fea INFORMATION Engineering Solutions

www.lstc.com

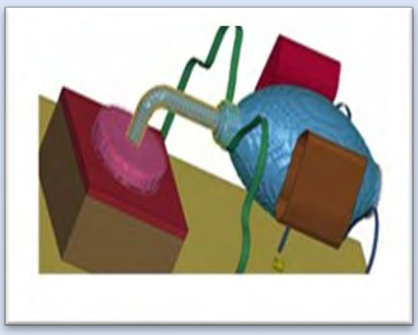
Ansys LST

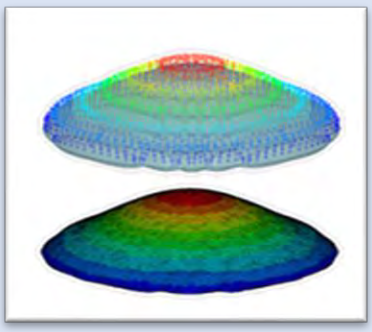
www.ansys.com

LS-DYNA® Structured ALE

Thematic Series Collection

1. On Setting up a Structured ALE Model
2. On Setting up Multi-Materials in S-ALE Models
3. On Setting up Boundary and Initial Conditions in S-ALE Models
4. On Setting up a 2D Structured ALE Model
5. On Setting up the Structured ALE Mesh
6. On Structured ALE Mesh Trimming





On Setting up a Structured ALE Model

Hao Chen, Ansys

LS-DYNA@ ALE has been widely used to simulating moving fluids interacting with structures. Unlike CFD, the focus is rather on the structure response under dynamic loading from fluids, than the fluids' motion. Fluids are agitated by a high pressure gradient; and then hit the structure, carrying a large momentum. The key in successfully capturing the physics lies in the fluid-structure interaction algorithm. It needs to accurately predict the peak of pressure loading during the impact, which is characterized as a momentum transfer process. This request could only be fulfilled by a transient analysis with a penalty-based coupling between fluids and structure.

In 2015, LSTC introduced a new structured ALE (S-ALE) solver option dedicated to solve the subset of ALE problems where a structured mesh is appropriate. As expected, recognizing the logical regularity of the mesh brought a reduced simulation time for the case of identical structured and unstructured mesh definitions. It also comes with a cleaner, conceptually simpler way of model setup. This article gives a brief description of the S-ALE model setup.

PART is the problem.

This section is to give a background information on the introduction of a new keyword *ALE_STRUCTURED_MULTI-MATERIAL_GROUP. For new users never used previous ALE/S-ALE setup, please skip this section as it is irrelevant and could be confusing without knowing some history of changes in ALE setups.

"Part" definition is conveniently used in Finite element models to link the material definition and mesh of a Lagrange structure. A typically "PART" definition contains three things: SECTION, which is the integration rule; MAT+EOS, the material property; and mesh, linked by the PARTID and listed under *ELEMENT keyword. In a Lagrange simulation, "PART" has dual meanings – it refers to both the mesh and the material.

In the world of ALE, it is a little more complicated. As we are dealing with fluids, our point of view is rather Eulerian, not Lagrangian. This means the mesh and materials are not; and should not be bundled together. Mesh is no longer a spatial representation of material; and its boundary surface is no long material interface. Rather, the mesh, in an ALE simulation, is simply a spatial domain, to provide room for the fluids to occupy and flow. They are multiple fluids inside this mesh and associate this mesh with any one material property does not make any sense.

The general ALE solver borrowed "PART" definition. This caused quite some confusion in our users, even the most experienced users. It is not surprising at all, as sometimes "PART" refers

the mesh; other times the material; and on several occasions, it refers to both mesh and materials.

When constructing the Structured ALE keywords, the author tried to separate this dual meanings "PART" definition into two distinguished definitions – "Mesh Part" and "Material Part". It helped to some extent, but still considerable confusion still exists. As S-ALE solver is gradually picking up more usage, it becomes mandate to address this issue once and for all. Streamlining the S-ALE setup would save a lot of users' effort, especially for our new users not familiar with LS-DYNA keyword setups.

A new keyword, *ALE_STRUCTURED_MULTI-MATERIAL_GROUP was introduced recently. It is available in the latest beta version executable and will be in the next R12.1 release. This keyword no longer uses "PART" to link the material properties and hence eliminates the concept of "Material Part". The author believes it would prevent most common user setup mistakes.

Three step setup

We follow a straight-forward three step setup. First, mesh; secondly, material properties of fluids; thirdly, filling the mesh with fluids. In this section, we describe the three keywords doing these three steps.

1. Mesh generation: *ALE_STRUCTURED_MESH; *ALE_STRUCTURED_MESH_CONTROL_POINTS

In S-ALE, mesh is always rectangular. Obviously an automatically generated mesh would get rid of lots of unnecessary hassles in the model building and execution. So that was the route we took. To determine the mesh layout in the space, we need the following information:

a. The mesh spacing along three axes (LCIDX,LCIDY,LCIDZ).
b. The origin (NID0), and local coordinate system (LCSID).

Other fields are for identification purpose only and are self-explanatory. MSHID stands for mesh ID; DPID Part ID; NBID and EBID are the IDs of first S-ALE mesh node and element, respectively. TDEATH is to the "death time" for S-ALE mesh. It is to turn off the S-ALE calculation once the most of fluid loading is applied; and keep the Lagrange model running as the structure deformation is not fully developed yet.

*ALE_STRUCTURED_MESH							
MSHID	DPID	NBID	EBID				TDEATH
CPIDX	CPIDY	CPIDZ	NID0	LCSID			

The LCIDX, LCIDY and LCIDZ are IDs of *ALE_STRUCTURED_MESH_CONTROL_POINTS cards. Each card specifies the mesh spacing along one axis.

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
CPID							

N1	X1	RATIO1		
…	…	…		
Nn	Xn	RATIOn		

The idea of the "_CONTROL_POINTS" card is simple. Let us forget about the "RATIO" field which is for progressive mesh spacing. This card contains some number of (N,X) pairs. And each (N,X) pair means "the Nth node's coordinate is X". Between any two consecutive (N,X) pairs mesh is evenly distributed. Take the simplest case, say 10 elements between [0,1]. A simple two pairs setup of (1,0.) and (11,1.) is sufficient. To another extreme, say we really want some insane irregular mesh. We could make 11 pairs of (N,X) like (1,0.), (2,0.07), (3,0.13), (4,0.2), (5,0.26), (6,0.37), (7,0.47), (8,0.53), (9,0.79), (10,0.8), and finally (11,1.).

The progressive mesh spacing is something worth a separate article by itself and hence skipped in this introductory paper. It is powerful but conceptually not so user-friendly. Efforts are being continuously made to improve this part. Please stay tuned for the updates.

2. Material definitions: *ALE_STRUCTURED_MULTI-MATERIAL_GROUP

S-ALE mesh is simply a spatial domain in which fluids flow. In order to let the code know what and how many fluids there are, we need to provide material properties of each fluid and list them all under the card *ALE_STRUCTURED_MULTI-MATERIAL_GROUP. Please note, AMMG stands for ALE multi-material group, a rather alternative and maybe confusing name for "ALE fluid". In this paper we are going to use AMMG and fluid interchangeably.

*ALE_STRUCTURED_MULTI-MATERIAL_GROUP							
AMMGNM1	MID1	EOSID1					PREF1
…	…	…					…
AMMGNMn	MIDn	EOSIDn					PREFn

"AMMGNM" is a name one gives to a AMMG (ALE Multi-Material Group), aka ALE fluid. It is used in other cards, for example, *SET_MULTI-MATERIAL_GROUP_LIST to refer to that AMMG. "MID" and "EOSID" are the material ID and EOS ID, respectively.

"PREF" is to describe the reference pressure or "base pressure" of that fluid. This might be somewhat new to our typical users from solids background. Pressure of a solid material, if not preloaded, always starts from zero. In such case, its reference pressure or base pressure, is zero. But most fluids have non-zero reference pressure. For example, air has a base pressure of 101325 Pa (1 bar atmospheric pressure). Traditionally this reference pressure is prescribed using the field "PREF" in *CONTROL_ALE card. The new *ALE_STRUCTURED_MULTI-MATERIAL_GROUP has a design to allow each AMMG to have its own reference pressure. The author believes this added flexibility could be proven very useful in certain applications.

3. Volume Filling: *ALE_STRUCTURED_MESH_VOLUME_FILLING

Now we have a rectilinear S-ALE mesh, which is our calculation domain. And we have several fluids which resides in this domain. Before carrying out our simulation, there is still one critical information missing. That is: how are those fluids occupying the domain? This information is given as the form of volume fraction per element. Basically we need to assign the volume fraction for each AMMG (ALE fluid) for each element. Say one element is fully occupied by AMMG "water", then the volume fraction of that element is (1, 0, 0), assuming the ALE materials are ("water", "air", "vacuum"). Similarly (0.6,0.3,0.1) if "water" 60%, air 30% and vacuum 10%. For each element, the code needs to know the volume fraction of all ALE fluids.

The first way is to explicitly list volume fractions for each element. This could be done through a keyword card called "*INITIAL_VOLUME_FRACTION". This approach is seldomly used as first, it is tedious and secondly, too much burden on users.

The much easier solution is to automatically generate the volume fraction information based on certain user supplied geometries. These geometries could be simple shapes like sphere, plane, box, cylinder. Or it could be user-defined complex shape like structure surfaces. With certain geometric shape defined, users could then fill the inside or outside of that shape with certain ALE fluid. The volume fractions are internally calculated, stored and then used in the subsequent simulation to reconstruct the fluid interface.

This is done by using the keyword "*ALE_STRUCTURED_MESH_VOLUME_FILLING". The "volume filling" process typically is done through multiple "tasks", each task by a separate keyword.

*ALE_STRUCTURED_ MESH_VOLUME_FILLING							
MSHID		AMMGTO		NSAMPLE			VID
GEOM	IN/OUT	E1	E2	E3	E4		

MSHID is the S-ALE mesh ID, defined in *ALE_STRUCTURED_MESH card. And AMMGTO is the name of ALE fluid to be filled, defined in *ALE_STRCUTURED_MULTI-MATERIAL_GROUP. VID is to prescribe the initial velocity of that fluid, if any. And NSAMPLE is default to 3 which means that one ALE cell is divided into 7x7x7 (7=2*3+1) sub-cells and each sub-cell is checked to see if it is inside/outside.

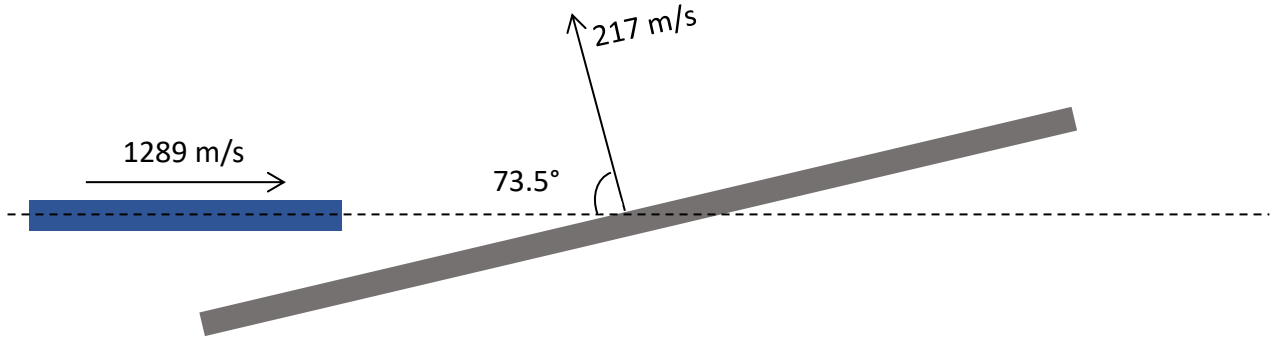
It supports 5 basic geometries: Plane, Cylinder, Ellipsoid, Box with indices and Box with coordinates. And E1-E4 are used to provide information of these geometries. For complicated geometries, we need users to provide us with a segment set (or something we could internally convert to a segment set). The assumption is that all segment normals are consistent; and those normals point to the "inside". For convenience, we provide a "IN/OUT" flag for an easier flip.

A Simple Example

Now let us use a simple example to illustrate this 3-step process. It is to model a long rod projectile impacting an oblique steel plate (Fugelso & Taylor 1978). The dimensions were from

ARL-TR-2173 (Schraml & Kimsey 2000) and material properties from "Numerical Simulation of High-Velocity oblique Impacts of Yawed Long Rod Projectile Against Thin-Plate" (Yo-Han Yoo 2002).

Below is a sketch showing a projectile at 1289m/s and hitting a plate which is moving up at 217 m/s. Projectile has a length of 76.7mm and a diameter of 7.67mm. Plate has a thickness of 6.4 mm, a width of 60mm and a length of 150mm. The unit used is mm-g-s.



Step 1: To construct a mesh spans (-107.5,-30,-15) to (107.5,30,15) with 215x60x30 elements.

```
*ALE_STRUCTURED_MESH
$    mshid       pid      nbid      ebid
         1        11    100001    100001
$    cpidx     cpidy     cpidz      nid0     lcsid
      1001      1002      1003
*ALE_STRUCTURED_MESH_CONTROL_POINTS
$     cpid
      1001
$                  N                   X
                   1           -107.5000
                 216            107.5000
*ALE_STRUCTURED_MESH_CONTROL_POINTS
$     cpid
      1002
$                  N                   X
                   1             -30.000
                  61              30.000
*ALE_STRUCTURED_MESH_CONTROL_POINTS
$     cpid
      1003
$                  N                   X
                   1            -15.0000
                  31             15.0000
```

Step 2: Set up ALE multi-material Groups (AMMGs or ALE fluids). There are totally 3 AMMGs defined. First is "Rod" by *MAT_JOHNSON_COOK (MID=1) and *EOS_LINEAR_POLYNOMIAL (EOSID=1). Next is "Vacuum" by *MAT_VACUUM (MID=3). The third is "Plate" by

*MAT_JOHNSON_COOK (MID=2) and *EOS_LINEAR_POLYNOMIAL (EOSID=2). Their materials properties are given as follows.

```
*MAT_JOHNSON_COOK
$      MID        RO         G         E        PR       DTF        VP    RATEOP
         1   18.6e-3  63.692e3   165.6e3       0.3
$        A         B         N         C         M        TM        TR      EPS0
   1.079e3    1.12e3      0.25    0.0070       1.0      1473       283   1.0e-3
$       CP        PC     SPALL        IT        D1        D2        D3        D4
     130.0
$       D5      C2/P      EROD     EFMIN    NUMINT

*EOS_LINEAR_POLYNOMIAL
$    eosid        c0        c1        c2        c3        c4        c5        c6
         1     0.000  138.00e3
$       e0        v0
     0.000     0.000
*MAT_JOHNSON_COOK
$      MID        RO         G         E        PR       DTF        VP    RATEOP
         2   7.87e-3  76.692e3  200.1e3       0.3
$        A         B         N         C         M        TM        TR      EPS0
   0.792e3   0.510e3      0.26    0.0140      1.03      1809       283   1.0e-3
$       CP        PC     SPALL        IT        D1        D2        D3        D4
     480.0
$       D5      C2/P      EROD     EFMIN    NUMINT

*EOS_LINEAR_POLYNOMIAL
$    eosid        c0        c1        c2        c3        c4        c5        c6
         2     0.000  166.75e3
$       e0        v0
     0.000     0.000
*MAT_VACUUM
$      MID
         3    1.0e-9
```

Now construct the ALE fluids by using *ALE_STRUCTURED_MULTI-MATERIAL_GROUP.

```
*ALE_STRUCTURED_MULTI-MATERIAL_GROUP
$     name       mid     eosid
pref
      rod          1         1
   vacuum          3
    plate          2         2
```

Step 3: Volume Filling the initial S-ALE mesh. By *ALE_STRUCTRUED_MESH_VOLUME_FILLING.

First, fill all with “vacuum”.

```
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$#   mshid         -    ammgto         -   nsample         -         -     vid
         1              vacuum
$#    geom    in/out
ALL               0
```

Next, assign the inside of box (BOXID=1) to “plate”.

```
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$#   mshid         -    ammgto         -   nsample         -         -       vid
         1               plate                                              1
$#    geom    in/out     boxid
BOXCOR              0         1
*DEFINE_BOX_LOCAL
$#   boxid       xmn       xmx       ymn       ymx       zmn       zmx
         1       0.0     150.0       0.0       6.4       0.0        30
$#      xx        yx        zx        xv        yv        zv
   0.95882   0.28402       0.0  -0.28402   0.95882       0.0
$#      cx        cy        cz
  -71.0026  -24.3694     -15.0
```

Finally, assign the inside of a cylinder to “rod”.

```
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$#   mshid         -    ammgto         -   nsample         -         -       vid
         1                 rod                                              2
$#    geom    in/out      nid1      nid2    radii1    radii2
CYLINDER            0         1         2     3.835     3.835
*NODE
$#   nid               x               y               z      tc      rc
       1          -103.0             0.0             0.0       0       0
       2          -26.33             0.0             0.0       0       0
```

And the initial velocities of “plate” and “rod” are prescribed by using the following cards.

```
*DEFINE_VECTOR
$#     vid        xt        yt        zt        xh        yh        zh       cid
         1   -61.631    208.06       0.0       0.0       0.0       0.0         0
         2    1289.0       0.0       0.0       0.0       0.0       0.0         0
```

That is all! But before we could make it run, we need to add in some more control card and database cards. We could use *CONTROL_ALE to make a choice of order of accuracy, 1st order (donor cell) or 2nd order (van Leer).

```
*CONTROL_ALE
$#     dct      nadv      meth      afac      bfac      cfac      dfac      efac
                            1
$#   start       end     aafac     vfact      prit       ebc      pref   nsidebc
```

The rest of fields should be left as blank most of the time, if not all. They are used in generic ALE solvers but mostly ignored in S-ALE solvers with only a few exceptions.

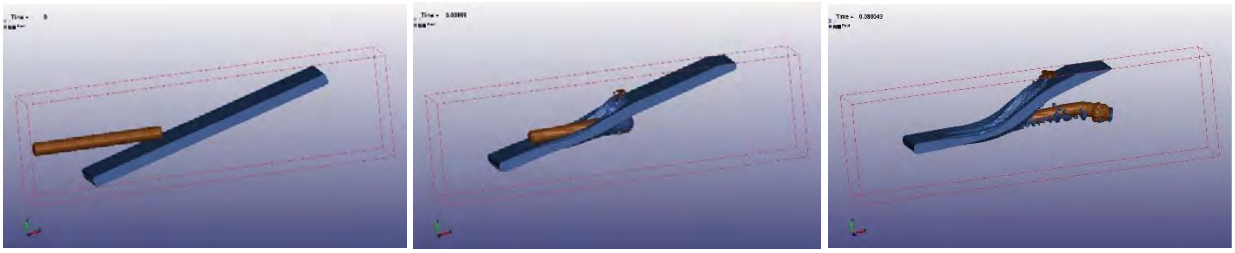
And other cards:

```
*CONTROL_TERMINATION
$#  endtim    endcyc     dtmin    endeng    endmas
 0.0800000
*CONTROL_TIMESTEP
```

```
$#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode     ms1st
     0.000  0.600000
$#  dt2msf   dt2mslc     imscl
     0.000         0         0
*DATABASE_BINARY_D3PLOT
$#      dt      lcdt      beam     npltc    psetid
  0.001000
```

Now run it with 16 core MPP executable. It will only take 39 seconds. Please note as not all S-ALE keywords are finalized until recently (Jan 2021), we need to use the latest DEV version of LS-DYNA. R12.1 version should be available in the first half of year 2021.

Below is the projectile and plate at t=0., t=0.04ms and t=0.08ms. The input deck is available at https://ftp.lstc.com/anonymous/outgoing/hao/sale/models_R121/cthrod.k



Ending Remarks

LS-DYNA ALE module has been known for its steep learning curve. Partially it was because setting up Eulerian models are intrinsically different from Lagrange models. But the design of ALE keyword cards, for sure, has caused quite a lot of confusions among our users, new and experienced.

To prompt LS-DYNA ALE usages, Structured ALE solver introduced a new, user-friendly, streamlined three-step setup. We hope this effort could help users, new or old, to perform their work more efficiently and smoothly.

On Setting up Multi-Materials in S-ALE Models

Hao Chen, Ansys

LS-DYNA ALE has been widely used to simulating moving fluids interacting with structures. Unlike CFD, the focus is rather on the structure response under dynamic loading from fluids, than the fluids' motion. Fluids are agitated by a high pressure gradient; and then hit the structure, carrying a large momentum. The key in successfully capturing the physics lies in the fluid-structure interaction algorithm. It needs to accurately predict the peak of pressure loading during the impact, which is characterized as a momentum transfer process. This request could only be fulfilled by a transient analysis with a penalty-based coupling between fluids and structure.

In 2015, LSTC introduced a new structured ALE (S-ALE) solver option dedicated to solve the subset of ALE problems where a structured mesh is appropriate. As expected, recognizing the logical regularity of the mesh brought a reduced simulation time for the case of identical structured and unstructured mesh definitions. It also comes with a cleaner, conceptually simpler way of model setup. This article gives a brief description on setting up the S-ALE multi-materials.

Multi-materials.

ALE Multi-materials, or ALE multi-material groups (AMMG), has been used throughout LS-DYNA user manuals and other documentations. This name, by itself, does not mean too much. It was just to follow the naming convention several decades old. The first generation ALE solver only supported one single material so it was called "single material" element formulation (ELEFORM=5 in *SECTION_SOLID). Then the concept of "volume fraction" was introduced and then a "single material with void" element formulation was implemented (ELEFORM=12). Based on the element volume fraction, we could use an interface reconstruction technique to regenerate material interface between that material and void.

Around the same time, the ALE developer found the same technique could be extended to deal with multiple materials flowing inside the ALE mesh. The idea is simple. In single material with void formulation, in each element we have a "volume fraction" which represents the ratio between the material volume and the total element volume. And later after advection, we use that information to reconstruct the material void interface in that element. Now we could simply allow for multiple materials; each has its own element volume fraction value; and each goes through the interface reconstruction process – to find the interface between this material and all other materials. We will cover that process a little bit more later as it is quite important to understand the process to set up the keyword right.

Then the multi-material element formulation (ELEFORM=11) was implemented. And these materials occupying the ALE mesh are referred as "ALE multi-materials". To the author, a more intuitive and shorted name "ALE fluids" might be more appropriate. In the following discussion, "ALE fluids" and "ALE multi-materials" are used interchangeably.

Multi-Material Setup in the Three step setup

S-ALE follows a straight-forward three step setup. First, mesh; secondly, material properties of fluids; thirdly, filling the mesh with fluids. For a detailed description of this 3-step setup, please refer to "On setting up a S-ALE model" published on FEA information Jan 2021 issue.

We set up the ALE fluids through two types of keywords: *MAT_+*EOS to provide material properties; and *ALE_STRUCTURED_MULTI_MATERIAL_GROUP to provide the list of ALE fluids and their order of interface reconstruction.

We will cover the material definitions in the next section and concentrate here on the card *ALE_STRUCTURED_MULTI_MATERIAL_GROUP. The card is of the following format.

*ALE_STRUCTURED_MULTI-MATERIAL_GROUP							
AMMGNM1	MID1	EOSID1					PREF1
…	…	…					…
AMMGNMn	MIDn	EOSIDn					PREFn

"MID" and "EOSID" are the material ID and EOS ID, respectively. *MAT_+*EOS_ provides the complete material properties of 1 ALE fluid. Almost ALE fluids need *EOS properties.

"AMMGNM" is a name one gives to a AMMG (ALE Multi-Material Group), aka ALE fluid. It is a character string and case insensitive. The leading and trailing spaces are trimmed so alignment is not an issue. We suggest short descriptive names such as "soil", "water", "air", "HE"; or sometimes "inAir", "outAir", "airBelow", "airAbove", etc. It will be cross-referenced in other keywords where AMMG needs to be specified. For example, *SET_MULTI-MATERIAL_GROUP_LIST.

"PREF" is to describe the reference pressure or "base pressure" of that fluid. This might be somewhat new to our typical users from solids background. Pressure of a solid material, if not preloaded, always starts from zero. In such case, its reference pressure or base pressure, is zero. But most fluids have non-zero reference pressure. For example, air has a base pressure of 101325 Pa (1 bar atmospheric pressure). Traditionally this reference pressure is prescribed using the field "PREF" in *CONTROL_ALE card. The new *ALE_STRUCTURED_MULTI-MATERIAL_GROUP has a design to allow each AMMG to have its own reference pressure. The idea of reference pressure will be covered more in the next section.

A key point, not well known among even the most experienced ALE users, is that the order of AMMG definitions matters. It plays an important role in the interface reconstruction. And a wrongfully ordered *ALE_STRUCTURED_MULTI_MATERIAL_GROUP card could lead to unphysical behaviors.

The interface reconstruction algorithm is designed to construct one interface between two fluids. It generates a plane cut inside an ALE element; puts one fluid on one side of that plane, another on the other side. When the volumes of these two polygons match the volumes of two fluids, we are all settled there. This algorithm needs to be extended to deal with more than two fluids. For two fluids, we generate one interface. Say three fluids, two interfaces. How should we proceed?

"Onion skin" was adapted. We do one interface at a time. First, we generate the interface between fluid #1 and other fluids. Next, between fluid #1+ fluid #2 and other fluids. Then fluid #1+fluid#2+fluid#3 and others, so on

and so forth. It is like peeling off the onion skin, first we peel #1 off, then #2, then #3. Only now we are constructing #1 first and then stack #2 up and then #3.

Apparently, in order for this process to make sense, an implied assumption needs to be made. That is, all these fluids listed next to each other must also be physically next to each other. Otherwise, the interface reconstruction will put fluids out of its real location.

For example, we have "air", "water", "soil" from up to bottom and we listed them in that order in *ALE_STRUCTURED_MULTI-MATERIAL_GROUP card. In the mixed cell contains all three fluids, we generate the "air" interface first. "Air" is now at the top. And then "air"+"water", so "water" now is below "air". And then "soil" is put at the bottom. Perfect, right? But how about if we list "air" on the first line, but switch the order between "water" and "soil"? Do you see the problem?

OK. First we do "air". "Air" is at the top. Next "air"+"soil" as "soil" is on the second line. Without knowing better, our interface construction algorithm puts "soil" next to "air" at the center. Then "water" at the bottom. We would suddenly find that after advection, there are some "water" mysteriously moved to the bottom of the element. WHY? The order matters.

Not surprisingly the combination of lack of information and limitation in the interface reconstruction algorithm led to lots of strange ALE results. I apologize for the trouble it has brought to our users. As we did not do well on passing this important piece of information effectively to our users. To all readers of this article, I would greatly appreciate if you could forward it to our fellow ALE users.

Material Properties

Most our users are from solid Mechanics background. Same for me. What we care is to apply the fluid loadings onto our structure. The structural response is what we are after. ALE is never intended to solve "CFD" kind of problems. Rather it is to convey a pressure wave through certain fluid(s) and then load it onto a structure.

But in order to do that well, we still need to understand a little bit about fluids. Majorly, how it is different to solids. The first thing here is Equation-of-State. Why almost always we have EOS definition in ALE fluids?

Let us start with internal energy. What is internal energy? Ask a structural engineer, he would say it is the energy deposited into the material by external load. A block free-resting on the ground has zero internal energy. And internal energy starts to increase when load is applied. But is this also true for fluids? Uncompressed fluid has zero internal energy? Take air, can we say it has zero energy when not compressed?

What us structural engineers referred to as "internal energy" is, in fact, "internal energy difference" or "change in internal energy". We pay no attention to the real internal energy of a block not loaded. What we have here is the external work = -internal energy change. The initial internal energy does not matter.

But in certain fluids, like air, we do have a "real" internal energy. And this internal energy contributes to the pressure value. And "EOS_" provides us the way to define the pressure dependence on the internal energy.

Typically in Solid Mechanics, *MAT_ itself is sufficient to describe the whole material behavior. A stress field is divided into dilatational and deviatoric. Dilatational linked to bulk modulus and deviatoric to shear modulus.

And in case of plasticity, dilatational we have no pressure change as it is incompressible. There is no internal energy dependency as it does not play any role in solids behavior.

Fluids are different. First, compression always leads to pressure change. (Please note as ALE is a hydrocode we do not allow incompressibility.) Secondly, pressure depends on internal energy. So for fluids, pressure is a function of compressibility and internal energy. And here comes the *EOS_ keywords to let users prescribe that function.

Take water as an example. Water, when not heavily loaded, could be assumed to be of a linear material with no internal energy dependence. *EOS_LINEAER_POLYNOMIAL (*EOS_01) with bulk modulus defined is sufficient.

```
*MAT_NULL
$#    mid      ro      pc      mu    terod    cerod      ym      pr
      1    998.0     0.0     0.0     0.0     0.0     0.0     0.0
*EOS_LINEAR_POLYNOMIAL
$#  eosid      c0      c1      c2      c3      c4      c5      c6
      1     0.0   2.2e9     0.0     0.0     0.0     0.0     0.0
$#     e0      v0
    0.0     1.0
```

Deviatoric stress only comes from viscous shear for water. This viscosity coefficient (mu) could be defined in *MAT_NULL. For pressure field, we have $p = c_1(\rho/\rho_0 - 1)$. All other polynomial coefficients c_i are zero. e_0 is the internal energy density (per unit volume); and $v_0 = \rho_0/\rho(t), t = 0$ is the compression ratio, at the initial state, respectively. Please note here ρ_0 is the uncompressed material density, not the initial material density. The above setup did two things. First, it defined material properties. Secondly, it also prescribed initial pressure to be 0. The later could be easily overlooked.

So what if we want to prescribe a non-zero initial pressure? For this water, it is simple. We back solve for the initial compression ratio. Say in an ALE model we have two fluids—air and water. To have a pressure equilibrium in the initial state, water needs to have a pressure of 1 bar=101,325 Pa, same as air. We let $v_0 = \rho_0/\rho(t) = \frac{1}{1+p/c_1} = \frac{1}{1+101325/2.2e9} = 0.9999539453.$

Now let us do air. Air is with idea gas law. We could either define it with *EOS_LINEAR_POLYNOMIAL (*EOS_01) or *EOS_IDEAL_GAS (*EOS_12). Let us use *EOS_01.

```
*MAT_NULL
$#    mid      ro      pc      mu    terod    cerod      ym      pr
      2    1.23     0.0     0.0     0.0     0.0     0.0     0.0
*EOS_LINEAR_POLYNOMIAL
$#  eosid      c0      c1      c2      c3      c4      c5      c6
      2     0.0     0.0     0.0     0.0     0.4     0.4     0.0
$#     e0      v0
 253312.5     1.0
```

Same we have no deviatoric stress and for pressure $p = [c_4 + c_5(\rho/\rho_0 - 1)] * E = 0.4 * \rho/\rho_0 * E$. Ring a bell? Idea gas law has a $\gamma = 1.4$ and $p = (\gamma - 1) * \rho/\rho_0 * E$. Its initial pressure of 101,325 Pa is prescribed by assigning internal energy density to $e_0 = \frac{p}{0.4} = 253312.5$ as $\rho/\rho_0 = 1$.

Take another example, high explosive (HE). It is defined by *MAT_HE + *EOS_JWL (*EOS_02).

```
*MAT_HIGH_EXPLOSIVE_BURN
$#    mid       ro        d       pcj      beta         k        g      sigy
      2    1630.0    6930.02.10000E10       0.0       0.0       0.0       0.0
*EOS_JWL
$#  eosid         a         b        r1        r2      omeg        e0        vo
      23.71200E113.231000E9      4.15      0.95      0.37.000000E9       0.0
```

Again, we assume no deviatoric stress and pressure is a function of both compression ratio and internal energy. $p = A\left(1 - \frac{\omega}{R_1 V}\right)e^{-R_1 V} + B\left(1 - \frac{\omega}{R_2 V}\right)e^{-R_2 V} + \frac{\omega E}{V}$, where $V = \rho_0/\rho$. It contains two exponential decay terms and an idea gas term to model gaseous explosion process. *MAT_HE controls burn fraction. Basically the ratio of burning at current state. And the pressure is burn fraction times the *EOS_JWL pressure. If a piece of HE is fully burnt, the pressure is simply *EOS_JWL pressure. If not burnt at all, the pressure is zero. At the initial state, everywhere HE is not detonated so everywhere HE has a pressure of zero. From the design of *MAT_HE and *EOS_JWL, we could see that the pressure here is rather gauge pressure than the "real" pressure and they are differed by 1 bar.

Wait. Then how about if we have both air and HE inside an ALE model? Air has an initial pressure of 1 bar (it has to. otherwise how does it expand?). And HE has an initial pressure of … 0! There is no way to reach pressure equilibrium at the first state.

This pressure incompatibility is there for real and we have been living with it for quite some long time. To our defense, HE pressure is of magnitudes higher than air so this discrepancy is considered small, if not tiny. But now we have a way to resolve it. We will see that later in this section.

Reference Pressure

Here is another new to us structural engineers. Reference pressure. We go back to that free-sitting block on the ground. What is its boundary condition? It has a z-direction constraints on nodal motion at the bottom face, right? Is that all?

Right, and wrong. Indeed we only need to constrain z motion at the bottom face to make the simulation right. But one thing most people would overlook is that at all other 5 faces we have pressure boundary condition. Pressure at those 5 faces is … zero. This boundary condition is naturally satisfied. We are getting used to it so much that we never remember there is a boundary condition at those faces.

Now let us continue working on the air, water, HE model. In the previous section, we set up the material definitions and prescribed the initial pressure of each material. Take air, it has an initial pressure of 1 bar and occupies the upper portion of the box-shaped S-ALE mesh. Do we need to apply boundary conditions, say, at the top face?

We must. The air pressure is 1 bar. Without applying a 1 bar pressure boundary condition, the inside air would expand and fly out of the S-ALE mesh. After some time, all air flows out and the run crashes.

One way to apply this pressure boundary condition is to pick all surface ALE segments and apply pressure loadings on those segments through *LOAD_SEGMENT. This is straightforward and easy to understand. But it is a little bit costly. So we took another approach. Instead we subtract this 1 bar pressure from all elements'

pressure when evaluating the internal force. As internal force is only affected by pressure difference, not the real pressure, the nodal force of internal nodes remains the same. To surfaces nodes, subtracting 1 bar off from element pressure is equivalent to applying a 1 bar pressure loading. This is a much faster and more efficient way computational cost wise.

So what we do is, instead of applying pressure boundary conditions, prescribing a reference pressure for each ALE fluid. This brings us a well preserved pressure equilibrium, between ALE fluids and outside.

In addition to that, the newly added *ALE_STRUCTURED_MULTI-MATERIAL_GROUP card had a design to reconcile pressures between different ALE fluids. In last section's example, water and air pressure both being 1 bar at the initial stage; but HE has a base pressure of 0 bar. Previously, the ALE setup only allows for one global reference pressure (PREF in *CONTROL_ALE). So in this case, in elements occupying by HE, pressure was wrongfully subtracted by 1 bar pressure in internal force calculation. *ALE_STRUCTURED_MULTI-MATERIAL_GROUP, to address this problem, allows for each ALE fluid to have its own reference pressure. We can easily reach pressure equilibrium by assigning reference pressure 1 bar to air and water, 0 to HE. Please refer to the example below to better understand this idea.

A Simple Example

We have three ALE fluids, water, air and HE. And a box-shaped S-ALE mesh. Air occupies the upper half; water the lower half; and HE is a small cylinder submerged in the water. Below is the ALE multi-material keywords setup.

Two points to note here.

1. Order matters. HE in the first line, water next, air the third. Of course, we could go the other way around, like air first, then water, HE last.
2. Each ALE fluid has its own reference pressure. Which is the pressure difference between itself and the outside world. For air and water, it is 1 bar. And for HE, it is 0.

```
*ALE_STRUCTURED_MULTI-MATERIAL_GROUP
$# mmgname      mid    eosid                          pref
  HE            3      3                               0.0
water           1      1                          101325.0
  air           2      2                          101325.0
*MAT_NULL
$#    mid      ro      pc      mu    terod    cerod      ym      pr
      1    998.0     0.0     0.0     0.0     0.0     0.0     0.0
*EOS_LINEAR_POLYNOMIAL
$#  eosid      c0      c1      c2      c3      c4      c5      c6
      1     0.0    2.2e9     0.0     0.0     0.0     0.0     0.0
$#     e0      v0
     0.0     1.0
*MAT_NULL
$#    mid      ro      pc      mu    terod    cerod      ym      pr
      2     1.23     0.0     0.0     0.0     0.0     0.0     0.0
```

```
*EOS_LINEAR_POLYNOMIAL
$#  eosid      c0      c1      c2      c3      c4      c5      c6
      2      0.0      0.0      0.0      0.0      0.4      0.4      0.0
$#     e0      v0

 253312.5      1.0
*MAT_HIGH_EXPLOSIVE_BURN
$#    mid      ro       d      pcj     beta       k       g     sigy
      2    1630.0    6930.02.10000E10      0.0      0.0      0.0      0.0
*EOS_JWL
$#  eosid       a       b      r1      r2     omeg      e0      vo
      23.71200E113.231000E9     4.15     0.95      0.37.000000E9      0.0
```

Ending Remarks

LS-DYNA ALE module has been known for its steep learning curve. Partially it was because setting up Eulerian models are intrinsically different from Lagrange models. But the design of ALE keyword cards, for sure, has caused quite a lot of confusions among our users, new and experienced.

To prompt LS-DYNA ALE usages, Structured ALE solver introduced a new, user-friendly, streamlined three-step setup. We hope this effort could help users, new or old, to perform their work more efficiently and smoothly.

On Setting up Boundary and Initial Conditions in S-ALE Models

Hao Chen, Ansys

LS-DYNA ALE has been widely used to simulating moving fluids interacting with structures. Unlike CFD, the focus is rather on the structure response under dynamic loading from fluids, than the fluids' motion. Fluids are agitated by a high pressure gradient; and then hit the structure, carrying a large momentum. The key in successfully capturing the physics lies in the fluid-structure interaction algorithm. It needs to accurately predict the peak of pressure loading during the impact, which is characterized as a momentum transfer process. This request could only be fulfilled by a transient analysis with a penalty-based coupling between fluids and structure.

In 2015, LSTC introduced a new structured ALE (S-ALE) solver option dedicated to solve the subset of ALE problems where a structured mesh is appropriate. As expected, recognizing the logical regularity of the mesh brought a reduced simulation time for the case of identical structured and unstructured mesh definitions. It also comes with a cleaner, conceptually simpler way of model setup. This article gives a brief description on setting up the boundary and initial conditions in S-ALE models.

What is Boundary?

In a Lagrangian model, mesh conforms to the material interface. The boundary is a collection of surface segments and nodes enclosing the material (*PART). And we could choose a set of segments or nodes to apply boundary conditions on them. For example, one uses *LOAD_SEGMENT to apply pressure on a set of segments; *BOUNDARY_SPC to apply nodal constraints on a set of nodes, etc.

However, in an ALE model, things are totally different. Mesh is no longer the spatial representation of the material. Rather, it is simply a domain in which we study the flow of that (and other) material. In Lagrangian, what really happened was that we applied boundary conditions on the material surface. The equivalent thing to do in an ALE model is to apply some force or constraints at the material interface, NOT at the mesh boundary. As the mesh boundary is just the limit of our working field, nothing more.

As the material interface is evolving in ALE models, plus it has no explicit description, generally it is not possible to apply "boundary conditions" directly at the material interface. Most of times, boundary conditions are applied through fluid-structure interaction (FSI). For example, we want to create a wave by pushing water sideways from left to right. If modeled as Lagrangian water, all we need to do is to find its boundary nodes and apply a prescribed motion on them. In ALE, it is not possible as these nodes do not move with water. Remember? Mesh does not move; fluids flow inside it. So what we do is to create a Lagrange plate and set up a FSI between water

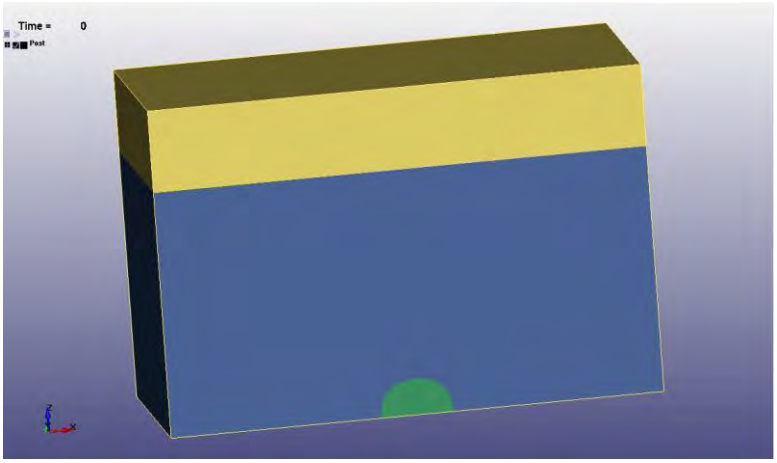
and the plate. Then we apply this prescribed motion onto the plate. Then FSI is going to enforce the water to move together with the plate.

Boundary Conditions

The above being said, on one condition we could still use boundary conditions in an ALE model. As you might have guessed, that is when the mesh boundary conforms to the material interface. Say we have a half symmetry model and -x mesh face is the symmetry plane. What we need to do is to apply nodal constraints along x direction on all nodes at that face. Another example is a flat bottom container. If we choose to align the bottom mesh face with that simple geometry container, we could apply a SPC along z direction at -z face nodes.

Another exception is transmitting boundary condition, i.e. *BOUNDARY_NON_REFLECTING. The transmitting boundary is to apply an impedance to minimize the pressure wave reflection at the mesh boundary when we use a finite mesh to model the infinite domain. It is designed for Lagrangian but used often in ALE models to help reducing the model size.

In both cases, we first define a set of nodes (SPC) or segments (NON_REFLECTING). And then we setup the SPC cards or NON_REFECTING cards using those definitions. Let us use a case of underwater explosion to show the process. We have three multi-materials (fluids) in the model - a half sphere high explosive (HE), water above and around it, and then air on the top, as shown in the figure below. We construct a half symmetry model with symmetric plane aligning with -y face.



https://ftp.lstc.com/anonymous/outgoing/hao/sale/models_R121/underwater/

We follow the three-step setup. First mesh. A 24x12x16 box spans from (-12,0,0) to (12,0,16).

```
*ALE_STRUCTURED_MESH
$    mshid       pid      nbid      ebid      ityp    nparts
         1         9    200001    200001         0         0
$     nptx      npty      nptz      nid0     lcsid
      3001      3002      3003
*ALE_STRUCTURED_MESH_CONTROL_POINTS
```

```
      3001
                     1               -12.00
                    25                12.00
*ALE_STRUCTURED_MESH_CONTROL_POINTS
      3002
                     1                 0.00
                    13                12.00
*ALE_STRUCTURED_MESH_CONTROL_POINTS
      3003
                     1                 0.00
                    17                16.00
```

Next, multi-materials. Water, HE and Air. Please note, air has a reference pressure of 1.0e-5; water and HE have their reference pressure set to 0.

```
*ALE_STRUCTURED_MULTI-MATERIAL_GROUP
$# mmgname        mid     eosid
pref
  water            10        10
  He               11        11
  air              12        12                                        1.0e-
5
*MAT_NULL
$#     mid        ro        pc        mu     terod     cerod        ym
pr
        10  1.000000
*EOS_GRUNEISEN
$#   eosid         c        s1        s2        s3     gamao         a
e0
        10  0.148000  1.750000     0.000     0.000  0.280000
$#      v0
  1.000000
*MAT_HIGH_EXPLOSIVE_BURN
$#     mid        ro         d       pcj      beta         k         g
sigy
        11  1.630000  0.784000  0.260000
*EOS_JWL
$#   eosid         a         b        r1        r2      omeg        e0
vo
        11  3.710000  0.032300  4.150000  0.950000  0.300000  0.043000
1.00000
*MAT_NULL
$#     mid        ro        pc        mu     terod     cerod        ym
pr
        12  0.001280
*EOS_LINEAR_POLYNOMIAL
$#   eosid        c0        c1        c2        c3        c4        c5
c6
        12     0.000       0.0     0.000     0.000  0.400000  0.400000
$#      e0        v0
   2.25e-5     0.000
```

And then, volume filling. First, all elements filled with water. Next, inside a sphere to HE. Then above a plane to air.

```
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$    mshid                  to
         1               water
$ geometry
       ALL
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$    mshid                  to
         1                  HE
$ geometry    in/out      NID1         r
 ELLIPSOID               199997       2.0
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$    mshid                  to
         1                 air
$ geometry    in/out      NID1      NID2
    PLANE                199998    199999
*NODE
  199997   0.0000000e+00   0.0000000e+00   0.0000000e+00
  199998   0.0000000e+00   0.0000000e+00  12.0000000e+00
  199999   0.0000000e+00   0.0000000e+00  15.0000000e+00
```

And let us be a Minimalist on *CONTROL_ALE.

```
*CONTROL_ALE
$#     dct      nadv      meth      afac      bfac      cfac      dfac
efac
                   1         1 -1.000000
$#   start       end     aafac     vfact      prit       ebc      pref
idebc
```

Now to apply boundary conditions on the six faces of the S-ALE mesh. They could be categorized into three types:

1. Symmetric plane: -y face. This translates to a *BOUNDARY_SPC on all nodes at that face along the local y direction.
2. No flow in and out: -z face. This face aligns perfectly to the seabed. Water and HE could move freely inside the plane. But nothing could flow in/out of the plane. This translates to a *BOUNDARY_SPC on all nodes at that face along the local z direction.
3. Transmitting boundary: all other 4 faces. At those four faces, we want to allow the pressure wave travel freely into the unmeshed infinite domain. We do that by adding a *BOUNDARY_NON_REFLECTING on all segments on those faces.

For a more user-friendly S-ALE setup, we added the following “macro-like” keyword to apply SPC constraints. It will be available in the next release in R12 (R12.1).

*BOUNDARY_SALE_MESH_FACE							
OPTION	MSHID	-X	+X	-Y	+Y	-Z	+Z
SYM	1			1			
NOFLOW	1					1	
NONREFL	1	1	1		1		1

Internally it is translated into the following keywords:

```
*BOUNDARY_SPC_SET
         1                    0         1         0
*BOUNDARY_SPC_SET
         2                    0         0         1
*SET_NODE_GENERAL
$       SID
         1
$    OPTION      MSHID        XMN        XMX        YMN        YMX        ZMN
ZMX
    SALEFAC          1                             1
*SET_NODE_GENERAL
$       SID
         2
$    OPTION      MSHID        XMN        XMX        YMN        YMX        ZMN
ZMX
    SALEFAC          1                                                      1
*BOUNDARY_NON_REFLECTING
$       SID
        11
*SET_SEGMENT_GENERAL
$       SID
        11
$    OPTION      MSHID        XMN        XMX        YMN        YMX        ZMN
ZMX
    SALEFAC          1          1          1                    1
1
```

It is totally users' choice to use *BOUNDARY_SALE_MESH_FACE or not. Other than being concise, another advantage is that it is less error prone. Especially when the mesh is tilted, i.e. there is a local coordinate system. In this case, if we use *BOUNDARY_SPC, we need to remember to set up a local coordinate system, and make sure that we constraint the correct component. *BOUNNDARY_SALE_MESH_FACE, on the other hand, handles all those silently and automatically and takes off those unnecessary burdens from users.

One thing to note is that the two options SYM and NOFLOW, are doing the same thing. That is to constrain the flow perpendicular to the plane. We have both options available, simply to provide a one-to-one match between the options and real physical scenarios.

Another option, as you might have guessed, is FIXED. It is to fix all nodal motions at that face.

SALECPT and SALEFAC in *SET_?_GENERAL

In the model example above, when using the traditional *BOUDNARY_SPC and *BOUNDARY_NON_REFLECTING setup, one can notice that *SET_NODE_GENERAL and *SET_SEGMENT_GENERAL are used to generate node and segment sets, respectively. Unlike traditional ALE, S-ALE relies solely on *SET_?_GENERAL to create node/segment/element sets. This is because S-ALE mesh is generated internally so it is difficult, if not possible, to explicitly list

out IDs of nodes/segments/elements. So what we do is to generate those sets by using SALECPT and SALEFAC options in *SET_?_GENERAL.

SALEFAC will pick up all nodes/segments/solids at certain S-ALE mesh face(s). The one below added all nodes at -y face of S-ALE mesh #1 into node set #1.

```
*SET_NODE_GENERAL
$      SID
         1
$   OPTION      MSHID        XMN        XMX        YMN        YMX        ZMN
ZMX
   SALEFAC          1                              1
```

SALECPT provides a more flexible way. It picks up all nodes/segments/elements inside a box indexed by S-ALE control points. For example, this one below added the bottom half of nodes in the S-ALE mesh #1 into node set #101. (Remember our mesh is a box of 24x12x16 elements?)

```
*SET_NODE_GENERAL
$      SID
       101
$   OPTION      MSHID        XMN        XMX        YMN        YMX        ZMN
ZMX
   SALECPT          1          1         25          1         13          1
9
```

One thing we need to stress here. No other options in *SET_?_GENERAL could be applied onto S-ALE mesh. A common mistake is BOX option. The reason we did not support it was to avoid user mistakes. Most of the time Lagrange mesh and S-ALE mesh are on top of each other, i.e., overlaps in space. And more likely users tend to pick nodes inside a box for the Lagrange mesh; or the S-ALE mesh. Not both. Supporting BOX option might accidentally include S-ALE nodes into an intended Lagrange node set, without users even realizing that.

Initial Conditions

Initial conditions are relatively simple. Typically it is only to assign some initial velocity to some nodes. But still, there is some fine difference between LAG and ALE models. Again, we must emphasize that ALE material interface is not necessarily at the boundary. And it makes sense to apply initial velocities on certain nodes only if those nodes are the real spatial representation of a material.

Most commonly, initial velocities are applied simultaneously with volume filling process, through the *ALE_STRUCTURED_MESH_VOLUME_FILLING card. Like the following:

```
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$#   mshid         -    ammgto         -   nsample         -         -       vid
         1              plate                                                  1
$#    geom    in/out     boxid
BOXCOR             0         1
```

And the initial velocity (vx=-61.631,vy=208.06) is prescribed by using the *DEFINE_VECTOR card.

```
*DEFINE_VECTOR
$#     vid        xt        yt        zt        xh        yh        zh       cid
         1   -61.631    208.06       0.0       0.0       0.0       0.0         0
```

The volume filling process will first fill the box with the multi-material named “plate” and then, find all nodes belongs to “plate” and assign an initial velocity to them.

Ending Remarks

LS-DYNA ALE module has been known for its steep learning curve. Partially it was because setting up Eulerian models are intrinsically different from Lagrange models. But the design of ALE keyword cards, for sure, has caused quite a lot of confusions among our users, new and experienced.

To prompt LS-DYNA ALE usages, Structured ALE solver introduced a new, user-friendly, streamlined three-step setup. We hope this effort could help users, new or old, to perform their work more efficiently and smoothly.

On Setting up a 2D Structured ALE Model

Hao Chen, Ansys

LS-DYNA ALE has been widely used to simulating moving fluids interacting with structures. Unlike CFD, the focus is rather on the structure response under dynamic loading from fluids, than the fluids' motion. Fluids are agitated by a high pressure gradient; and then hit the structure, carrying a large momentum. The key in successfully capturing the physics lies in the fluid-structure interaction algorithm. It needs to accurately predict the peak of pressure loading during the impact, which is characterized as a momentum transfer process. This request could only be fulfilled by a transient analysis with a penalty-based coupling between fluids and structure.

In 2015, LSTC introduced a new structured ALE (S-ALE) solver option dedicated to solve the subset of ALE problems where a structured mesh is appropriate. As expected, recognizing the logical regularity of the mesh brought a reduced simulation time for the case of identical structured and unstructured mesh definitions. It also comes with a cleaner, conceptually simpler way of model setup.

Since the past year, efforts have been made to expand S-ALE into the 2D territory. This would be beneficial to both users and the developer. Using a single set of coding for both 3D and 2D models would greatly simplify the maintenance. In the meantime, it also provides a unified platform for future developments. As S-ALE 3D coding has been tested intensively during the last five years, we believe the S-ALE 2D solver would be stable and efficient from the beginning. For users, using the same set of keywords in both 3D and 2D applications could reduce the learning time and modeling efforts so they could focus more on the physical part of the modeling.

This article gives a brief description of the S-ALE 2D model setup.

Three step setup

We follow a straight-forward three step setup. First, mesh; secondly, material properties of fluids; thirdly, filling the mesh with fluids. In this section, we describe the three keywords doing these three steps.

4. Mesh generation: *ALE_STRUCTURED_MESH; *ALE_STRUCTURED_MESH_CONTROL_POINTS

In S-ALE, mesh is always rectangular. To determine the mesh layout in the space, we need the following information:

c. Mesh spacing along three axes (LCIDX,LCIDY,LCIDZ) in 3D; two axes (LCIDX,LCIDY) in 2D.
d. The origin (NID0), and local coordinate system (LCSID).

Other fields are for identification purpose only and are self-explanatory. MSHID stands for mesh ID; DPID Part ID; NBID and EBID are the IDs of first S-ALE mesh node and element, respectively.

TDEATH is to the "death time" for S-ALE mesh. It is to turn off the S-ALE calculation once the most of fluid loading is applied; and keep the Lagrange model running as the structure deformation is not fully developed yet.

*ALE_STRUCTURED_MESH							
MSHID	DPID	NBID	EBID				TDEATH
CPIDX	CPIDY	CPIDZ	NID0	LCSID			

The CPIDX, CPIDY and CPIDZ are IDs of *ALE_STRUCTURED_MESH_CONTROL_POINTS cards. Each card specifies the mesh spacing along one axis.

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
CPID							
N1		X1		RATIO1			
…		…		…			
Nn		Xn		RATIOn			

In a 2D simulation, the mesh is 2 dimensional so CPIDZ is no longer needed. Nonzero value of CPIDZ would be simply ignored and hence harmless. For 2D Axisymmetric case, special caution needs to be taken when using NID0 (origin shift) and LCSID (mesh rotation) feature, to make sure the created mesh is not in conflict with the assumption that the axis of symmetry is at x=0.

5. Material definitions: *ALE_STRUCTURED_MULTI-MATERIAL_GROUP_?

S-ALE mesh is simply a spatial domain in which fluids flow. In order to let the code know what and how many fluids there are, we need to provide material properties of each fluid and list them all under the card *ALE_STRUCTURED_MULTI-MATERIAL_GROUP. Please note, AMMG stands for ALE multi-material group, a rather alternative and maybe confusing name for "ALE fluid". In this paper we are going to use AMMG and fluid interchangeably.

For 2D, *ALE_STRUCTURED_MULTI-MATERIAL_GROUP also carries another role. It is to specify the nature of the 2D simulation (element formulation), i.e. Plane strain or Axisymmetric. An option string at the end, either PLNEPS or AXISYM does the job. These two options directly correspond to setting elform=13 and elfrom=14 in *SECTION_ALE2D card.

*ALE_STRUCTURED_MULTI-MATERIAL_GROUP_PLNEPS/AXISYM							
AMMGNM1	MID1	EOSID1					PREF1
…	…	…					…
AMMGNMn	MIDn	EOSIDn					PREFn

"AMMGNM" is a name one gives to a AMMG (ALE Multi-Material Group), aka ALE fluid. It is used in other cards, for example, *SET_MULTI-MATERIAL_GROUP_LIST to refer to that AMMG. "MID" and "EOSID" are the material ID and EOS ID, respectively.

"PREF" is to describe the reference pressure or "base pressure" of that fluid. This might be somewhat new to our typical users from solids background. Pressure of a solid material, if not preloaded, always starts from zero. In such case, its reference pressure or base pressure, is zero. But most fluids have non-zero reference pressure. For example, air has a base pressure of 101325 Pa (1 bar atmospheric pressure). Traditionally this reference pressure is prescribed using the field "PREF" in *CONTROL_ALE card. The new *ALE_STRUCTURED_MULTI-MATERIAL_GROUP has a design to allow each AMMG to have its own reference pressure. The author believes this added flexibility could be proven very useful in certain applications.

6. Volume Filling: *ALE_STRUCTURED_MESH_VOLUME_FILLING

We created a rectangular 2D S-ALE mesh in step 1 and came up multiple fluids (AMMGs) definitions in step 2. We now fill the mesh with those fluids. We do that by specifying which fluid occupies either inside or outside of certain geometry. These geometries could be simple shapes like sphere, plane, box, cylinder. Or it could be user-defined complex shape like structure surfaces.

This is done by using the keyword "*ALE_STRUCTURED_MESH_VOLUME_FILLING". The "volume filling" process typically is done through multiple "tasks", each task by a separate keyword.

*ALE_STRUCTURED_ MESH_VOLUME_FILLING							
MSHID		AMMGTO		NSAMPLE			VID
GEOM	IN/OUT	E1	E2	E3	E4		

MSHID is the S-ALE mesh ID, defined in *ALE_STRUCTURED_MESH card. And AMMGTO is the name of ALE fluid to be filled, defined in *ALE_STRCUTURED_MULTI-MATERIAL_GROUP. VID is to prescribe the initial velocity of that fluid, if any. And NSAMPLE is default to 3 which means that one ALE cell is divided into 7x7x7 (7=2*3+1) sub-cells in 3D and 7x7 in 2D and each sub-cell is checked to see if it is inside/outside.

It supports 5 basic geometries: Plane, Cylinder, Ellipsoid, Box with indices and Box with coordinates. And E1-E4 are used to provide information of these geometries. For complicated geometries, we need users to provide us with a segment set (or something we could internally convert to a segment set). The assumption is that all segment normals are consistent; and those normals point to the "inside". For convenience, we provide a "IN/OUT" flag for an easier flip.

Boundary/Initial Conditions and *CONTROL_ALE

Boundary Conditions. The most used type of boundary condition is single point constraint (SPC). It is to constrain motion at a point (node) along certain direction. Traditionally, we use *BOUNDARY_SPC to apply SPC constraints in S-ALE. In R12.1 (Ansys 2021R2), we added a new "macro-like" keyword "*BOUNDARY_SALE_MESH_FACE" to make this process more user-friendly.

*BOUNDARY_SALE_MESH_FACE							
OPTION	MSHID	-X	+X	-Y	+Y	-Z	+Z
SYM	1			1			
NOFLOW	1					1	1
NONREFL	1	1	1		1		

Three options, SYM, NOFLOW and FIXED, are for SPC constraint. SYM and NOFLOW, are doing the same thing. That is to constrain the flow perpendicular to the plane. We have both options available, simply to provide a one-to-one match between the options and real physical scenarios. FIXED is to fix all nodal motions at that face.

The other one, NONREFL, is to apply non-reflecting boundary at S-ALE mesh face(s). Internally this option is translated to *BOUNDARY_NON_REFLECTING + *SET_SEGMENT.

Initial conditions. IC setup is relatively simple. Typically it is only to assign some initial velocity to some nodes. Most commonly, initial velocities are applied simultaneously with volume filling process, through the *ALE_STRUCTURED_MESH_VOLUME_FILLING card. Like the following:

```
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$#   mshid         -    ammgto         -   nsample         -         -       vid
         1              plate                                                  1
$#    geom    in/out     boxid
BOXCOR             0         1
```

And the initial velocity (vx=-61.631,vy=208.06) is prescribed by using the *DEFINE_VECTOR card.

```
*DEFINE_VECTOR
$#     vid        xt        yt        zt        xh        yh        zh       cid
         1   -61.631    208.06       0.0       0.0       0.0       0.0         0
```

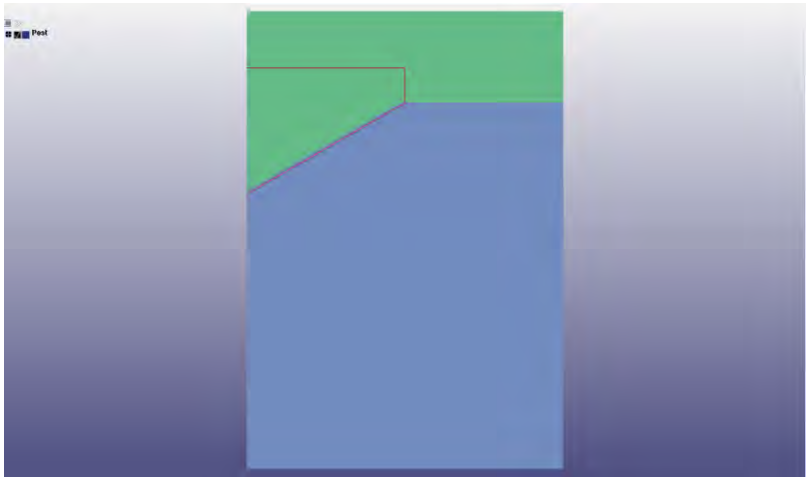
Control Card. *CONTROL_ALE is used to set up certain global control parameters in a S-ALE simulation. The only relevant parameters are: NADV and METH. METH=1/2 controls the order of advection scheme, 1st or 2nd order. NADV could be adjusted to increase the interval (in number of cycles) between two advections. It is to provide users with a way to shorten the running time for problems in which deformation is small compared to element size in one cycle. Then we could wait until the element is severely deformed and then perform an advection (remapping). All other parameters most cases the user could safely ignore.

```
*CONTROL_ALE
$#     dct      nadv      meth      afac      bfac      cfac      dfac
efac
                   1         1 -1.000000
$#   start       end     aafac     vfact      prit       ebc      pref
idebc
```

The parameter PREF was to set the reference pressure. However, S-ALE came up with a better design to allow users to prescribe reference pressure for each ALE fluid (AMMG) through "PREFn" in *ALE_STRUCTURED_MULTI-MATERIAL_GROUP. PREF will still be kept there for backward compatibility but we strongly recommend setting reference pressure through the *ASMMG card.

Example

Now let us use a simple example to illustrate the 2D S-ALE setup. We have a rigid body wedge composed of beam elements slamming into water as the figure shown below. It is a 2D plane strain model. The unit used is mm-ton-s. Pressure unit is Mpa.



Step 1: To construct a mesh spans (0,-433.013) to (500,288.675) with 100x150 elements.

```
*ALE_STRUCTURED_MESH
$    mshid       pid      nbid      ebid
         1        11    100001    100001
$    cpidx     cpidy     cpidz      nid0     lcsid
      1001      1002      1003
*ALE_STRUCTURED_MESH_CONTROL_POINTS
      1001
$                x1                  x2                  x3                  x4
                  1                 0.0
                101               500.0
*ALE_STRUCTURED_MESH_CONTROL_POINTS
      1002
$                x1                  x2                  x3                  x4
                  1            -433.013
                151             288.675
*ALE_STRUCTURED_MESH_CONTROL_POINTS
      1003
$                x1                  x2                  x3                  x4
                  1                -0.5
                  2                 0.5
```

Here CPIDZ is harmless. All nodal coordinates will be initialized with z=0. We will see its purpose later.

Step 2: Set up ALE multi-material Groups (AMMGs or ALE fluids). There are totally 2 AMMGs defined. First is "water" by *MAT_NULL (MID=4) and *EOS_LINEAR_POLYNOMIAL (EOSID=4). The second is "air" by * MAT_NULL (MID=5) and *EOS_LINEAR_POLYNOMIAL (EOSID=5). Their materials properties are given as follows.

```
*MAT_NULL
$#     mid        ro        pc        mu     terod     cerod        ym        pr
         4  1.026e-9       0.0       0.0       0.0       0.0       0.0       0.0
*EOS_LINEAR_POLYNOMIAL
$#   eosid        c0        c1        c2        c3        c4        c5        c6
         4       0.0       2.3       0.0       0.0       0.0       0.0       0.0
$#      e0        v0
       0.0       0.0
*MAT_NULL
$#     mid        ro        pc        mu     terod     cerod        ym        pr
         5 1.293e-12       0.0       0.0       0.0       0.0       0.0       0.0
*EOS_LINEAR_POLYNOMIAL
$#   eosid        c0        c1        c2        c3        c4        c5        c6
         5       0.0       0.0       0.0       0.0       0.4       0.4       0.0
$#      e0        v0
       0.0       0.0
```

Here for simplicity, we assigned initial pressure to be zero for both air and water. In problems where the air compressibility becomes important, we could prescribe the initial pressure of 1 bar by assigning e0=0.101325/0.4=0.258125. If we do that, please do not forget to set the reference pressure of water to 1 bar by making PREF=0.101325 in *ALE_STRUCTURED_MULTI-MATERIAL_GROUP (*ASMMG) otherwise water pressure is not well balanced.

Now construct the ALE fluids by using *ALE_STRUCTURED_MULTI-MATERIAL_GROUP. And specify the nature of the simulation to 2D plane strain by adding option _PLNEPS at the end.

```
*ALE_STRUCTURED_MULTI-MATERIAL_GROUP_PLNEPS
$     name       mid     eosid                                              pref
   water             4         4
  air                5         5
```

Step 3: Volume Filling the initial S-ALE mesh. By *ALE_STRUCTRUED_MESH_VOLUME_FILLING.

First, fill all with "water".

```
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$#   mshid         -    ammgto         -   nsample         -         -
vid
         1             water
$#    geom    in/out
ALL
```

Next, switch above the plane to "air".

```
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$#   mshid         -    ammgto         -   nsample         -         -       vid
         1               air
$#    geom    in/out      nid1      nid2
```

```
 PLANE                    10001     10002
*NODE
$#   nid               x               y               z       tc      rc
   10001             0.0         144.338
   10002             0.0         145.338
```

Finally, we switch the fluid inside the wedge to “air”.

```
*ALE_STRUCTURED_MESH_VOLUME_FILLING
$#   mshid          -    ammgto          -   nsample          -         -
vid
         1               air
$#    geom    in/out    ssetid
 SEGSET            1         1
```

Boundary Conditions and *CONTROL_ALE card

```
*BOUNDARY_SALE_MESH_FACE
$   option     mshid        -x        +x        -y        +y        -z        +z
    NOFLOW         1         1         1         1         1         1         1
```

Here we also set the SPC constraints at -z and +z mesh faces. It is not necessary but generally harmless. We will see why we did that in a minute.

```
*CONTROL_ALE
$#     dct      nadv      meth      afac      bfac      cfac      dfac      efac
                            1
$#   start       end     aafac     vfact      prit       ebc      pref   nsidebc
```

Remember how we want our users to be minimalist on *CONTROL_ALE?

FSI Setup

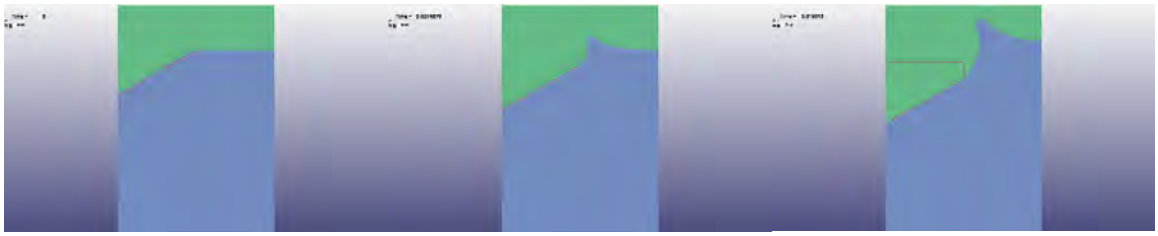
In the next R12.1 (Ansys 2021R2) release, *ALE_STRUCTURED_FSI card becomes fully functional with added 2D support. And here we are using this much simpler, stabler card embedded with a better leakage control algorithm.

```
*ALE_STRUCTURED_FSI
$#   slave    master     sstyp     mstyp                                   mcoup
         1        11         2         1                   4                -
123
$#   start       end      pfac                          flip
       0.0       0.0        -1
*SET_MULTI_MATERIAL_GROUP_LIST
$#  ammsid
       123
$# ammgid1   ammgid2   ammgid3   ammgid4   ammgid5   ammgid6   ammgid7   ammgid8
   water
*DEFINE_CURVE
$#    lcid
         1
$#                a1                  o1
                0.0                 0.0
                1.0                 1.0
```

Simply put, what we did in the above cards is as follows:

1. Couple a segment set (sstyp=2) with id 1 (slave=1) to "water" (mcoup=-123) in the S-ALE mesh part (mstyp=1) with id 11 (master=11).
2. Specify the couple stiffness to use the load curve id 1 (PFAC=-1). This load curve establishes a linear relationship between penetration and penalty spring pressure. It comes with two points. The first is always (0,0) which means no penalty pressure is given when the penetration is zero. The second point we recommend (0.1xdl, p_max). This is to tell the FSI algorithm to apply the p_max on coupling surface when fluids penetration is around 1/10 of the S-ALE element size. P_max is the maximum impact pressure that users need to come up with their own guess/estimate. The load curve used here instructs FSI to apply a penalty pressure of 10 bars at a penetration of 1 mm. We did not follow the rule here only because the toy nature of this simple model.

Below is the water and wedge at t=0., t=7.5ms and t=15ms. The input deck is available at https://ftp.lstc.com/anonymous/outgoing/hao/sale/models_R121/slam_wedge/. It took only 2 seconds to run this model with my 16 core machine.



Axisymmetric and 3D models

Here we show how easy we could switch between Plane Strain, Axisymmetric and 3D simulations. First let us change this plane strain model to an axisymmetric one. All we need to do is to replace PLNEPS by AXISYM in the *ASMMG card.

```
*ALE_STRUCTURED_MULTI-MATERIAL_GROUP_AXISYM
$     name       mid     eosid                                          pref
   water            4         4
  air               5         5
```

For Lagrange part of the model, we need to change the beam element formulation from 7 (plane strain) to 8 (axisymmetric). And that is it!

```
*SECTION_BEAM
$#   secid    elform
         1         8
$#      t1        t2        t3        t4
       1.0       1.0       1.0       1.0
```

Now we change it to a 3D model. All we need to do is to strike out _PLNEPS from the *ASMMG card.

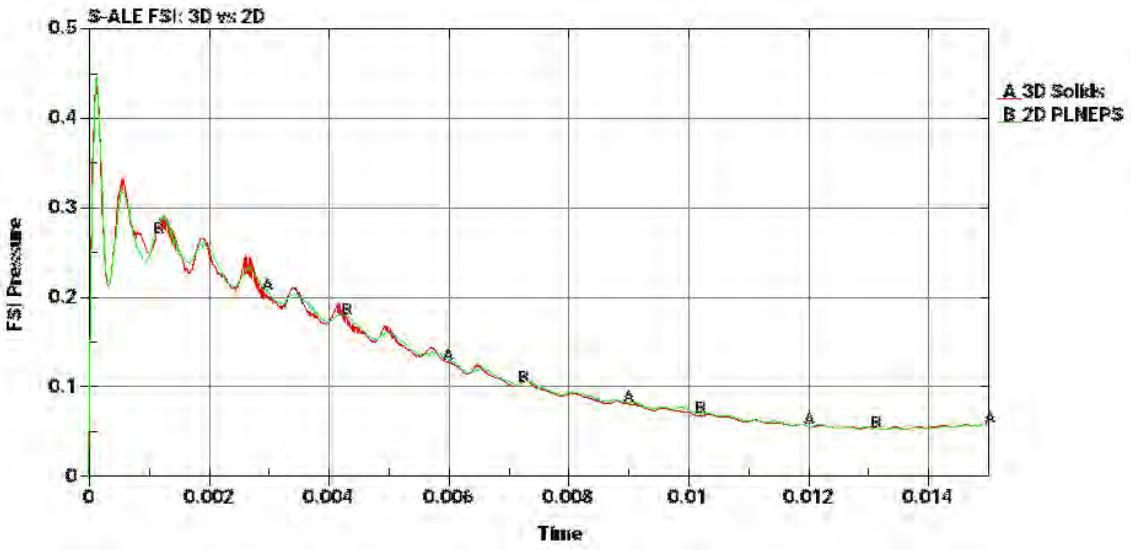
```
*ALE_STRUCTURED_MULTI-MATERIAL_GROUP
$     name        mid     eosid                                          pref
   water            4         4
  air               5         5
```

Of course, for LAG part, we need to prepare a different set of mesh with different nodes and elements defined. The model input deck contains both 2D and 3D Lagrange mesh and one could include the corresponding mesh files.

```
*INCLUDE
$mesh_wedge_2d.k
mesh_wedge_3d.k
```

3D simulation is a little bit more costly – 5 seconds on my 16 core machine. So we urge the reader to download the test case input deck and play with the switching between plane strain, axisymmetric and 3D simulations. We promise it is going to be fast and easy!

Below figure compares FSI pressure between plane strain and 3D simulations. We could see these two curves almost on top of each other. This validates the newly developed 2D S-ALE works as expected.



Ending Remarks

LS-DYNA ALE module has been known for its steep learning curve. Partially it was because setting up Eulerian models are intrinsically different from Lagrange models. But the design of ALE keyword cards, for sure, has caused quite a lot of confusions among our users, new and experienced.

To prompt LS-DYNA ALE usages, Structured ALE solver introduced a new, user-friendly, streamlined three-step setup. We hope this effort could help users, new or old, to perform their work more efficiently and smoothly.

On Setting up the Structured ALE Mesh

Hao Chen, Ansys

LS-DYNA ALE has been widely used to simulating moving fluids interacting with structures. Unlike CFD, the focus is rather on the structure response under dynamic loading from fluids, than the fluids' motion. Fluids are agitated by a high pressure gradient; and then hit the structure, carrying a large momentum. The key in successfully capturing the physics lies in the fluid-structure interaction algorithm. It needs to accurately predict the peak of pressure loading during the impact, which is characterized as a momentum transfer process. This request could only be fulfilled by a transient analysis with a penalty-based coupling between fluids and structure.

In 2015, LSTC introduced a new structured ALE (S-ALE) solver option dedicated to solve the subset of ALE problems where a structured mesh is appropriate. As expected, recognizing the logical regularity of the mesh brought a reduced simulation time for the case of identical structured and unstructured mesh definitions. It also comes with a cleaner, conceptually simpler way of model setup. This article gives a brief description on how to setup the S-ALE mesh.

Automated mesh generation.

S-ALE solver expects a regular, box-shaped rectilinear mesh. This regularity enables an automated mesh generation by the solver, instead of pre-processor. The whole process is simple and easy. All we need to do is to specify 1. Origin, 2. Local coordinate system and 3. Most importantly, mesh spacing along the three directions (two if 2D), through the keyword named *ALE_STRUCTURED_MESH. S-ALE solver, will read in the keyword, process it, and then generate all the nodes, elements during the initialization phase. This process is fully automated, uninterrupted and without the need of user intervention.

This practice is different from most of other Finite Element solvers and new to most of our LS-DYNA users. WHY would we want to do that? There were several reasons.

1. **Convention Compliance.** The solver utilizes faster, more efficient algorithms based on the regularity of the mesh, through a set of conventions. For example, when numbering the elements, it swipes through x first, y next, and z last. Following this convention, switching between IJK index and element ID becomes a simple calculation without any condition check: eleID=K*Ny*Nx+J*Nx+I. The easiest way to ensure that all meshes feed into the solver follow these convention, is to let the solver take full control during the whole process of mesh generation. Ironically, by putting all burdens on the solver, we avoid all the trouble of generalizing, publicizing, enforcing, error-checking these internal conventions.

2. **Efficiency.** This streamlined approach provides us with savings on several aspects. The "normal" way for a Finite Element solver to get mesh involves three steps. First, geometric data to mesh, through a mesh generator (pre-processor). Next, mesh output by mesh generator is then included in keyword and read in by the solver. Finally, the mesh is processed and translated into the internal mesh database by the solver. Now we only have one step, the geometric data is read in by the solver and immediately translated into the database. All the memory, diskspace used in the three steps are no longer needed, simply because all mid-steps are skipped.

   For a big model, say one that has 100 million elements, not only do these savings become significant, but also crucial to make the model runnable. As S-ALE elements are not generated until each processor has its own subdomain hence its S-ALE elements block only, the huge memory required by processor #0 to build its elements database and perform the domain partition is no longer needed. Because of this, S-ALE could run far much larger model than ALE when using MPP.

3. **Less Error-prone.** This process could ensure a clean mesh with much less user mistakes. During the author's 18 years work at LSTC then Ansys, he has seen so many different user mistakes on mesh. And as ALE mesh and Lagrange mesh are on top of each other, some mistakes are, if not possible, very difficult to find. For example, the ALE mesh and Lagrange mesh are aligned at certain surface. The user thought he has ensured that ALE and Lagrange meshes are not sharing any node. And we checked some, they were all good. We then switched our focus onto other parts of the input deck and tried and tried and tried… And guess what, who would think ahead of time that there is one node skipped our scrutiny? And even if we thought about that, how could we single out that one node from like a thousand nodes, without help of some specifically written scripts? Another time we even had two sets of ALE mesh, identical, on top of each other. Do not even ask how it happened. The run was all OK, except for that the fluid keeps "leaking" out of the container. Not until half a day later, just by coincidence, the mistake revealed itself. I had to use the word "revealed itself" as I did not give even a tiny little thought on the mesh. Now all those mistakes are gone. More importantly, we could focus more on studying the model and the physics.

Offsetting and Rotating the Mesh

The keyword *ALE_STRUCTURED_MESH is used to take the user-provided geometric data. That is 1. Origin; 2. Local coordinate system; 3. Mesh spacing. 1 and 2 are optional. If they are left blank, we are constructing the mesh with neither any offset, nor any rotation. And yes, that is what NID0 (origin) and LCSID (local coordinate system) do. They specify some translation and rotation motion to the mesh. In that sense, it is kind of like *INCLUDE_TRANSFORM. You could

construct a S-ALE mesh, and then offset and rotate it to align this mesh with the other Lagrange structure in the model. BUT there is more to that.

This S-ALE mesh construction is not only at the first step, during initialization. Rather it is constructed at EVERY timestep, during the full course of the run. What does that mean? The mesh can move. And NID0 and LCSID control how it moves. Remember that origin and the axes are prescribed using nodes? Origin by 1 node and the local coordinate system by 3 nodes. If these four nodes are stationary, good; the mesh is stationary in space. But if some specific motion is provided to these four nodes, then the S-ALE mesh will move with these four nodes accordingly. This greatly simplifies the S-ALE mesh motion and more importantly, much more efficient than mesh motion algorithm employed by generic ALE solver.

For example, we are modeling fuel tank sloshing. The tank moves horizontally forward and backward for a few cycles. S-ALE mesh needs to follow this translational motion. What we could do it to simply pick a Lagrange node from the fuel tank structure, a node where no big deformation is expected, and then set that node as the origin. Of course, we need to offset the nodal coordinates in the three *ALE_STRUCTURED_MESH_CONTROL_POINTS card as the following.

```
*ALE_STRUCTURED_MESH
$    mshid      dpid      nbid      ebid
         3         4     40001     40001
$    cpidx     cpidy     cpidz      nid0     lcsid
      1001      1002      1003         1       101
*ALE_STRUCTURED_MESH_CONTROL_POINTS
$     CPID      NONE      NONE       SFO      NONE      OFFO
      1001                                          0.5
$               x1                x2                x3                x4
                 1           &xminale
              &nxale           &xmaxale
*ALE_STRUCTURED_MESH_CONTROL_POINTS
$     CPID      NONE      NONE       SFO      NONE      OFFO
      1002                                          0.5
$               x1                x2                x3                x4
                 1           &yminale
              &nyale           &ymaxale
*ALE_STRUCTURED_MESH_CONTROL_POINTS
$     CPID      NONE      NONE       SFO      NONE      OFFO
      1003
$               x1                x2                x3                x4
                 1           &zminale
              &nzale           &zmaxale
*NODE
       1            -0.5            -0.5             0.0       0       0
```

The full model is at https://ftp.lstc.com/anonymous/outgoing/hao/sale/models/tankslosh2/. By setting NID0 to Node 1 which is a Lagrange node on the tank, we attached the S-ALE mesh to the tank. Now S-ALE mesh follows whatever translational motion of Node 1 takes. Same thing for

rotation, if we pick three orthogonal nodes on the tank and assign the resulting local coordinate system to the S-ALE LCSID, the S-ALE mesh rotates in the exact same pattern as the tank.

Mesh Spacing.

For a 3D problem, we need to construct a box shaped rectilinear mesh. So what is the least information we need to generate that box? Divide and conquer. Let us deal with one dimension at a time. And then repeat it three times. In 1D, mesh become a line containing some number of segments. The minimum information we need is 1. Starting point; 2. Ending point; 3. Number of segments. We provide this information through the keyword *ALE_STRUCTURED_MESH_CONTROL_POINTS. Below we specify a 1D mesh containing 10 evenly distributed elements (11 nodes) spanning from -0.5 to 0.5.

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
CPID 101							
N1 1		X1 -0.5					
N2 11		X2 0.5					

By the way we have an implicit convention here. Abscissa and ordinates must be ascending. Let us assume this is our intended mesh spacing along x-axis and y-axis. We only need another card to construct the z-axis mesh.

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
103							
1		0.0					
11		1.0					

And now let us construct the box mesh by putting these three _CONTROL_POINTS into the *ALE_STRUCTURED_MESH card.

*ALE_STRUCTURED_MESH							
MSHID 1	DPID 11	NBID 10001	EBID 20001				
CPIDX 101	CPIDY 101	CPIDX 103	NID0	LCSID			

Job done. We now have a box shaped S-ALE mesh which spans from (-0.5,-0.5,0.0) to (0.5,0.5,1.0) containing 1,000 elements and 1331 nodes, with their IDs starting from 10001 and 20001, respectively. The generated mesh has a mesh ID of 1 and is assigned with a Part ID of 11 (DPID). Simple, right? But is it enough for all applications?

By taking controls away from users, the keyword should be designed to cover all possibilities. This means no matter how complicated the mesh spacing is, the design should be able to provide a mechanism to describe and process it. Now let us go wild. No reason, someone wants the following mesh along z-direction. The sizes of these 10 element are generated randomly between 0.09 and 0.11. By executing ***r=0.02*rand(1,10)+0.09*** in Octave (free version of Matlab), It comes out to be [0.099215 0.108612 0.095970 0.099048 0.103958 0.102598 0.090284 0.096290 0.096238 0.094896]. Let us keep the starting point unchanged at 0.0, the ending point is now at 0.98711 by doing a ***sum(r)***. But that is not helping. The elements in between these two nodes are assumed to be of evenly distributed, which is not true. So how do we do it? A little cumbersome but still doable. Let us get the nodal coordinates of these 11 nodes first. The first node is at 0.0. The rest 10 nodes are [0.099215 0.207827 0.303798 0.402846 0.506803 0.609401 0.699685 0.795975 0.892213 0.987110] by doing a ***cumsum(r)***. Now the ugly part. We enter them one by one into the *ALE_STRUCTURED_MESH_CONTROL_POINTS card.

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
103							
1		0.0					
2		0.099215					
3		0.207827					
4		0.303798					
5		0.402846					
6		0.506803					
7		0.609401					
8		0.699685					
9		0.795975					
10		0.892213					
11		0.987110					

Tedious it is, right? To our defense, this is only to accommodate the extreme case. With help from some simple scripts and pre-processors, we believe that the workload is still somewhat manageable on the user's side. Plus we do not expect it from happening too often.

The design of this keyword should be relatively easy to understand now, after covering the two extreme cases. One is 10 elements evenly distributed between two nodes; another is 10 completely random-sized elements. There are lots of possibilities in between these two extremities. Let us study one to introduce the concept of regions.

One might want to divide the line into several regions and choose to have different mesh densities in these regions. Say we want to have three regions: 3 elements in [0.0, 0.3]; 8 elements in [0.3, 0.7]; and another 3 elements in [0.7, 1.0]. The reason behind the finer mesh in the middle,

let us say, is to capture the detonation process of the high explosive placed at z=0.5 more accurately.

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
103							
1		0.0					
4		0.3					
12		0.7					
15		1.0					

As shown above, the design is based on the concept of "regions". Two points define a region; and the mesh can contain as many regions as one wants. Unless progressive meshing is specified, inside a region the mesh is always evenly distributed. And in case of progressive meshing, the mesh size inside a region is gradually increased (or decreased).

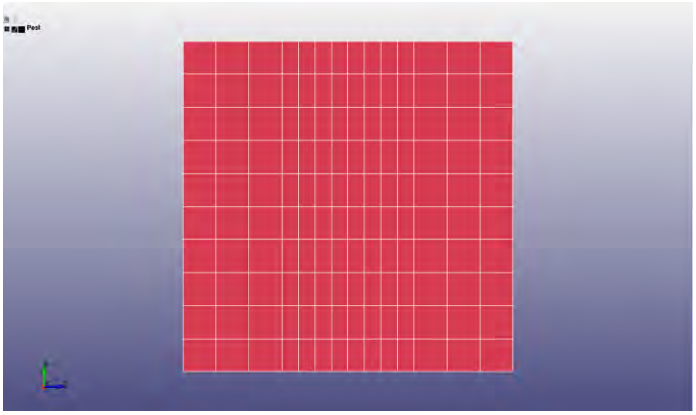


Fig 1. Z-direction mesh has 3 regions with 2 different mesh densities.

We used two mesh densities in the above mesh. An element size of 0.1 in the below and upper; 0.05 in the middle region. Most users will not stop there as we all know a sudden change in mesh density triggers wave reflection. And we DON'T want that. So is it possible to make a smooth transition between regions?

Progressive Mesh Spacing

The answer is yes. And there are several ways to do that. The first way puts all burdens on the user. Along a direction, one could design a line mesh containing several regions, with different mesh densities; and choose to have gradually increasing/decreasing element size in some regions. And then by using some pre-processor, or some scripting language like pyhton or matlab, or even hand calculation, one could derive nodal coordinates of all these nodes. And then list all these coordinates under the *ALE_STRUTURED_MESH_CONTROL_POINTS card.

The second one is to use "Ratio" option in the _CONTROL_POINTS card. Remember "region"? A region is composed of any two consecutive entries under the _CONTROL_POINTS card. Let us revisit the CPID #103 in the example we studied in the last section.

*ALE_STRUCTURED_MESH_CONTROL_POINTS						
103						
1		0.0				
4		0.3				
12		0.7				
15		1.0				

CPID #103 contains 3 regions, defined by 4 entries. First region is from entry #1 (1,0.0) and #2 (4,0.3); second from #2 (4,0.3) and #3 (12,0.7); third from #3 (12,0.7) and #4 (15,1.0). The mesh size in region 2 is one half of those in region 1 and 3. Now let us smoothen the transition from region 2 to region 3 with a ratio of increase of 0.1 from left to right. This means that in region 3, the 2nd element size is 1.1*dl, the 3rd 1.1^2*dl, assuming the 1st element size is dl. That leads us to the following equation: dl*(1.1^0+1.1^1+1.1^2)=0.3. From high school algebra, (1-x^n)/(1-x)=sum(1+x+x^2+…+x^(n-1)), the above equation could be simplified to dl*(1-1.1^3)/(1-1.1)=0.3 ➔ dl=0.090634. The three element sizes are 0.090634, 0.099698 and 0.10967, respectively. Compared with the element size of 0.05 in region 2, it appears the transition is not that smooth. So let us try using a ratio of 0.5. By repeating the above process, we get a new set of three sizes of 0.063158, 0.094737 and 0.18947. Still a little bit off as 0.063158/0.5=1.26 which means a ratio of 0.26. Then we try some numbers between 0.1 and 0.5 and it turns out a ratio of 0.4 yields a set of [0.068807, 0.096330, 0.13486], which we think good enough. Now let us add that into the entry #3 which precedes the region 3.

*ALE_STRUCTURED_MESH_CONTROL_POINTS						
103						
1		0.0				
4		0.3				
12		0.7		0.4		
15		1.0				

The convention here is, a mesh expansion ratio of 0.4 will be applied at the region beginning at the 12th node and ending at the 15th node. We require the expansion ratio to be put in the entry where the region starts at.

Let us deal with region 1 to region 2 transition now. Simply put, all we need to do is to put a negative expansion ratio of -0.4 at the first entry as follows. And the three sizes are, from left to right, [0.13486, 0.096330, 0.068807].

*ALE_STRUCTURED_MESH_CONTROL_POINTS						
103						
1		0.0		-0.4		
4		0.3				
12		0.7		0.4		
15		1.0				

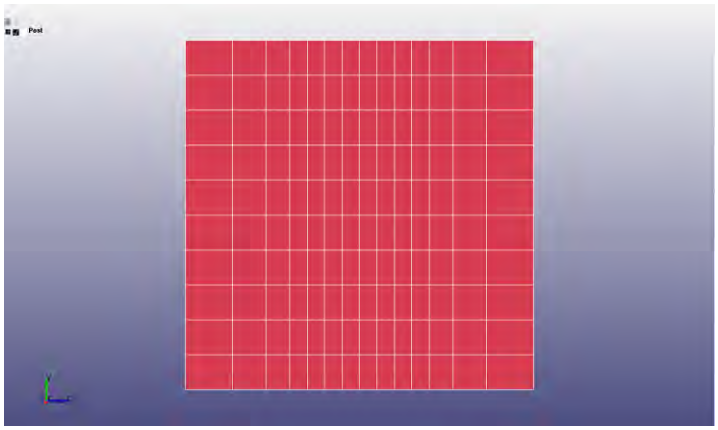


Fig 2. Progressive mesh along z-direction with a ratio of 0.4

The "ratio" parameter used here, is positive in case of expansion and negative in reduction. A special note here. A ratio of 0.4 means each time the element size increased by 40%, i.e. $dl_{n+1} = 1.4 * dl_n$. But a ratio of -0.4 does NOT mean a decrease of 40% in size from left to right. Rather it means a size increase of 40% in the reversed direction, from right to left. We could illustrate this using equation forms. A decrease of 40% in size is, $dl_{n+1} = (1 - 0.4) * dl_n = 0.6 * dl_n$. While what we want is an increase of 40% from the (n+1)th cell to the nth cell, i.e. $dl_n = 1.4 * dl_{n+1} ==> dl_{n+1} = 0.71429 * dl_n$. A common misconception one could have, is that increasing 10% along the positive direction is the same as decreasing 10% along the negative direction. But 1/1.1 is not 0.9; it is 0.90909. This misconception led to some confusions among our users and the author hopes the above explanation could help in resolving it.

The prescription of expansion ratio mentioned above is our second method of making a progressive mesh spacing. It requires some user efforts. A few iterations might be needed to find the optimal expansion ratio to provide us with a smooth transition. It works but lacks user-friendliness the authors longs for.

At the beginning of this year, 2021, one ACE expert from our ANSYS European team, came up with an idea. Instead of asking for ratio, why do not we let the user input the intended element size at certain control points? And then we could do all the tedious calculation and iterations internally to provide a smoothed transitioning mesh. And here comes the option 3. The ICASE option.

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
103		ICASE					
1		0.0					
4		0.3					
12		0.7					
15		1.0					

Currently ICASE can be either 1 or 2. More options might be added in the future to deal with more special cases per users' request.

ICASE=1: Let us illustrate its usage with CPID #103. What we prescribe now is the element size, dl, instead of ratio. In region 2, from node 4 to 12, we have an element size of 0.05. Try with the following setup:

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
103		1					
1		0.0					
4		0.3		0.05			
12		0.7		0.05			
15		1.0					

The mesh comes out a little off from what we expected.

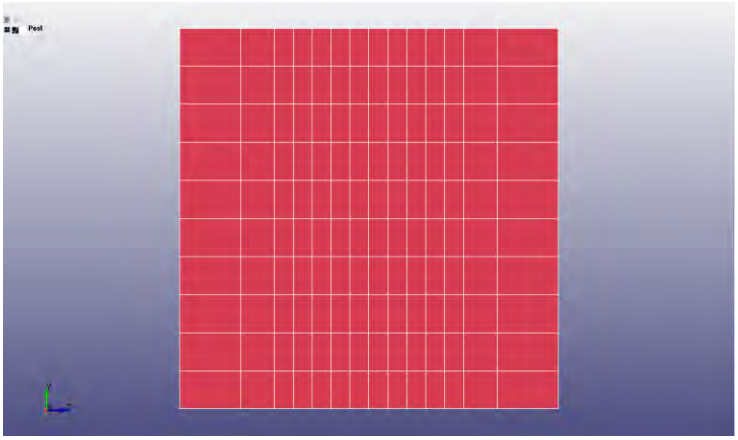


Fig 3. Progressive mesh along z-direction using ICASE=1, “not quite right”.

We could see in Fig 3 that the 3rd element is of a size of 0.05, also the 12th element. What we intended to do, is to prescribe the element size from the 4th element to the 11th element. So a little modification:

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
103		1					
1		0.0					
5		0.35		0.05			
11		0.65		0.05			
15		1.0					

And now everything is in order. The three cells have a size of 0.134047, 0.0964929 and 0.0694597, respectively. The reason is as follows. We are prescribing the element size at a node. This requires us to assign element sizes at both elements connecting to that node. In this example, the number of nodes along one direction is only 10. In cases of more nodes used, this small discrepancy will go unnoticed.

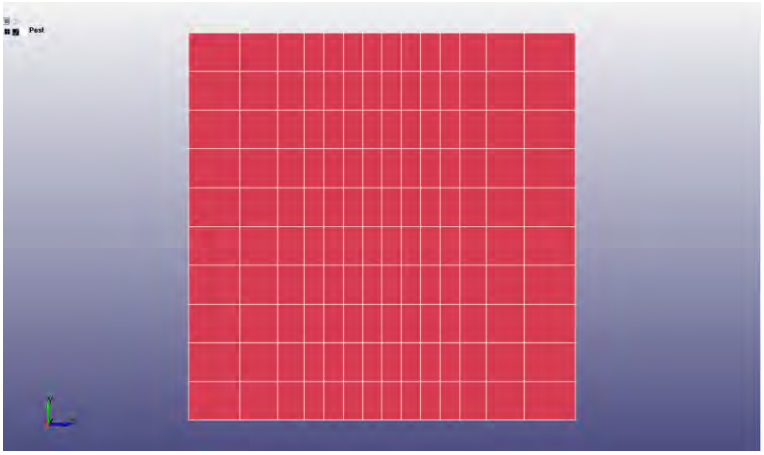


Fig 4. Progressive mesh along z-direction using ICASE=1.

There are also other ways to obtain this progressively spaced mesh. For example, we could also set the element size to be 0.13486 at the first and the last cell.

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
103		1					
1		0.0		0.134047			
4		0.4					
12		0.7					
15		1.0		0. 134047			

And we will have the exact same mesh spacing as shown in Fig 4.

ICASE=2: The idea behind ICASE=2 is to build the progressive spaced mesh starting from a point and extend the mesh to the left and the right. To use this option, we need to first pick a "base node" and put it at a location by specifying its coordinate. Let us continue our play with CPID #103. This time we pick a "base node" and then extend the mesh into left and right. We put node #5 at x=0.35 and ask for a cell size of 0.05. We extend the mesh to left with mesh size gradually increased until it reaches the 1st node with a cell size of 0.134047. On the right side, we keep the cell size unchanged until the 11th cell and then starting from the 12th cell, gradually increase the mesh size from 0.05 until it reaches the right end with a cell size of 0.134047.

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
103		2					
1				0.134047			
5		0.30		0.05			
11				0.05			
15				0.134047			

We would end up with the exact same mesh as in Fig 4. Real case scenario we would not use such awkward number as "0.134047". Pick node #8 to be the "base node" with a coordinate of 0.5 and change the cell size to 0.12 for the leftmost and rightmost element. We get the following mesh.

*ALE_STRUCTURED_MESH_CONTROL_POINTS							
103		2					
1				0.10			
5				0.05			
8		0.50		0.05			
11				0.05			
15				0.10			

Along the z direction, mesh spans from 0.05763 to 0.942366. As shown below.

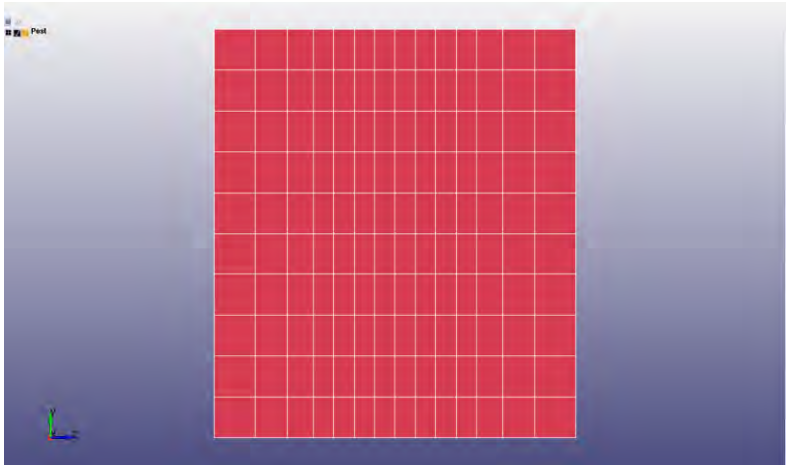


Fig 5. Progressive mesh using ICASE=2. First and last element have a size of 0.1.

For more examples on ICASE usages, please refer to LS-DYNA user's manual.

Ending Remarks

LS-DYNA ALE module has been known for its steep learning curve. Partially it was because setting up Eulerian models are intrinsically different from Lagrange models. But the design of ALE keyword cards, for sure, has caused quite a lot of confusions among our users, new and experienced.

To prompt LS-DYNA ALE usages, Structured ALE solver introduced a new, user-friendly, streamlined three-step setup. We hope this effort could help users, new or old, to perform their work more efficiently and smoothly.

On Structured ALE Mesh Trimming

Hao Chen, Ansys

LS-DYNA ALE has been widely used to simulating moving fluids interacting with structures. Unlike CFD, the focus is rather on the structure response under dynamic loading from fluids, than the fluids' motion. Fluids are agitated by a high pressure gradient; and then hit the structure, carrying a large momentum. The key in successfully capturing the physics lies in the fluid-structure interaction algorithm. It needs to accurately predict the peak of pressure loading during the impact, which is characterized as a momentum transfer process. This request could only be fulfilled by a transient analysis with a penalty-based coupling between fluids and structure.

In 2015, LSTC introduced a new structured ALE (S-ALE) solver option dedicated to solve the subset of ALE problems where a structured mesh is appropriate. As expected, recognizing the logical regularity of the mesh brought a reduced simulation time for the case of identical structured and unstructured mesh definitions. It also comes with a cleaner, conceptually simpler way of model setup. This article gives a brief description on the mesh trimming feature.

Mesh regularity.

S-ALE solver expects a regular, box-shaped rectilinear mesh. This regularity enables an automated mesh generation by the solver, instead of pre-processor. And more importantly, this mesh regularity enables a simpler algorithm leads to a reduced simulation time and memory usage. It comes with a price though.

In the generic ALE solver, ALE mesh is generated by the user and could be of arbitrary shape. Users have the flexibility to mesh the ALE domain to better fit their needs. For example, to model a spherical blast one could generate a spherical ALE mesh domain. Another example is to model fuel sloshing in a rigid body container. One could choose to make the ALE outer surface conform to the Lagrange container. And then constrain the normal directional flow by using *ALE_ESSENTIAL_BOUNDARY.

The above two examples are without any fluid structure interactions. Now let us go to cases with FSI as these are what S-ALE targets to solve. Still on fuel sloshing problem, what we care is to see how the fuel tank deforms under a sudden fluid impact so container has to be flexible and FSI is needed. The shape of the container, for example, is somewhat very irregular. Let us say its height on the left side is much less than the right side, only ¼. To reduce the number of ALE elements, one might want to mesh the left side with less ALE elements. Just to make sure it covers the Lagrange container with several extra cushion layers of elements. As ALE runs are comparatively much more expensive, stinginess becomes a virtue.

From the above example, one could understand the motivation behind the development of mesh trimming feature. Its major objective is to keep the element cost low, by trimming the mesh to better fit the domain of interest. In most cases, to better fit the geometry of Lagrange structures coupled to ALE fluids. Simply put, mesh trimming is to remove the elements far away from the structure or some geometric entities.

A first glance.

Let us see how it is done. Below is the format and description of the keyword *ALE_STRUCTURED_MESH_MESH_TRIM. There could be multiple trimming cards which are executed in the order of their appearances.

*ALE_STRUCTURED_MESH_MESH_TRIM							
MSHID	OPTION	OPER	OUT/IN	E1	E2	E3	E4

MSHID is the Mesh ID of the S-ALE mesh, defined in *ALE_STRUCTURED_MESH card. OPTION is to provide geometries to trim the S-ALE mesh and could be chosen from the following six: PARTSET, SEGSET, PLANE, CYLINDER, BOXCOR, BOXCPT and SPHERE. OPER has two choices: 0 to trim or 1 to keep. IN/OUT is to prescribe inside or outside; 0 – outside, 1 – inside. E1,E2,E3,E4 are to provide the geometric data and have different meanings for each different OPTION.

A picture is better than thousand words. The figure below shown a S-ALE mesh trimmed by using OPTION=”SEGSET”, OPER=”0” (trim) and OUT/IN=”0” (outside) with an offset of “10”. The segment set is the tube-like surface of red color. S-ALE mesh is on top and around the red tube, transparent and of a color of light blue.

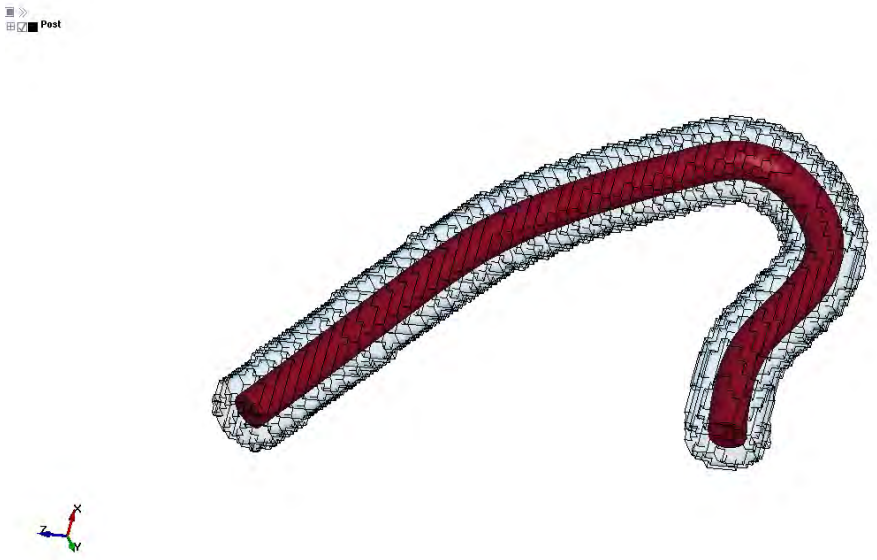


Figure 1. S-ALE mesh trimmed around “glue” (the red tube)

This model is done by Mukul Atri, Ansys ACE India, mukul.atri@ansys.com. The input deck could be found at https://ftp.lstc.com/anonymous/outgoing/hao/sale/models_R121/trim/tube_trim.k

Let us go through its setup to illustrate the usage of mesh trimming card. The initial mesh is a box mesh of 75x63x68 elements spans from (-150,-125,-135) to (150,125,135). There are two ALE multi-materials in this mesh – "glue" and "vacuum". The segment set #2 is created from the outer surface of a Lagrange dummy part; And used later to do the volume filling for ALE fluid "glue". We want to study how "glue" flows under compression (not included in this model for simplicity).

```
$=========================================================================
=
*ALE_STRUCTURED_MESH
$   meshid      dpid      nbid      ebid
         1         2
$    cpidx     cpidy     cpidz      nid0     lcsid
         1         2         3
*ALE_STRUCTURED_MESH_CONTROL_POINTS
$     cipd
         1
$        n                 x
         1              -150
        76               150
*ALE_STRUCTURED_MESH_CONTROL_POINTS
$     cipd
         2
$        n                 x
         1              -125
        64               125
*ALE_STRUCTURED_MESH_CONTROL_POINTS
$     cipd
         3
$        n                 x
         1              -135
        69               135
$=========================================================================
=
*ALE_STRUCTURED_MULTI-MATERIAL_GROUP
$# mmgname       mid     eosid                                          pref
  glue          1001      1001
  vacuum        1002
*MAT_ALE_VISCOUS
      1001  1.000E-5   -1.0E10    2.0E-6       8.0   2.16E-2             0.201
*EOS_LINEAR_POLYNOMIAL
      1001       0.0    1000.0       0.0       0.0       0.0       0.0       0.0
       0.0       0.0
*MAT_VACUUM
      1002   1.0E-08
$=========================================================================
=
```

After setting up the above cards, we have a box mesh of 321,300 S-ALE elements, as shown below. There is quite some waste. All space not taken by "glue" (which occupies the same space as the red Lagrange tube) is going to be filled with "vacuum". Most of the space, "glue" will never flow into it. There

is no point to keep all those elements far away from “glue”. So using a box mesh mostly filled with vacuum is really something we strongly dislike.

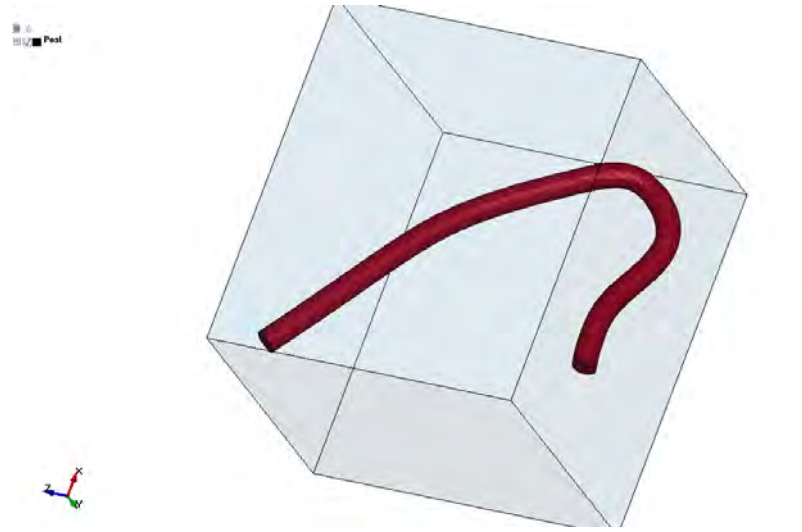


Figure 2. S-ALE box mesh untrimmed

To eliminate this waste, it is easy and straightforward. Simply adding the card below.

```
*ALE_STRUCTURED_MESH_TRIM
$    mshid    option      oper    out/in     segid    offset
         1    SEGSET         0         0         2      10.0
```

What it does is to remove S-ALE elements far away (10.0) from the outer surface of the red tube. The number of S-ALE elements is reduced to 14,173 from 321,300. Impressive, right? We would not expect the cost saving is always this significant, but still for special cases it could be quite surprisingly good.

*ALE_STRUCTURED_MESH_TRIM

Let us describe the design of *ALE_STRUCTURED_TRIM to get a better understanding of its usage.

*ALE_STRUCTURED_MESH_MESH_TRIM							
MSHID	OPTION	OPER	OUT/IN	E1	E2	E3	E4

1. Multiple entries allowed. Processed in the order of their appearances.
2. Five basic geometries: PLANE, CYLINDER, BOXCOR, BOXCPT and SPHERE.
3. Complicated geometries provided using PARTSET or SEGSET.
4. Operation be either trim (0) or keep (1) (or call it untrim, bring the trimmed elements back from previous trims)
5. OUT/IN: To operate on the elements outside (0) or inside (1) of the geometry For PARTSET and SEGSET options, “outside” is defined as the region to which the segment normal points.

Most of the above points could be easily understand by our users. Maybe with one exception: One might wonder what the purpose of operation "keep" is. Let us illustrate its usage by using an airbag model. The full input deck could be found at the following directory. https://ftp.lstc.com/anonymous/outgoing/hao/sale/models/meshmotion/airbag1/

The S-ALE mesh is shown in the figure below. The mesh is composed of two boxes. The upper one has 7 layers of elements along z direction from 0.0315 to 0.2415; each layer contains 16x16 elements and spans from (-0.36, -0.36) to (0.36, 0.36). The bottom one has 3 layers from -0.0885 to 0.0315; each layer contains 6x6 elements and spans from (-0.135, -0.135) to (0.135, 0.135).

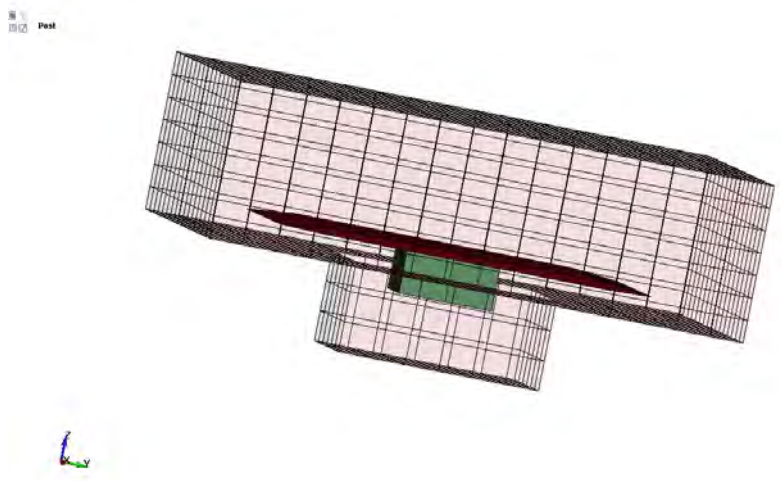


Figure 3. S-ALE mesh trimmed to have less elements at the bottom

As always, let us define the ALE mesh first.

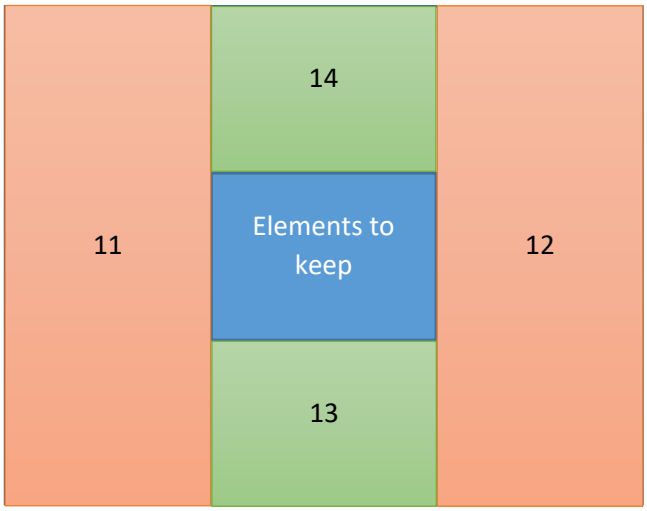
```
*ALE_STRUCTURED_MESH
$    mshid       pid      nbid      ebid
         1        11    100001    100001
$     nptx      npty      nptz      nid0     lcsid
      1001      1001      1003       272      1001
*DEFINE_COORDINATE_NODES
      1001       272       254       233         1
*ALE_STRUCTURED_MESH_CONTROL_POINTS
      1001
$                 x1                  x2
                   1               -0.36
                  17                0.36
*ALE_STRUCTURED_MESH_CONTROL_POINTS
      1003
$                 x1                  x2
                   1             -0.0885
                   4              0.0315
                  11              0.2415
```

Now we need to trim the mesh at the bottom away from the center. The most suitable choice is to use BOXCPT option to define the geometry. For now, let us forget about the "keep" operation and do the trim with "trim" operation only. The bottom mesh starts from node [6, 6] to node

[12, 12] in the xy plane. So this means we need to trim 4 times, each time with a box of elements. One way to do it is as follows.

```
*ALE_STRUCTURED_MESH_TRIM
$    mshid    command      oper     out/in      boxid
         1     BOXCPT                             11
         1     BOXCPT                             12
         1     BOXCPT                             13
         1     BOXCPT                             14
*DEFINE_BOX
$    boxid       xmin       xmax       ymin       ymax       zmin       zmax
        11          1          6          1         17          1          4
        12         12         17          1         17          1          4
        13          6         12          1          6          1          4
        14          6         12         12         17          1          4
```

These four boxes are shown in the figure below.



The result fits our expectations and the process seems OK. But somehow you have a gut feeling that there must be a better way to do it, right?

*ALE_STRUCTURED_MESH_TRIM is designed to do addition or subtraction between geometries. That is where “keep” operation comes into play. The above card could be simplified to:

```
*ALE_STRUCTURED_MESH_TRIM
$    mshid    command      oper     out/in      boxid
         1     BOXCPT                                1
         1     BOXCPT          1          1          2
*DEFINE_BOX
$    boxid       xmin       xmax       ymin       ymax       zmin       zmax
         1          1         17          1         17          4         11
         2          6         12          6         12          1          4
```

It contains two operations. The first is to trim any element outside (OUT/IN=0) of the box 1 at the top. After this, all elements at the bottom 3 layers are deleted. That is not what we wanted – we still need the center 6x6 elements at the bottom. So let us reverse the trim operation, but

only at those 6x6 elements. That is what the second card does here. It is to “keep” (untrim) the elements inside (OUT/IN=1) the box 2 which spans from node [6, 6] to [12,12] in the bottom 3 layers.

One can easily imagine the added flexibility it achieved by adding this “keep/untrim” operation. Most cases a complicated geometry could be described using addition/subtraction between 2 or at most 3 simple geometries. For a more detailed description of other options/flags in this keyword, please refer to LS-DYNA user’s manual.

Ending Remarks

LS-DYNA ALE module has been known for its steep learning curve. Partially it was because setting up Eulerian models are intrinsically different from Lagrange models. But the design of ALE keyword cards, for sure, has caused quite a lot of confusions among our users, new and experienced.

To prompt LS-DYNA ALE usages, Structured ALE solver introduced a new, user-friendly, streamlined three-step setup. We hope this effort could help users, new or old, to perform their work more efficiently and smoothly.