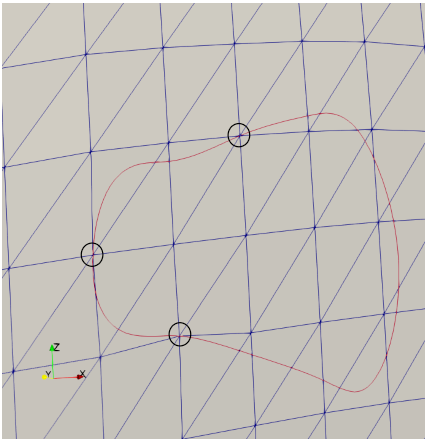
Update on Recent developments and future plans

## Sub Meshing

Last we discussed, I was still trying to successfully re-mesh tow surface elements that had been intersected by the interpenetration boundary curve. We were able to get a new mesh but agreed that the quality was not very good. First, we reduced the refinement of the boundary curve to only include the points where the boundary curve intersected in surface elements. We were still getting bad meshes because the boundary curve was very close to some surface nodes and causing very small elements.

This figure shows the original boundary curve we are using as a test case to fix some issues within the sub-meshing routine. The three circles indicate where the surface mesh (blue elements) has been adjusted so that a surface node lies on the boundary curve itself. This adjustment occurs when the boundary curve (red) is close to a surface node and results elements that have high aspect ratios and/or very small compared to the overall surface refinement (figure B). When the surface mesh is moved to the boundary curve, this situation is eliminated and more ideal elements are achieved (figure C).



Figure

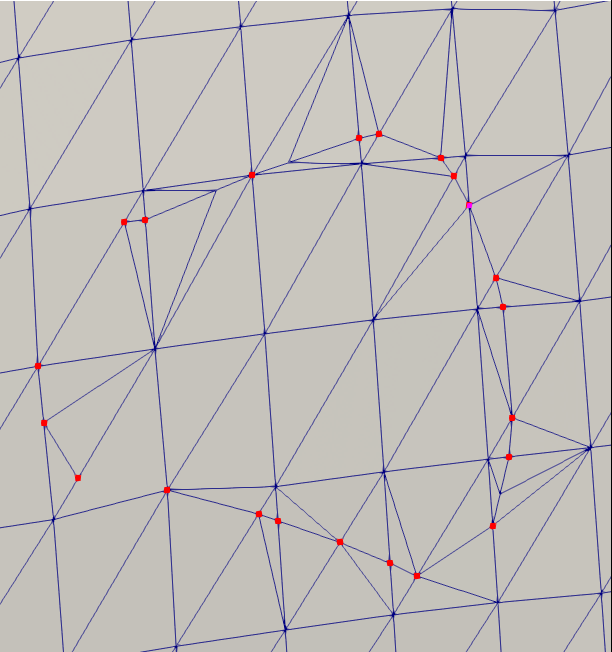


Figure C

A

B

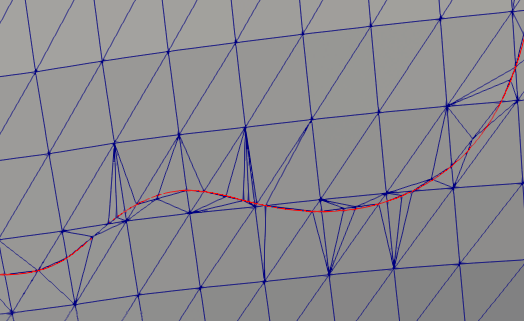


Figure B

We use the sub-meshing routine used by BetaMesh, which uses the Triangle sub-meshing library. The sub-meshing routine requires a list of boundary segments that the library uses to determine the required boundaries that must be included in the new mesh. The boundaries we give the algorithm include all surface element edges along with any interpenetration boundary curve segments that lie in or on the surface element and its edges. This is important because it requires that any edge of a surface element must be subdivided if intersected by a boundary curve segment before being included in the boundaries list sent to the sub-meshing routine. Our own "sub-meshing" routine accepts a surface element and conducts all checks and preprocessing required to call the Triangle sub-meshing routine. This includes sub-dividing the element edges and collecting all boundary curve segments that intersect the surface element. Our routine then calls the Triangle library function that creates a mesh from the boundary segments.

The figure C shows another problem. In the case where the boundary curve enters/exits a surface element through an element vertex (label A), the sub-meshing algorithm does not correctly sub-divide the surface element. In the example, there should be a line from A to B sub-dividing the element. However, this does not occur. I believed the algorithm was detecting the boundary curve intersection properly but was not programmed to sub-divide the element when one node of a boundary curve segment coincided with a surface node. Our "sub-meshing" algorithm was written before the surface node movement algorithm and was never refactored to account for this situation.

The flaw in the algorithm was in the check to verify if there were any edges intersected by a boundary curve segment. This check is made to collect any boundary curve segments that either lie inside or cross into the surface element being sub-meshed. When the check is made, the statement was true only if a segment had more than one intersection with a surface element edge. However, in the case where a segment enters a surface element through a vertex (A), there is only one edge intersection recorded. By changing the statement to be true for any number of edge intersections greater than zero, this situation is now caught by the if statement. The result is figure D.

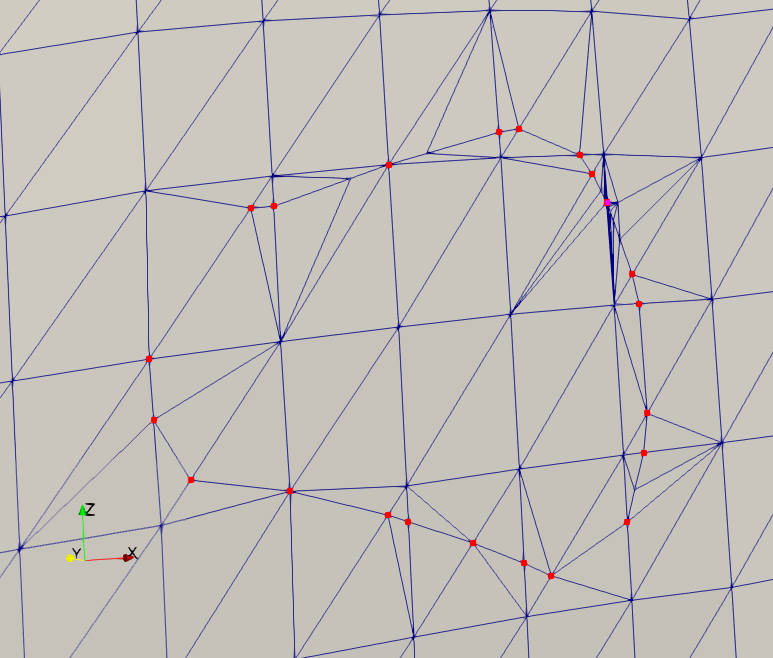


Figure D

The element that was lacking a dividing segment is now properly sub-divided. This is crucial because it correctly shows how the boundary curve sub-divides the surface element. It also ensures compatibility between surface elements that are being sub-divided by the boundary curve. However, this figure shows another problem. In the upper right region of figure D, the mesh has a very high refinement as well as having high aspect ratio elements. Initially I thought something was wrong with the parameters given to the Triangle library sub-meshing function. Upon further inspection, I found that there were two intersection points in very close proximity where the boundary curve crosses the surface element edge. Therefore, I added a small check to verify that boundary intersection points cannot be closer than the distance the tow surface nodes are allowed to move when surface nodes are close to the boundary curve. This is because the only scenario when two intersection points should be very close is when the boundary curve enters and exits a surface element very close to a surface element vertex, essentially cutting a corner of an element. If the distance between the two intersection points is larger than the distance a surface node is allowed to move, then it is safe to say that the corner of the element is realistically cut. If the nodes are close together than we assume that the nodes are duplicates, especially if they lie on the same element edge. The result of implementing this change is seen below. Another note is that I believe the reason the refinement in that area was not high in the same area in figure C has to do with not correctly identifying single edge intersection boundary segments. When we corrected that issue, the two intersection points in close proximity were both added as boundary nodes to the Triangle sub-meshing routine, requiring that both intersection points be included in the new element meshes. The mesh in the figure below is much better than previous meshes we have created in terms of relative refinement and high aspect ratio elements. It is true that some elements still have a higher aspect ratio than we would like but it is the best we can do without spending significant time altering the mesh on our own.

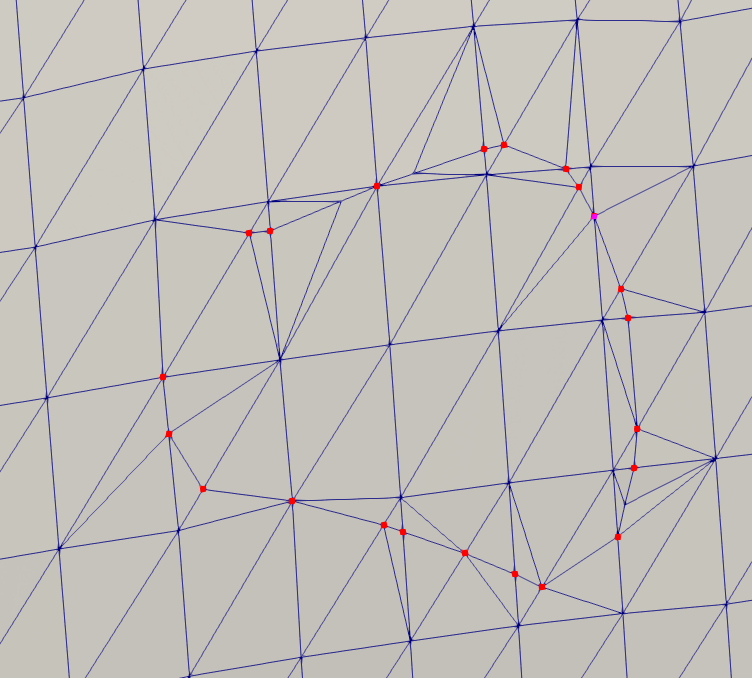


Figure E

## Interpenetration elements collection

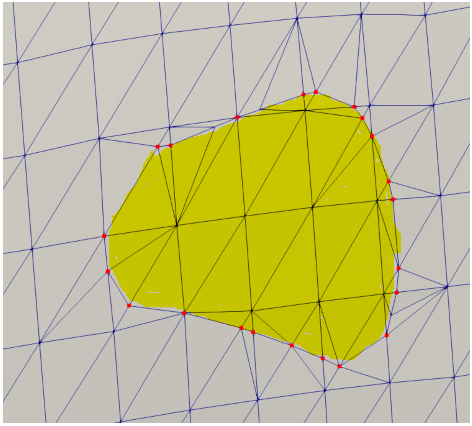


Figure F

Once the inclusion of all boundary curves into the tow surface mesh has finished, the elements contained in the boundary curve still need to be collected (yellow highlighted, figure F). The basic premise that will be used is that the regions that are interpenetrating contain less elements than the amount not interpenetrating. Therefore, if we can collect two groups of elements, one lying inside a boundary curve and one lying outside, and compare their sizes, the interpenetrating element group can be determined.

To collect the groups, an element (element 1) that shares an edge with the boundary curve (black line) is chosen. Then, the element that shares the boundary curve edge with element 1 (element 6) is also chosen and saved in a separate group. Then, an element that shares an edge with the first element (element 1) that is not on the boundary curve is chosen (elements 2 or 10). This element (starting with element 2) finds its remaining two elements that it shares an edge with and selects them (element 3 and an element not shown). Element 2 does not have an edge that lies on the boundary curve, therefore the two found elements can be added to the same group that element 2 belongs to and then individually checked. If instead element 10 had been selected, the two elements need to be checked to decide if the edge it shares with element 10w lies on the boundary curve (such as element 9). If the shared edge does lie on the boundary curve, then the element (element 9) is then added to the element group that 10 does not belong to (the same as element 6). In this manner, two element groups will be built so that their sizes may be checked. This algorithm is preliminary and will have to be developed for special cases that may arise, such as an element touching two separate boundary curves. Keith and I have discussed this method previously but have not been able to implement it yet. It is the last milestone before we move to creating models for analysis.

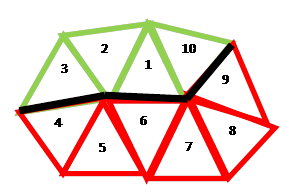


Figure G