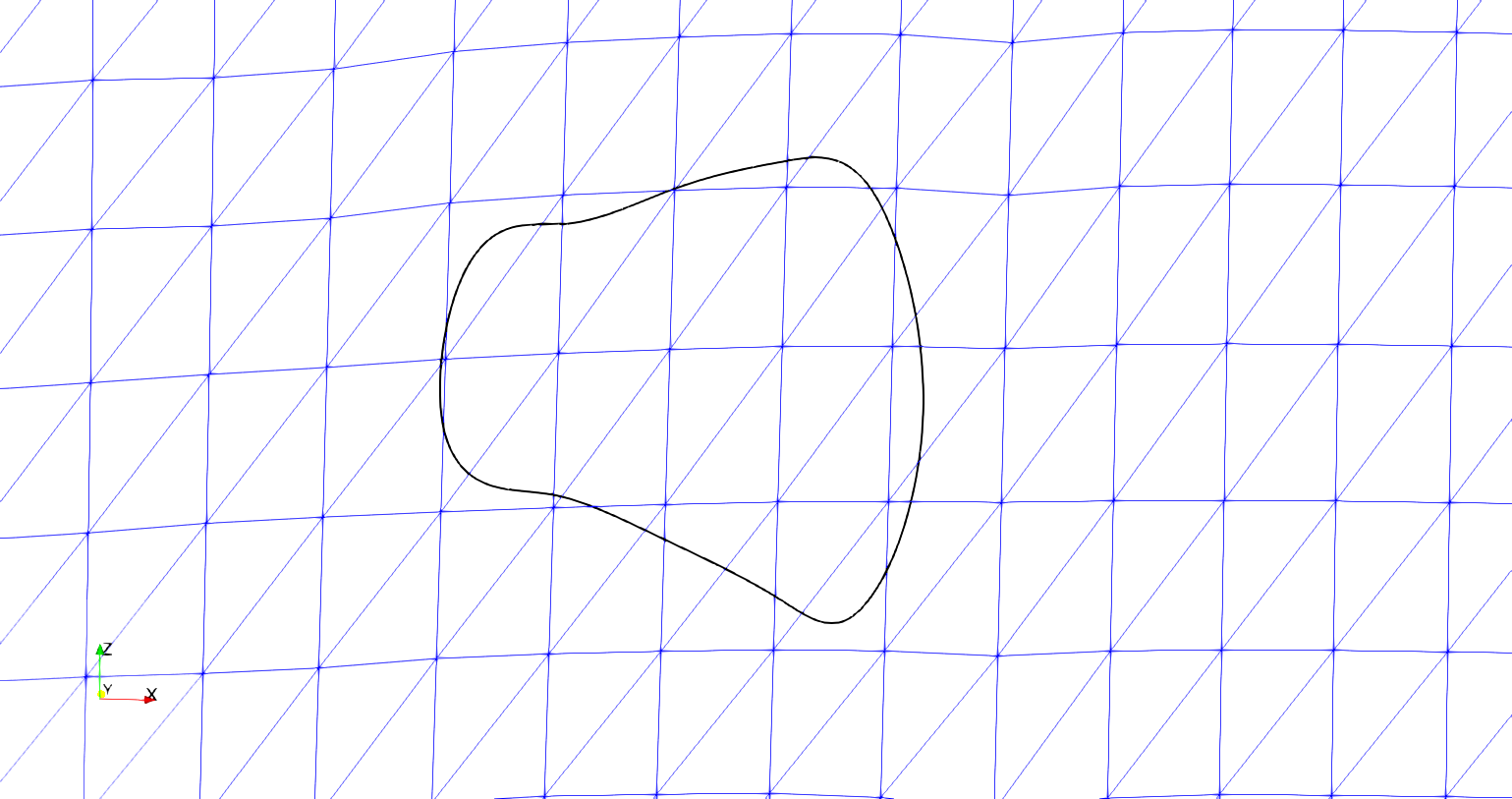
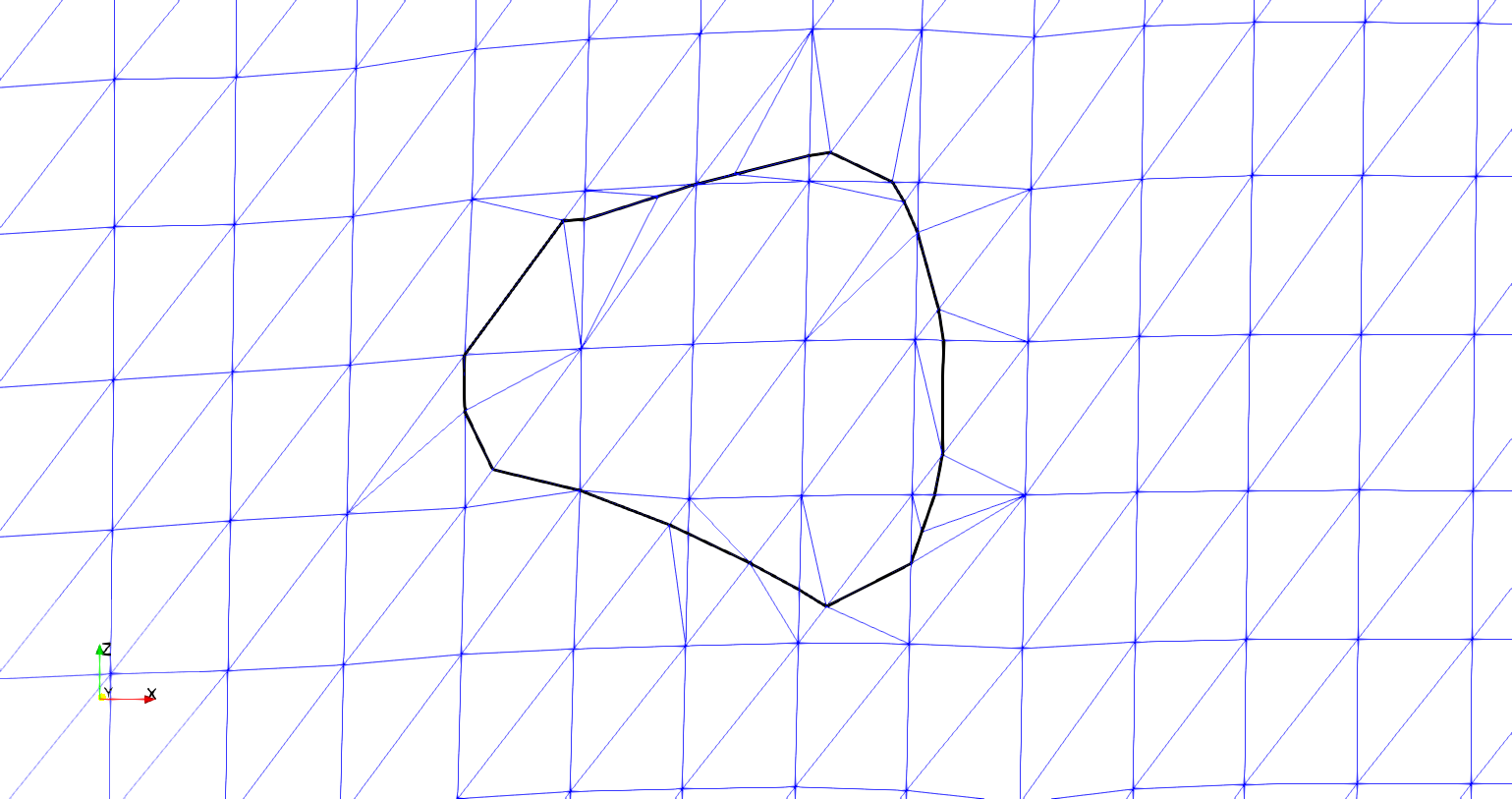
## Detecting surface elements intersected by a boundary curve

Once the boundary curves have been made unique and closed, the boundary curves are used to divide and sub-mesh the tow surface elements they intersect. Figure A shows an example of a boundary curve that is used to sub-divide the tow surface mesh and the result of the algorithm.

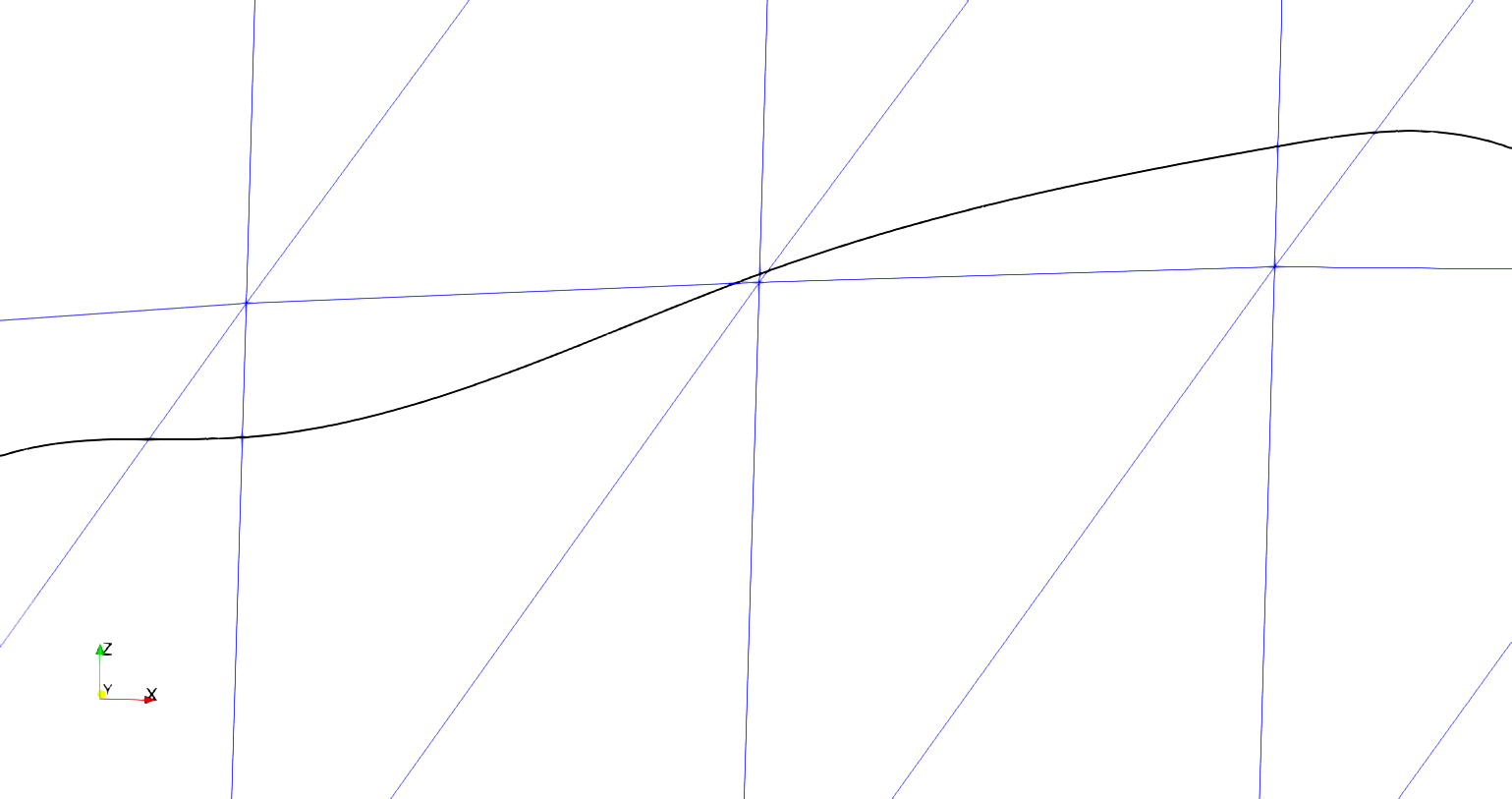


**Figure A: Boundary curve and sub-mesh result**



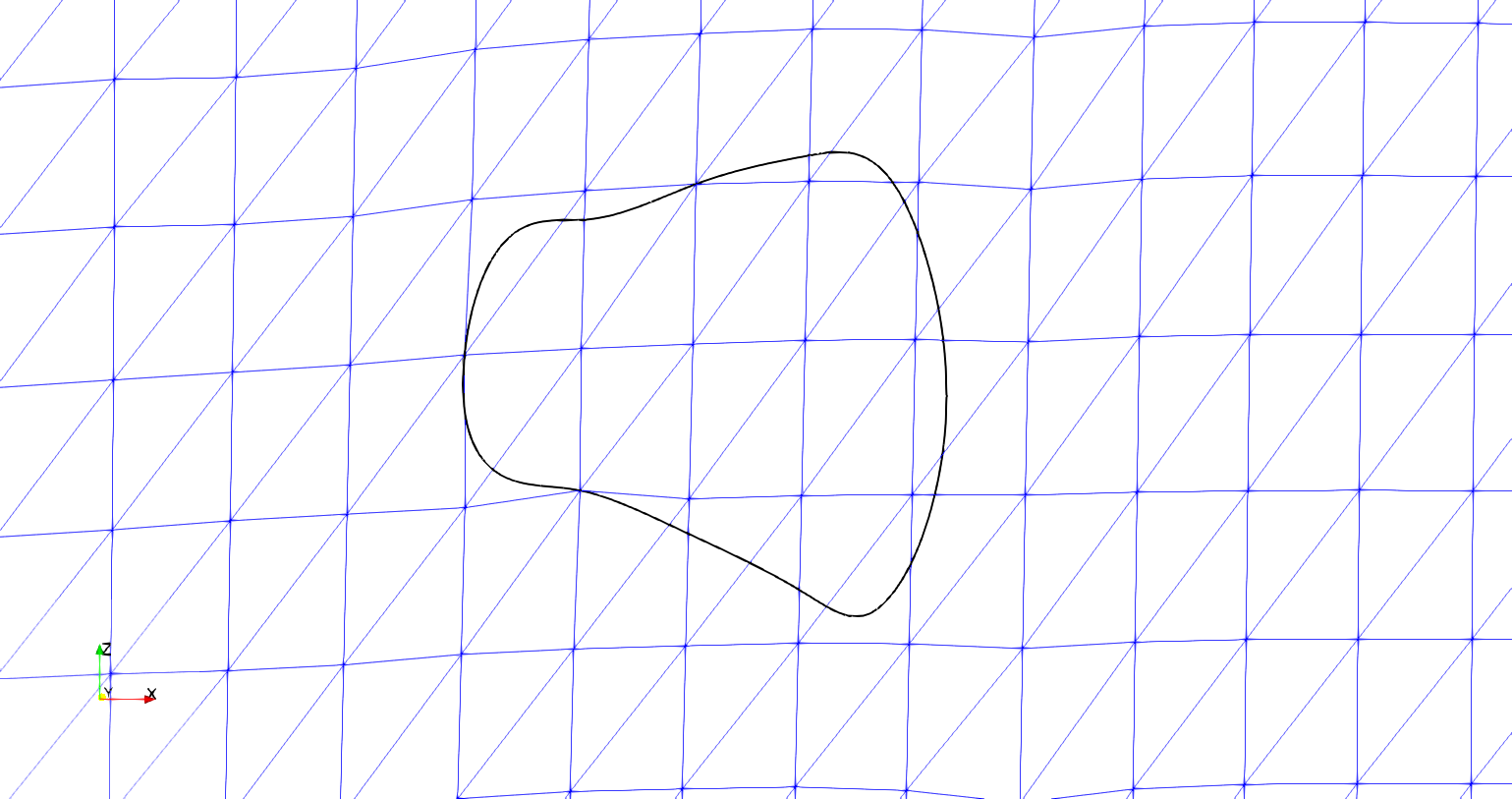
1. **Initial boundary curve and surface mesh**
2. **Embedded boundary curve and resulting mesh**

The curve must be represented in some form so that the interpenetrating elements from the surfaces can be corrected or removed. The chosen solution is to sub-divide the surface elements where they are intersected by the boundary curve and embed the boundary curve into the surface mesh. Once the boundary curve is embedded into both surface meshes, it becomes the common interface along which the two surfaces will be compatible. The first step is to move any surface node that is in close proximity to the boundary curve. Figure B shows a section of the upper part of the curve in figure A that is very close to a surface node on the surface mesh.



**Figure B: Section of boundary curve in close proximity to a surface node**

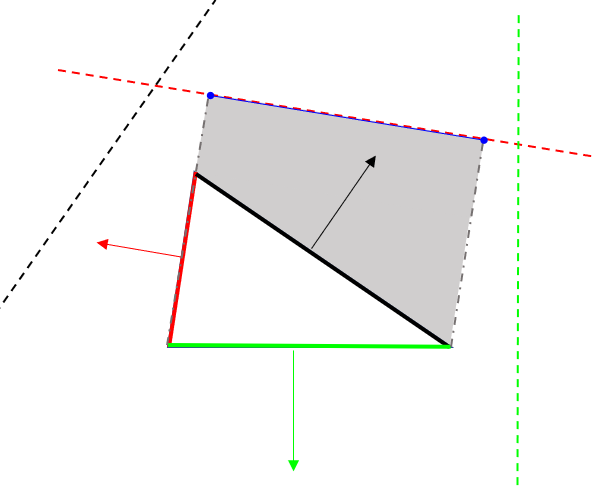
When the boundary curve comes close to a surface node, it affects the ability of the algorithm to re-mesh the intersected surface elements effectively. The solution is to move any surface nodes that are within a chosen tolerance to an existing node on the boundary curve. Because the relative refinement of the boundary curve is very high compared to the refinement of the number of surface elements it intersects (usually two orders of magnitude higher in number of elements), moving the surface node to the nearest existing boundary curve node is an acceptable solution. An alternative method is to calculate the closest location on the boundary curve to the surface node and create a point at this location. However, due to the high refinement of the boundary curve, this solution requires extra calculations that do not yield a better result. Instead, a k-d tree searching algorithm is implemented to find the nearest boundary curve node to a surface mesh node. The distance between the two points is calculated and compared against the established tolerance. A larger tolerance allows for surface nodes farther away from the boundary curve to be adjusted to lie on the curve. The result is shown in figure C.



**Figure C: Boundary curve with surface mesh nodes moved to the boundary curve**

The arrows in figure C indicate where surface nodes have been moved to the boundary curve and can be compared to figure A.a. The result eliminates complications that arise when the surface nodes are too close to the boundary curve such as high relative refinement regions and high-aspect ratio elements.

Once the surface nodes have been adjusted, the intersection points between the boundary curve and the surface mesh are calculated. The Separating Axis Theorem (SAT) is implemented to identify which elements are intersected and to refine the boundary curve where it intersects surface elements. The SAT starts by projecting the surface elements onto axes. The projection can be thought of as the shadow of the surface element on an axis. The axes are created by taking the normal direction of an element edge and creating an imaginary infinite line in the same direction. Figure D shows a projection of a triangle onto the red axis (RA) that is determined by the red edge (RE) and is parallel to the red edge normal (REN). The line segment along RA (labeled “Triangle Projection”) is the projection of the triangle onto the axis RA in Figure D. Three axes are identified, one corresponding to each side of the triangle.



**Figure D: Projection of a triangle on an axis**

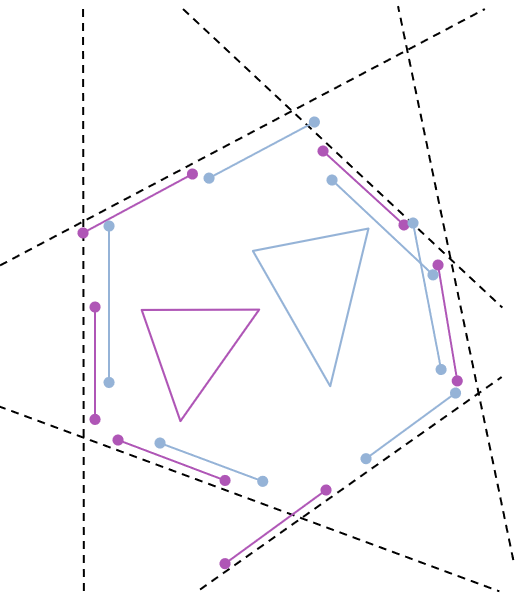
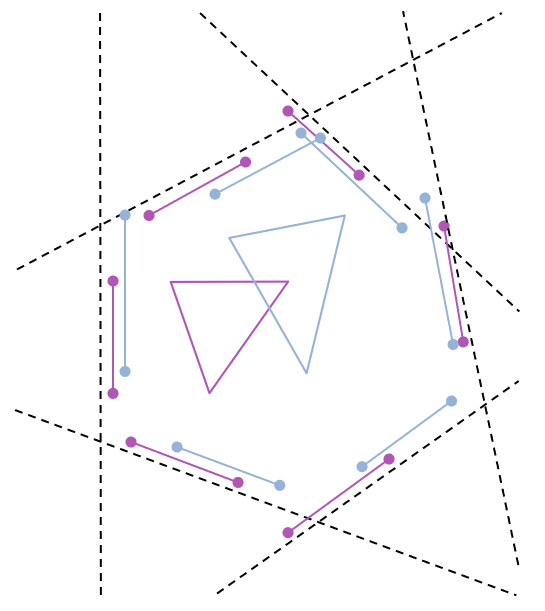
**RA**

**REN**

**RE**

**Triangle Projection**

Next, the projections are tested to see if they overlap. If there is any axis on which the projections do not overlap, then the polygons do not intersect. If the projections overlap on every axis, then the polygons do intersect. A reference picture is shown in Figure E.



**a) SAT in which triangles overlap**

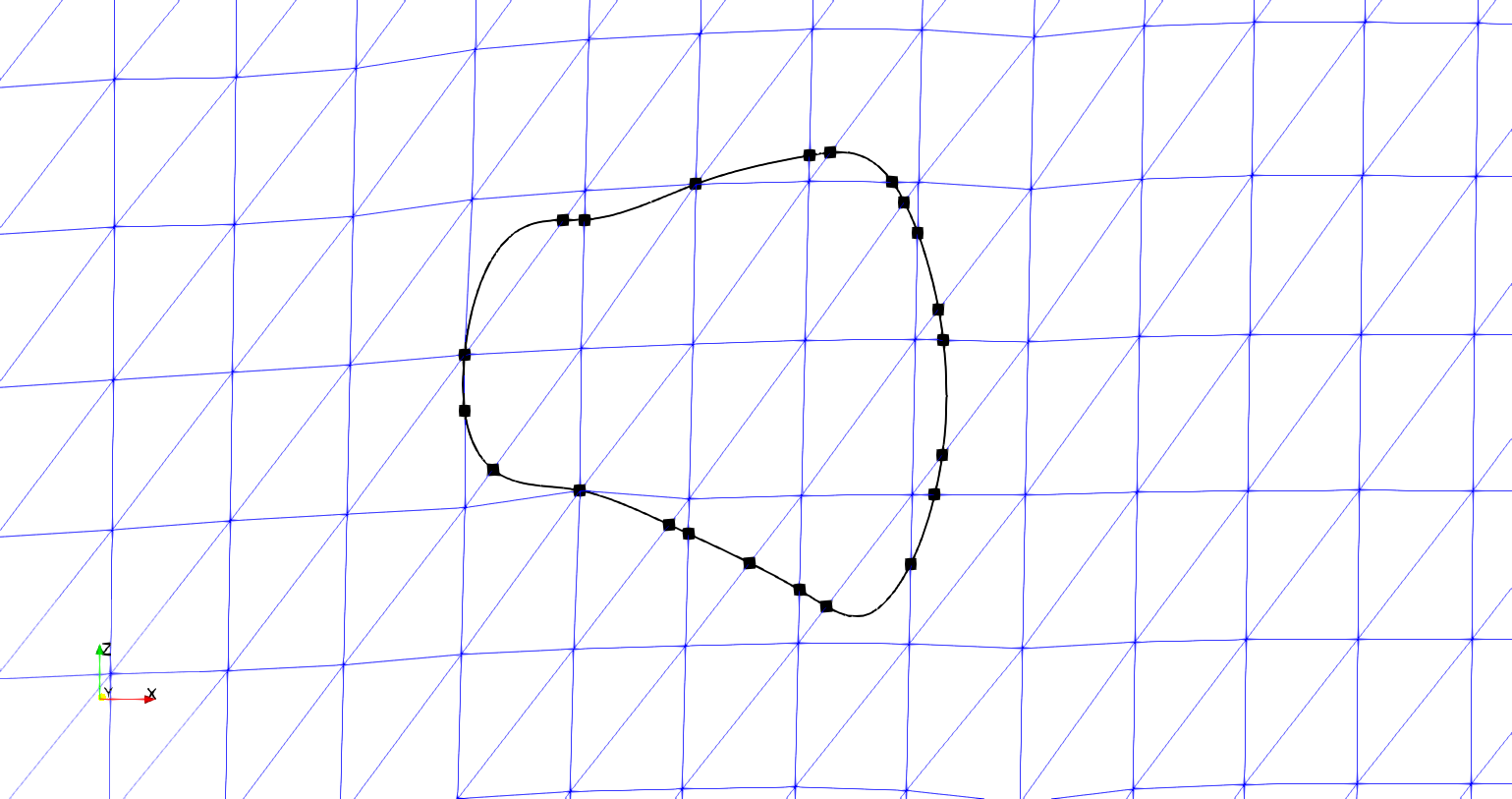
**b) SAT in which triangles do not overlap**

**Figure E: Two cases for testing the Separating Axis Theorem**

In Figure E.a, two triangles are shown to intersect. This can be verified by looking at each dotted line that represents a projection axis. Along each axis the bounds of the triangles are shown. There is no axis in which the bounds do not overlap. Figure E.b shows the case when the two shapes do not intersect. The axes (dotted lines) are the same in E.a and E.b, since the orientations of the two triangles are the same, only the positioning is different. Circled are shape bounds that do not overlap in E.b and therefore verify that the triangles do not intersect. This method is adapted so that the second shape is simply a line segment from the boundary curve. The result is fewer calculations required to establish intersections.

Once the intersected elements are identified, the points where the boundary curve segments intersect the edges of the surface element can be calculated. I will complete this later. Keith has added a different method (from BetaMesh) than what I had originally wrote to do this task and I need to understand it better. I didn’t want to hold up the rest of this report by spending time understanding it.

The result is clearly defined intersection points where the boundary curve intersects surface element edges, shown in figure F. During this process the elements that have been intersected by the boundary curves are recorded for later use in the sub-meshing routines.

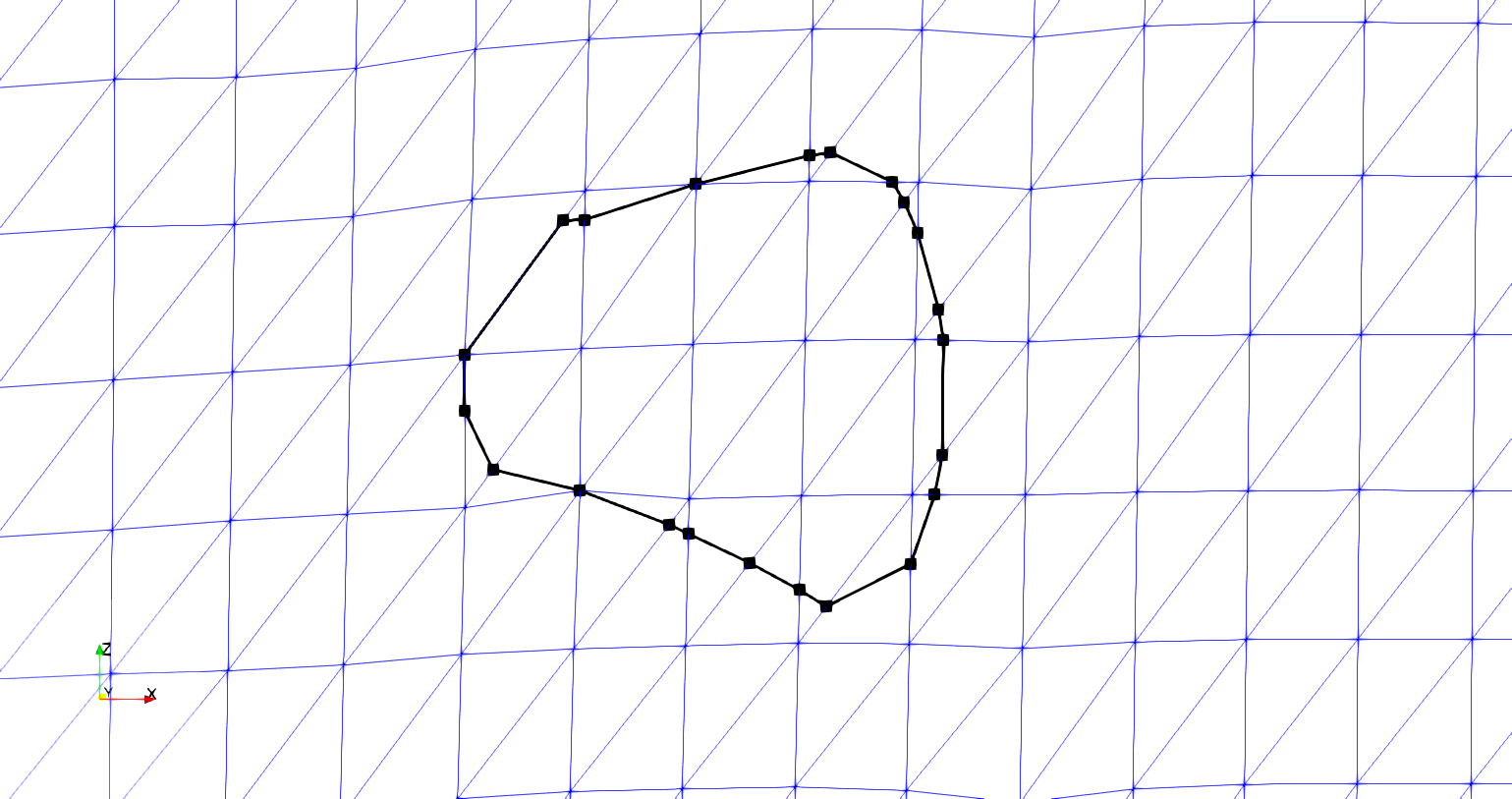


**Figure F: Boundary curve with marked surface element intersection points**

## Using interpenetration points to coarsen and embed boundary curve

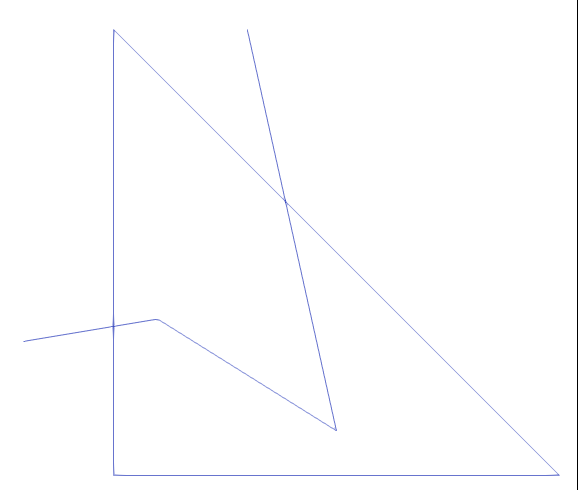
The intersection points are used in coarsening the boundary curve. If the default sampling from the SISL library is used, the resulting mesh refinement around the curve would be much higher than the existing surface refinement. The intersection points identify the path the boundary curve takes through each individual element. By connecting the intersection points with line elements, the curve refinement is effectively reduced. Also, because the intersection points are taken before the boundary curve refinement is reduced, it is ensured that any surface element intersections and surface nodes will still be identified and contained within the curve defined only by the intersection points. This is accomplished by sub-dividing any boundary curve segment at the point of intersection. An iterative loop is run to remove any boundary curve points between two intersection points before the intersection points are connected. This occurs until all non-intersection points are removed. The result of the method can be seen in figure G.

It can be seen from figure G that all of the surface nodes contained in the original boundary curve from figure F are still contained in figure G. Also, the relative shape of the curve is the same, indicating that the lower refinement does not overly affect the interpenetration boundary curve. Figure G is the result of capturing the interpenetration points of just one surface for illustrative purposes. However, the algorithm calculates the intersection points for both tow surfaces whose interpenetrations are bounded by the curve. The result is a curve that becomes the line on which the surfaces initially share compatibility. It is known that along this curve both surfaces have elements with edges that run directly through a point on the curve. Therefore, elements on each surface intersected by the boundary curve will have a set of points on the curve that their edges will line up with, which is required in traditional finite elements. This is the main purpose of calculating the intersection points. Once this compatibility is created along this boundary curve, the curve can be embedded into both meshes using a sub-meshing routine.



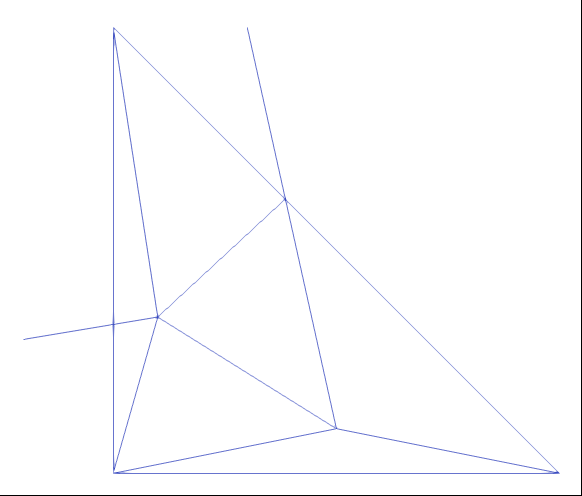
**Figure G: Reduced refinement boundary curve with marked intersection points**

The mesh routine used is from the Triangle software library. The function that calls the routine can take in multiple argument types and produce a triangular mesh of the original shapes. This research uses a form of the arguments which consists of boundary segments that describe all edges required to be in the new mesh, all new mesh nodes, and triangle shape parameters such as minimum angle and maximum sub-element area. The new surface mesh with the boundary curve is created by individually adding each intersected element’s divided mesh back into the original surface mesh. Therefore, an iterative loop is established that iterates over each intersected element previously recorded. The elements are stored by which curve intersected that element. Only elements intersected by the current curve being embedded into the surface mesh are iterated over. This reduces the number of times the intersection detection algorithm is called. The intersection algorithm is computationally expensive but is required to identify the correct boundary curve segments for the current element being re-meshed. The same intersection algorithm is used between the boundary curve segment and the surface element and the boundary curve segments are collected. Once the segments have been collected, they are checked against the element edges to verify which boundary curve segments have endpoints on the element’s edge. Figure G shows only one segment per element but once intersection points are added from both surfaces, there could be more than one segment per element. Figure H illustrates this point with an example element with multiple boundary segment.



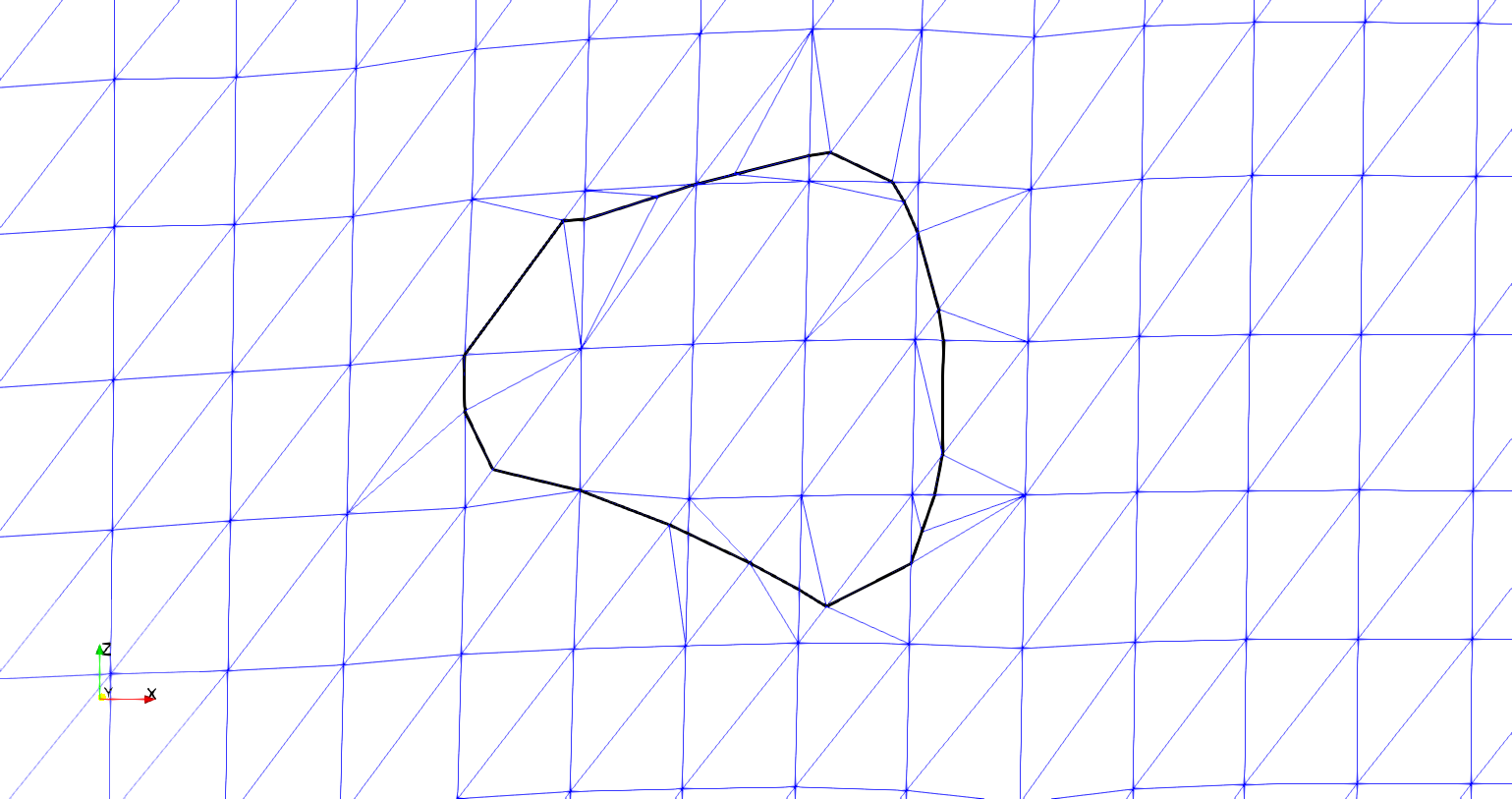
**Figure H: Example surface element with three boundary curve segments intersecting**

Figure H shows two boundary curve segments with endpoints lying on the surface element edges and one curve segment completely contained in the element. The segments that have endpoints on the edge are used to divide the element edges at the intersection point. The resulting two edge segments that make up the element edge are added to a list of segments that will be used to define the boundaries passed to the sub-mesh routine. Once the divided edge segments are added to the list, the boundary curve segments are then also added to this list, along with any remaining edges of the element. The result is a collection of line segments that can be used with the Triangle sub-meshing routine to create a resulting sub-mesh of the single element, as shown in figure I.



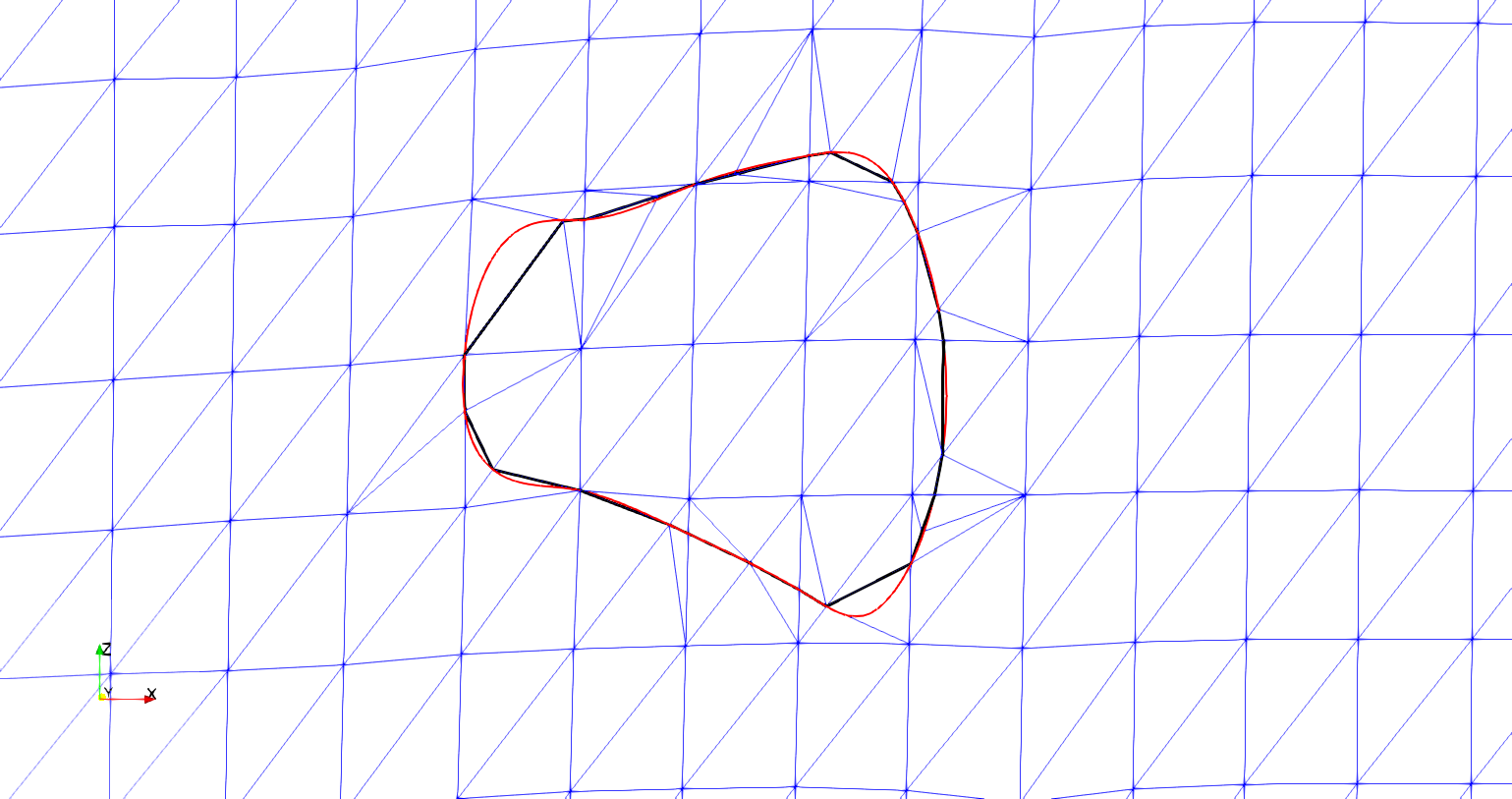
**Figure I: Sub-mesh of surface element with boundary curve**

The sub-mesh element can then be added back into the surface mesh, replacing the original element. Figure J shows the result of the algorithm for the test case shown.



**Figure J: Reduced refinement boundary curve and re-meshed surface elements**

Figure J shows both the relative refinement of the new sub-meshed elements as well as the reduced refinement boundary curve. The refinement is comparable between the sub-meshed surface elements and the untouched elements. Figure K shows the original boundary curve compared to the new curve used to re-mesh the surface.



**Figure K: Original (red) vs reduced refinement boundary curve with new surface mesh**

Figure K shows how the overall shape of the boundary curve is still maintained and that no features such as surface nodes or elements are left out from the reduced refinement boundary curve. Some sections of the curve are not captured fully when the curve has been coarsened but this does not affect the ability of the algorithm to capture the correct interpenetrating elements or nodes between the two surfaces. The result is a fully embedded boundary curve into the surface mesh. When the curve has been embedded into both surfaces, they will then share a curve to being enforcing compatibility between the surfaces. This is the most important feature of the methods developed during this research. Previously, there has not been a method that will ensure a compatible region between any two tow surfaces. Now, the two surfaces share a common boundary where they cross into each other. There is still no compatible surface at this point.