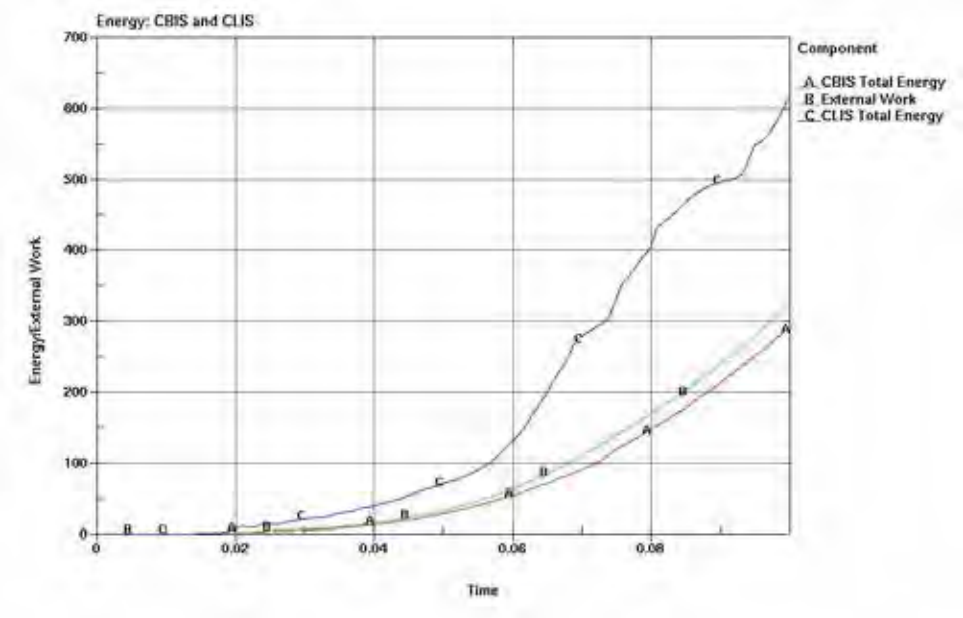
Unfortunately CLIS CTYPE 2 did not do it right. It moved half the beam element mass from nodes to these coupling points. This approach is rather arbitrary and lacks of theoretical basis. Another mistake it made was in the constraint process. The velocity was mapped only from master nodes to beam nodes, not to coupling points. So the overall process is not complete. These two errors won't reveal themselves if we only look at the structure motion. But when we checked the energy plot, we saw a spurious large internal energy increase.

Bridging Coupling

So how should we address this problem? The author came up with an idea which he called "bridging coupling". As the beam mesh is too coarse to be directly coupled to the solid elements, a "slave beam" is constructed in between to couple to both "master beam" and "master solid". Now we have two couplings. The first is between the "slave beam" and the "master beam"; the second between the "slave beam" and the solid mesh. "Slave beam" serves as a "bridge" connecting the real beam and solid elements. The concept is shown in the figure below.



We could see from the following figure that with the new CBIS implementation, the previously shown spurious energy increase disappeared. This mysterious spurious energy increase had puzzled both the author and our users for quite some time.



The mass at coupling point now simply takes the value of the “slave beam” nodal mass. It has a clear physical meaning and is theoretically correct. The coupling point has its velocity constrained by solid nodes during the “mapped-back” stage. And it then distributes the corresponding momentum to the “master beam” nodes.

The author does not intend to bother the readers with too many details. The idea of bridging coupling is conceptually simple and straightforward. However its implementation has been through some difficulties. It underwent several trial and error loops. Our users tested it extensively and provided valuable feedbacks.

Bucket sorting and searching

There were also some other improvements in CBIS. One deserves some explanation here. CBIS has an enhanced, independent bucket sorting and searching routine. That is in contrast to CLIS CTYPE 2, which shares these routines with ALE FSI coupling.

There are two advantages for CBIS to have its own sorting and searching routines. First, only solid elements belong to the "master" concrete are included in the bucket sorting. ALE elements won't be included. This brings memory reduction and a more efficient execution.

Secondly, as most ALE elements are hexahedron, searching subroutine, for efficiency, doesn't contain a separate treatment for tetrahedron and pentahedron. Rather they are treated as degenerated hexahedron. While in ALE FSI case it won't have too much difference, it is not acceptable for unstructured mesh used in rebar coupling. The new CBIS subroutine contains the enhanced algorithm to treat these three different solid elements separately.

Conclusion

By introducing the new *CONSTRAINED_BEAM_IN_SOLID keyword, we successfully fixed problems in the legacy *CONSTRAINED_LAGRANGE_IN_SOLID CTYPE 2. By separating this functionality from ALE FSI, we achieved both a clean keyword card and a clean code base for future development.

The author wants to express his gratitude towards our users for their effort testing this new keyword and valuable feedbacks. And hopes it could be of help to our users in solving their challenging problems.

Introduction to the new Framework for User Subroutine Development of LS-DYNA

Zhidong Han and Brian Wainscott
Livermore Software Technology Corp.

Abstract

A new framework has been introduced to LS-DYNA since the recent release R9.0. It allows all the LS-DYNA user subroutines to be complied into an independent dynamic library and loaded on demand. Multiple dynamic libraries can be loaded simultaneously. Conflicts resulting from multiple implementations of the same material models can be resolved through rules specified in the keyword file during run time. The new framework is completely backward compatible with the current user subroutines, without any changes to the existing source code. In the present paper, it is shown how to build, load and debug the dynamic library of the user subroutines. The rules are introduced for the case that multiple dynamic libraries are loaded.

Introduction

LS-DYNA is a general-purpose finite element program for multi-physics analysis. It has been widely used in many industries for several decades. LS-DYNA consists of a single executable and supports various state-of-the-art parallelization technologies. In addition, users have created their own customized versions of LS-DYNA through user subroutines to extend the applications of LS-DYNA. User subroutines support many features, including

- User material models (*MAT_USER_DEFINED_MATERIAL_MODELS)
- User thermal material models (*MAT_THERMAL_USER_DEFINED)
- User equations of state (*EOS_USER_DEFINED)
- User solid elements (*SECTION_SOLID)
- User shell elements (*SECTION_SHELL)
- Various utility subroutines, such as material failure criteria, load and boundary conditions, and outputs.

Of these, the user material models have been the most widely used. Within LS-DYNA, ten material IDs are reserved for user material models from the very beginning, i.e. UMAT41-UMAT50. Indeed, there have been numerous material models developed successfully. Many of them have also been adopted and included in the LS-DYNA releases. In recent years, a common request among LS-DYNA users is to allow the concurrent use of one user material ID from multiple model developers. This request is beyond the capability of the original development framework which was defined tens of years ago, which allows only one implementation of each user subroutine to be compiled and linked into the LS-DYNA executable.

A new framework for user subroutines has been introduced into the recent release, R9.0. It has a distinguishing feature: that LS-DYNA is completely decoupled from the user subroutines. It makes the use of the user subroutines more flexible

- One LS-DYNA executable is needed. No more customized executables.
- The LS-DYNA executable runs well with all features. No user subroutines are required if they are not used in the input deck.
- The LS-DYNA executable can load one single dynamic library during runtime if one set of user subroutines are implemented. It works as the original development framework and no rules are required.
- The LS-DYNA executable can load multiple dynamic libraries in which multiple implementations of the same user subroutines are allowed. The rules can be specified in the input deck to define how they are applied to different parts.

This feature is enabled with three new keywords which are documented in the R9 manuals. Keyword *MODULE_PATH specifies the path where the dynamic libraries are stored. Keyword *MODULE_LOAD specifies the file name of the library to be loaded. Keyword *MODULE_USE defines the rules on how to apply the loaded dynamic libraries to the corresponding parts in the model. These three keywords can be repeated as many times as necessary. This feature is still under development and currently supported by the Linux MPP executables. It requires a MODULE-enabled executable to support these three MODULE keywords.

In general, this feature supports all user subroutines, including the user material models, elements, utilities and so on. In the present paper, it is demonstrated through the use of the user material models only. Interested users may refer to the LS-DYNA manuals for more details.

Compilation of the user subroutines

The new framework provides a user subroutine package. All the source files are identical to those provided for the original framework. It allows the existing user subroutines to be reused without any changes. The only difference is in the build script, “Makefile”, in which all compiler settings are specified as variables, for an example a MPP package with the Intel compiler, as:

```
MY_FLAG = -fPIC -O2 -safe_cray_ptr -xSSE2 -align array16byte ......
FC = /opt/platform_mpi/bin/mpif90
LD = /opt/platform_mpi/bin/mpif90 -shared -nofor_main
export MPI_F77 := /opt/intel/composer_xe_2013.5.192/bin/intel64/ifort

MY_TARGET = demo.so
MY_OBJS = dyn21.o dyn21b.o init_dyn21.o ......
MY_INC = nlqparm define.inc define2.inc ......
```

where

MY_FLAG defines the compiler flags. It should not be changed in most cases.

FC and LD specify the MPI compiler and linker respectively which should match the MPI installation.

MPI_F77 specifies the Intel Fortran compiler which should match the compiler installation.

MY_TARGET defines the output file name of the dynamic library. It can be changed accordingly.

Once done, run “make” in the same folder and the dynamic library will be built and saved as “demo.so” which is specified in “MY_TARGET”. This dynamic library can be tested by running the MODULE-enabled LS-DYNA executable with the sample input deck provided in Appendix A.

The user can make the changes to the source files provided in the user subroutine package, or add more source files by changing the variable “MY_OBJS” in “Makefile”.

The compiler flags defined in “MY_FLAG” are set for building the release version of the user subroutines. The option “-O2” may be replaced with “-g -O0” to build the source code for debug. The LS-DYNA executable can be started within a debugger, for example, “gdb”, as

```
gdb mppdyna
```

It needs to be pointed out that the dynamic library is not loaded until the LS-DYNA executable is executed. So the breakpoints within the dynamic library may not be set in some systems. If so, the following option needs to be set within “gdb”,

```
set breakpoint pending on
```

which indicates that all pending breakpoints should automatically set when the dynamic library is loaded. Then the breakpoint can set for the dynamic library before the LS-DYNA is started, for example,

```
break umat41
```

The MODULE Feature

The MODULE-enabled LS-DYNA executable does not need any dynamic libraries to run a regular job, if no user subroutine is used in the model. It can be used as a standard release version of the LS-DYNA executable.

If one dynamic library is required, only the keyword *MODULE_LOAD needs to be added to the input deck. As shown in Figure 1, the keyword defines a unique id within the current input deck of the dynamic library to be loaded. The title is just for information purposes. The existing user subroutines prior to R9.0 can be built into a dynamic library, instead of a customized executable. It can be loaded by the MODULE-enabled LS-DYNA executable through keyword *MODULE_LOAD. The dynamic library can be used without recompilation if the LS-DYNA executable is updated, as they are completely decoupled. The user may put all dynamic libraries in a common shared folder, and use the keyword *MODULE_PATH to specify this folder. So the LS-DYNA executable is able to load the required dynamic libraries from there.

Card Sets. Repeat as many sets data cards as desired (cards 1 and 2) to load multiple libraries. This input ends at the next keyword ("*") card.

Card 1	1	2	3	4	5	6	7	8
Variable	MDLID		TITLE					
Type	A20		A60					
Default	none		none					

Card 2	1	2	3	4	5	6	7	8
Variable	FILENAME							
Type	C							
Default	none							

Figure 1. keyword *MODULE_LOAD

One of the most important uses of the MODULE is to load multiple user dynamic libraries. For example, three dynamic libraries are required within one job, one is developed and compiled in house and other two are provided by the material vendors without the source code. Keyword *MODULE_LOAD can load all libraries at once, as

```
*MODULE_LOAD
my_mod
libusermat_105657.so
mod_a
/vendor/lib_vendorA.so
mod_b
/vendor/lib_vendorB.so
```

in which three unique ids are defined for three dynamic libraries, as “my_mod”, “mod_a”, and “mod_b”. Let us assume that the following user material models are implemented in each dynamic library,

```
my_mod contains UMAT41
mod_a contains UMAT41, UMAT42, UMAT43
mod_b contains UMAT41, UMAT42, UMAT45, UMAT46
```

Case 1:

For a simple case, the model needs UMAT41 in “my_mod”, UMAT42 in “mod_a”, and UMAT45 & UMAT46 in “mod_b”. In other words, UMAT41 defined in the model should be mapped to “my_mod”,

and UMAT42 to “mod_a”, and UMAT45 & UMAT46 to “mod_b”. One may define the rules with the use of keyword *MODULE_USE as shown in Figure 2, in a straight forward manner, as

```
*MODULE_USE
my_mod
UMAT, 41, 41
*MODULE_USE
mod_a
UMAT, 42, 42
*MODULE_USE
mod_b
UMAT, 45, 45
UMAT, 46, 46
```

It does not require any changes to the existing input deck except adding the MODULE keywords.

The rules defined in *MODULE_USE are applied to only one dynamic library.

Card 1	1	2	3	4	5	6	7	8
Variable	MDLID							
Type	A20							
Default	none							

Rule Cards. Card 2 defines rules for the module specified in Card 1. Repeat as many cards as needed if defining multiple rules. New rules override existing rules, in case of a conflict. Input ends at the next keyword (“*”) card.

Card 2	1	2	3	4	5	6	7	8
Variable	TYPE		PARAM1		PARAM2			
Type	A20		A20		A20			
Default	none		blank		blank			

Figure 2. keyword *MODULE_USE

Case 2:

For another simple case, the model needs UMAT41 from all three libraries. UMAT41 defined in the model should be mapped to UMAT41 in “my_mod”, UMAT42 to UMAT41 in “mod_a”, and UMAT43 to UMAT41 in “mod_b”. The corresponding rules can be defined as

```
*MODULE_USE
my_mod
UMAT, 41, 41
```

```
*MODULE_USE
mod_a
UMAT, 42, 41
*MODULE_USE
mod_b
UMAT, 43, 41
```

Case 3:

As a general practice, one may use virtual material model IDs to define the rules. With LS-DYNA, the material IDs from 1001 to 2000 can be used as UMAT TYPE in the user material card, besides ten reserved IDs from 41 to 50 (referring to *MAT_USER_DEFINED_MATERIAL_MODELS). The virtual material IDs can be assigned to the dynamic libraries from the vendors. For example,

UMAT IDs 1201~1210 are assigned to UMAT41~50 in “mod_a”

UMAT IDs 1211~1220 are assigned to UMAT41~50 in “mod_b”

Then the corresponding user material models can always be used through their virtual material IDs. For the example defined in Case 2, the rules can be defined as

```
*MODULE_USE
my_mod
UMAT, 41, 41
*MODULE_USE
mod_a
UMAT, 1201, 41
*MODULE_USE
mod_b
UMAT, 1211, 41
```

in which UMAT41 in “mod_a” is always used as UMAT1201, and UMAT41 in “mod_b” as UMAT1211.

Some New Keywords for User Subroutines

Two new keywords are under development to support user subroutines. The first one is keyword *USER_PARAMETER. Some user parameters/data can be provided in the input deck for the user subroutines to access during runtime. These data can be integers, real numbers, or multiple line text, with a unique label. The LS-DYNA executable reads and saves the data into memory as is. The use of the data is solely dependent on the implementation of the user subroutines. Some typical uses may include extra parameters for the material models, version control, settings, and so on.

Another keyword is *USER_KEYWORD. Once the LS-DYNA keyword reader reaches this keyword, the keyword reader provided in the dynamic library will continue to read the input deck and generate the dynamic keywords for the LS-DYNA keyword reader. It means that the user keyword reader can read its own keywords and create models during runtime. Such models can be generated from the parameters read from the input deck, loaded from the central database, or any other sources.

Some Remarks

The new framework is under development. Its MPP version is available in R9.0 for the Linux system. The new framework will strive to maintain backward compatibility. The dynamic libraries compiled for R9.0 will be loaded and used by the later LS-DYNA MODULE-enabled executables, as in its binary form. It becomes easier to maintain the user subroutines which become a part of the input deck.

Appendix A: A sample input deck

```
*KEYWORD
*TITLE
$#                                                                         title
UMAT development
*MODULE_LOAD
myumat41
demo.so
*CONTROL_TERMINATION
$#  endtim    endcyc     dtmin    endeng    endmas
  1.000000         0     0.000     0.000 1.0000E+8
*DATABASE_BINARY_D3PLOT
$#      dt      lcdt      beam     npltc    psetid
    1.0E-2         0         0         0         0
*BOUNDARY_PRESCRIBED_MOTION_SET
$#    nsid       dof       vad      lcid        sf       vid     death     birth
         1         1         0         1       1.0         0 1.0000E28       0.0
*SET_NODE_LIST_TITLE
x=L
$#     sid       da1       da2       da3       da4    solver
         1       0.0       0.0       0.0       0.0MECH
$#    nid1      nid2      nid3      nid4      nid5      nid6      nid7      nid8
         2         4         6         8         0         0         0         0
*BOUNDARY_SPC_SET
$#    nsid       cid      dofx      dofy      dofz     dofrx     dofry     dofrz
         2         0         1         0         0         0         0         0
*SET_NODE_LIST_TITLE
x=0
$#     sid       da1       da2       da3       da4    solver
         2       0.0       0.0       0.0       0.0MECH
$#    nid1      nid2      nid3      nid4      nid5      nid6      nid7      nid8
         1         3         5         7         0         0         0         0
*BOUNDARY_SPC_SET
$#    nsid       cid      dofx      dofy      dofz     dofrx     dofry     dofrz
         3         0         0         1         1         0         0         0
*SET_NODE_LIST_TITLE
all
$#     sid       da1       da2       da3       da4    solver
         3       0.0       0.0       0.0       0.0MECH
$#    nid1      nid2      nid3      nid4      nid5      nid6      nid7      nid8
         1         2         3         4         5         6         7         8
*PART
$#                                                                         title
one-element test
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         1         1         1         0         0         0         0         0
*SECTION_SOLID
$#   secid    elform       aet
         1         1         0
*MAT_USER_DEFINED_MATERIAL_MODELS
$#     mid        ro        mt       lmc       nhv    iortho     ibulk        ig
         1    7800.0        41         4         0         0         3         4
```

```
$#   ivect     ifail    itherm    ihyper      ieos      lmca    unused    unused
         0         0         0         0         0         0
$#      p1        p2        p3        p4        p5        p6        p7        p8
    1.5E11      0.25    1.0E11      6E10
*DEFINE_CURVE_TITLE
X-velocity
$#    lcid      sidr       sfa       sfo      offa      offo    dattyp     lcint
         1         0       1.0       1.0       0.0       0.0         0         0
$#               a1                  o1
                0.0                 1.0
                1.0                 1.0
*ELEMENT_SOLID
$#   eid     pid      n1      n2      n3      n4      n5      n6      n7      n8
       1       1       1       2       4       3       5       6       8       7
*NODE
$#   nid               x               y               z      tc      rc
       1             0.0             0.0             0.0       0       0
       2             2.0             0.0             0.0       0       0
       3             0.0             1.0             0.0       0       0
       4             2.0             1.0             0.0       0       0
       5             0.0             0.0             1.0       0       0
       6             2.0             0.0             1.0       0       0
       7             0.0             1.0             1.0       0       0
       8             2.0             1.0             1.0       0       0
*MAT_ELASTIC
$#     mid        ro         e        pr        da        db  not used
        11    7800.0    1.5E11      0.25       0.0       0.0         0
*END
```

Thermal Coupling Method Between SPH Particles and Solid Elements in LS-DYNA

Jingxiao Xu, Jason Wang

LSTC

INTRODUCTION:

Heat transfer is very important in many industrial and geophysical problems. Many of the problems of geophysical and industrial fluid dynamics involve complex flows of multiple liquids and gases coupled with heat transfer. The motion of the surfaces of the liquids can involve sloshing, splashing and fragmentation. Thermal and chemical processes present further complications. The simulation of such systems can sometimes present difficulties for finite difference and finite element methods, particularly when coupled with complex free surface motion, while smoothed particle hydrodynamics can easily follow wave breaking, and it provides a reasonable simulation of splash on a length scale exceeding that where surface tension must be included.

SPH is a Lagrangian method for solving partial differential equations. Essentially, the domain is discretized by approximating it by a series of roughly equi-spaced particles. They move and change their properties (such as temperature) in accordance with a set of ordinary differential equations derived from the original governing PDEs. Cleary and Monaghan (1995) extended the method to heat conduction and then to coupled heat and mass flows due to that SPH has a range of strong advantages in modeling industrial heat and mass flows:

1. The Lagrangian framework allows momentum dominated flows to be easily handled.
2. Complex free surface and material interface behavior, including break-up into fragments, can be modeled naturally.
3. Complicated physics such as multi-phase, realistic equations of state, compressibility, radiation, solidification and fracturing can be added with comparative ease.

For the thermal coupling between any two parts in LS-DYNA, the standard way is through thermal contacts which require the contact areas between those two parts. Due to particle property, SPH particles can handle extremely large deformations, particles can be moved without limitation. In real engineering applications, SPH particles may have very complex free surface and material interface behaviors, including break-up into fragments, and new surfaces will be generated automatically every cycle when interacting with Solid elements. It is quite difficult to update new contact surfaces and calculate the true contact areas between SPH particles and Solid elements. Here we introduce a new thermal coupling method between SPH particles and Solid elements through keyword *DEFINE_ADAPTIVE_SOLID_TO_SPH with icpl=3 and iopt=0 options without using thermal contacts.

KEYWORD IN LS-DYNA AND APPLICATIONS:

Keyword *DEFINE_ADAPTIVE_SOLID_TO_SPH is used to adaptively transform a Lagrangian solid Part or Part Set to SPH particles, when the Lagrangian solid elements comprising those parts fail (Shown in Fig 1), or used as hybrid elements to couple between original SPH parts and Solid parts. One or more SPH particles (elements) will be generated for each failed element. The SPH particles replacing the failed

element inherit all of the properties of the failed solid element, e.g. mass, kinematic variables, and constitutive properties.

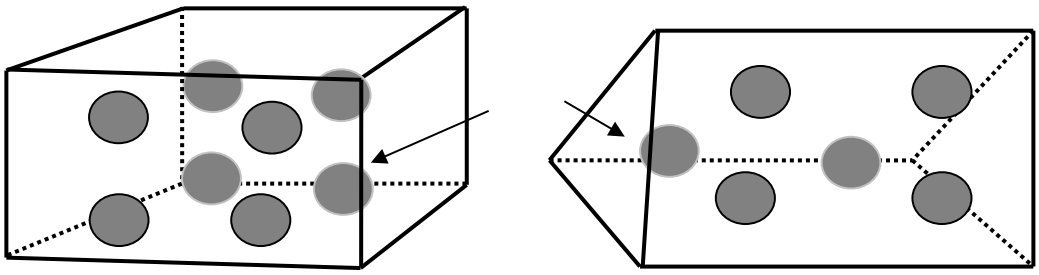


Fig 1. Transform Solid elements into SPH particles

With ICPL=0, this keyword is used for debris simulation, no coupling happens between newly generated SPH particles and solid elements, user need to define node to surface contact for the interaction between those two parts. When ICPL=1 and IOPT=1, the newly generated SPH particles are bonded with solid elements as one part through the coupling (Hybrid elements).

With ICPL=1 and IOPT=0, this keyword is used as Hybrid Elements coupling SPH with Solid. Hybrid elements are used as transit layers between SPH particles and Solid elements, for a portion of grid model comprises SPH particles because the likelihood of enduring large deformation, while the rest of the model comprises FEM solid elements, hybrid elements are placed between the solids and the particles, each hybrid element comprises two layers: solid layer and particle layer.

A new function was introduced into this keyword for the pure thermal coupling between SPH particles and Solid elements with ICPL=3 and IOPT=0. In this function hybrid elements were used, we have the SPH formulation and at the same time we have the Solid meshes which clearly describe the material interfaces. Solid elements constrain SPH nodal locations. SPH particles here provide real thermal coupling between original SPH parts and Solid element parts, also a coupling conductivity parameter between SPH part and Solid part was introduced here in the 8th parameter of the keyword input *DEFINE_ADAPTIVE_SOLID_TO_SPH (as shown in Fig 4).

EXAMPLES

Case 1:

A simple 3D pure thermal conduction example was tested here. For both SPH and Solid parts, initial temperature conditions were set to 0.0. On the surface nodes of the right side of the SPH part, a constant thermal BC was set (as shown in Fig 2). The thermal coupling between SPH and Solid part was applied through keyword *DEFINE_ADAPTIVE_SOLID_TO_SPH with icpl=3 and iopt=0. The default coupling conductivity between SPH and Solid parts was set (average value of the conductivities of two parts). Fig 3 shows the temperature results from the thermal conduction and the thermal coupling between SPH part and Solid part.

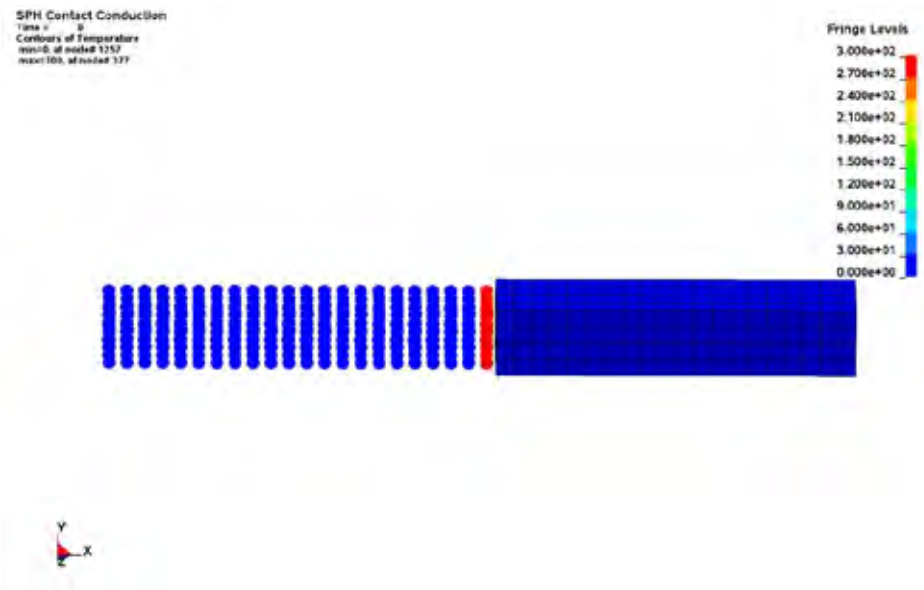


Fig 2. Initial set up for thermal coupling between SPH part and Solid part

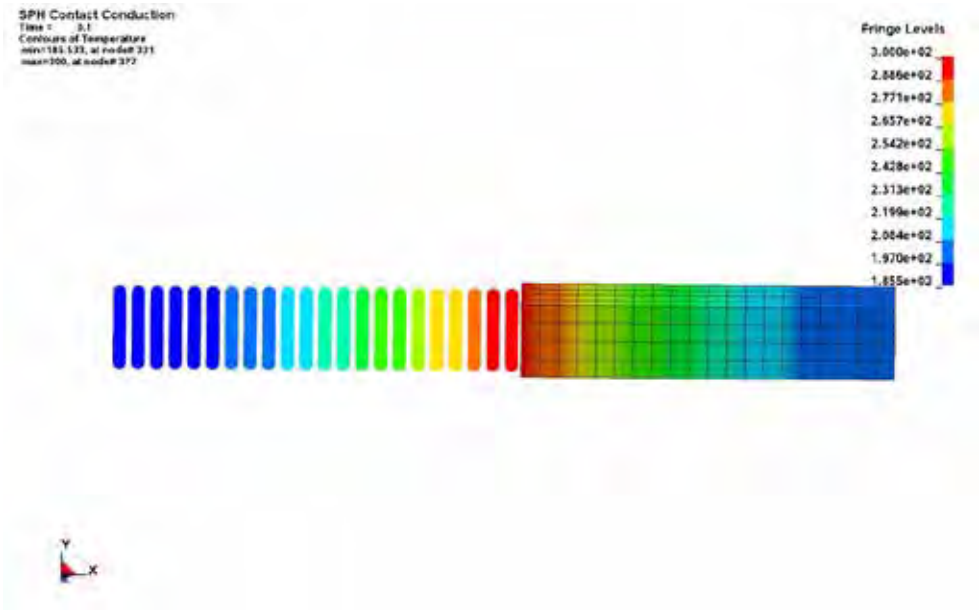


Fig 3. Temperature contour plot for thermal coupling between SPH part and Solid part

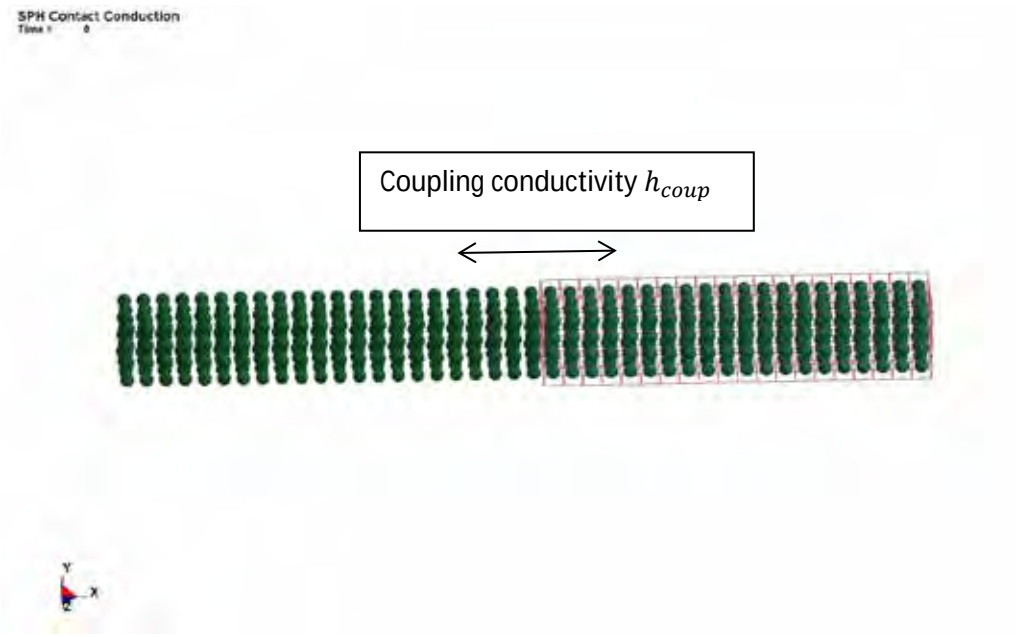


Fig 4. Hybrid element set up for thermal coupling between SPH part and Solid part

Case 2:

A more complicated high velocity impact problem was tested in this case. A SPH part hit a thin Solid plate with very high initial speed. During the impacting process, SPH part endured very complex free surface and material interface behaviors, including break-up into fragments and new surfaces generation every cycle when interacting with Solid plate. For the structure behavior, a node to surface contact was used for the interaction between SPH part and Solid plate. The temperature of the SPH body changed due to the conversion of mechanical work into heat through plastic deformation. The process was fast enough such that there is no heat transfer with the environment. Thermal coupling between SPH part and Solid plate was applied through *DEFINE_ADAPTIVE_SOLID_TO_SPH (Fig 6 shows the hybrid elements setup for this coupling) with ICPL=3 and IOPT=0, the default coupling thermal conductivity was set for the thermal interaction between SPH part and Solid plate. Fig 5 shows the temperature contour plot during thermal coupling processing for HVI problem between SPH part and Solid plate.

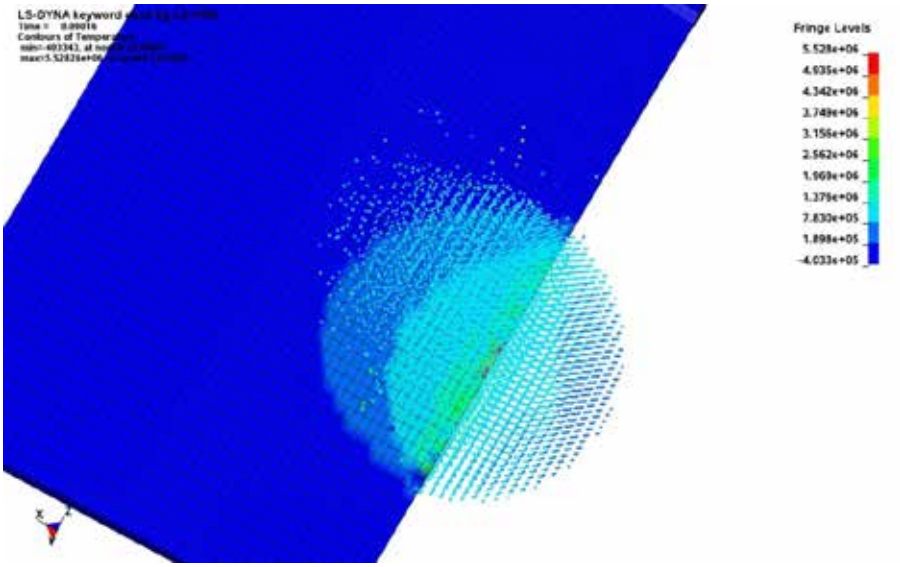


Fig 5. Temperature contour plot for HVI between Solid plate and SPH part with thermal coupling.

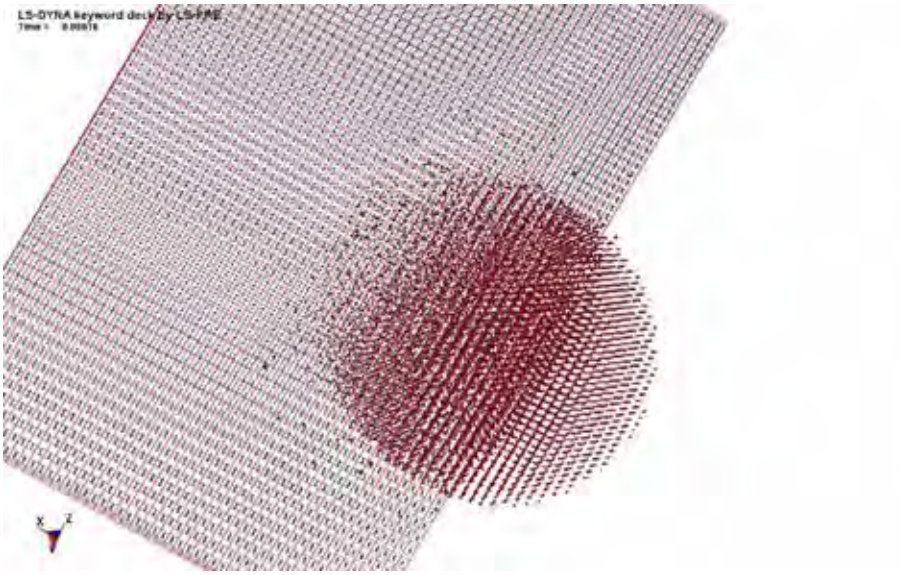


Fig 6. Hybrid element set up for HVI between SPH part and Solid plate with thermal coupling.

FUTURE WORKS

A more sophisticated coupling thermal conductivity model between SPH part and Solid part is needed for the further application of this keyword to the more complicated engeneering behaviors.