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).

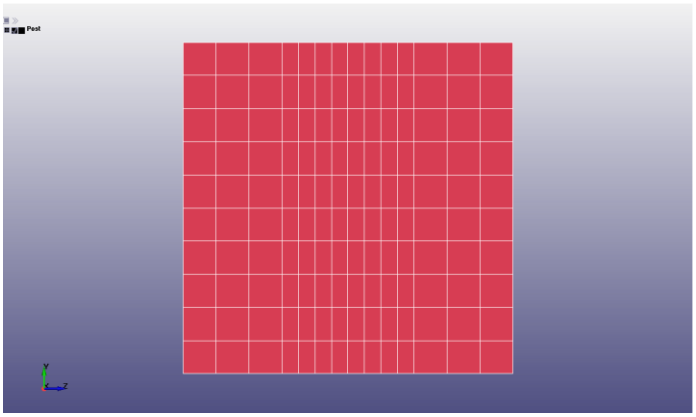

Fig 1. Z-direction mesh has 3 regions with 2 different mesh densities.

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					

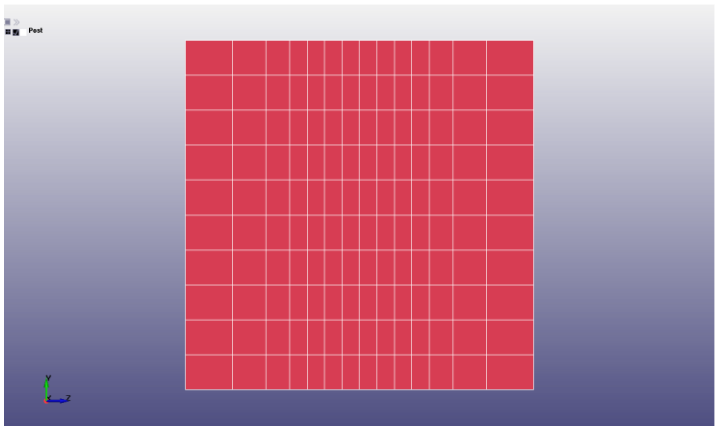

Fig 2. Progressive mesh along z-direction with a ratio of 0.4

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.

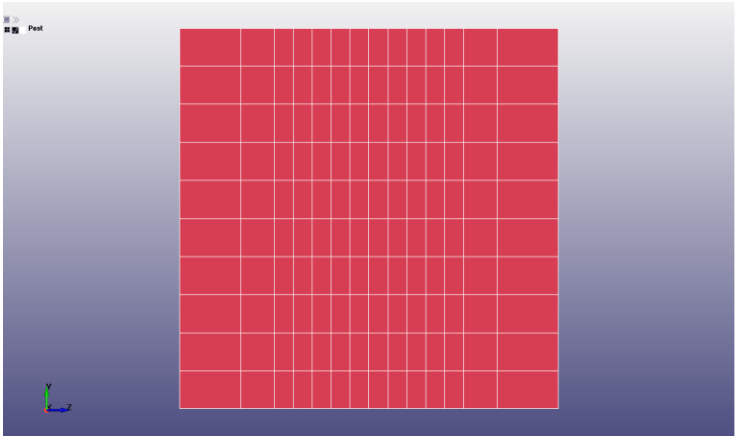

Fig 3. Progressive mesh along z-direction using ICASE=1, “not quite right”.

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.

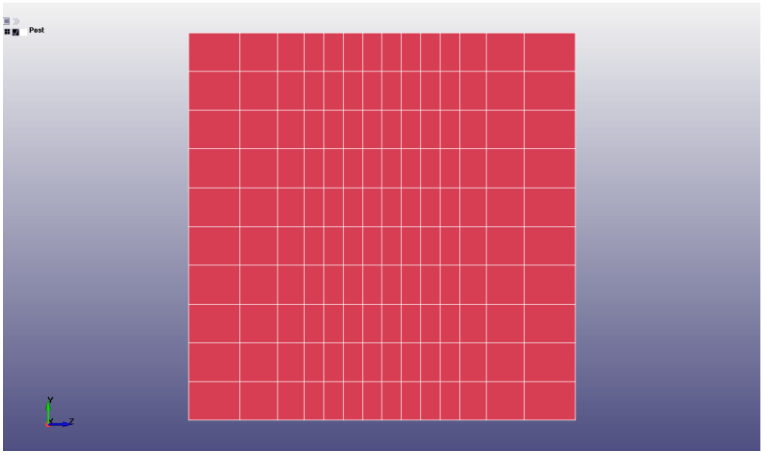

Fig 4. Progressive mesh along z-direction using ICASE=1.

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.

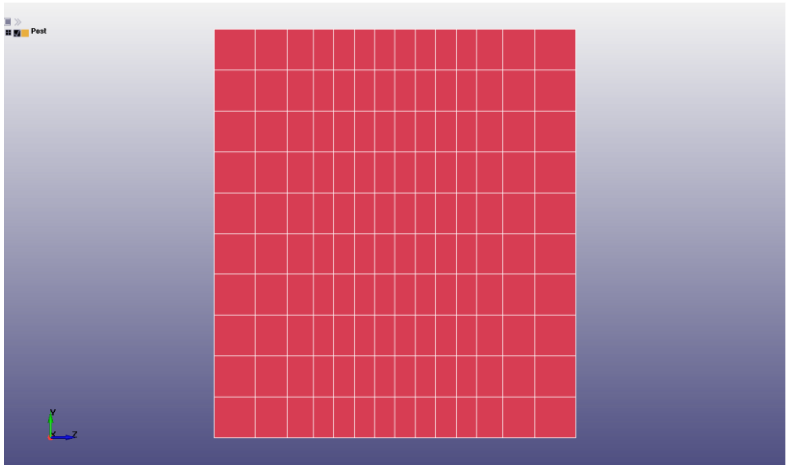

Fig 5. Progressive mesh using ICASE=2. First and last element have a size of 0.1.

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.