# Use of SISL Non-Uniform Rational B-Splines Library to Identify Interpenetrations

The method used by VTMS to remove surface interpenetrations removes the majority of interpenetrations but lack the accuracy to detect all interpenetrations and the methods to remove the interpenetrations were not robust. Several other techniques were also evaluated and developed to remove the surface interpenetrations within VTMS but were also not sufficiently robust. These methods are described in the appendix (Appendix ref). Therefore, another approach was needed to reliably detect and fix surface interpenetrations. There are many references that use parametric surface representations to detect interpenetration between surfaces [REFs]. One of the more popular parametric surface types is the non-uniform rational b-spline surface (NURBS). NURBS were chosen because of the documentation available, third-party support, and ease of implementation. The SISL library from the Department of Applied Mathematics at SINTEF ICT is a NURBS library designed for the “modeling and interrogation of curves and surfaces.” [REF] The library is used for the fitting of a NURBS surface to the VTMS surface geometry and the detection of the intersections between two NURBS surfaces. The faceted VTMS surfaces are passed to the library in the format required by the library, which is discussed in further detail in a later section. The library then fits a NURBS surface to each VTMS tow surface and calculates all the intersections between the two surfaces. The library returns these intersections as b-spline curves. The library does not correct the surfaces to eliminate the interpenetrating regions between the tow surfaces. Also, the b-spline curves returned from the library are not always closed and , a requirement that is later discussed in detail.

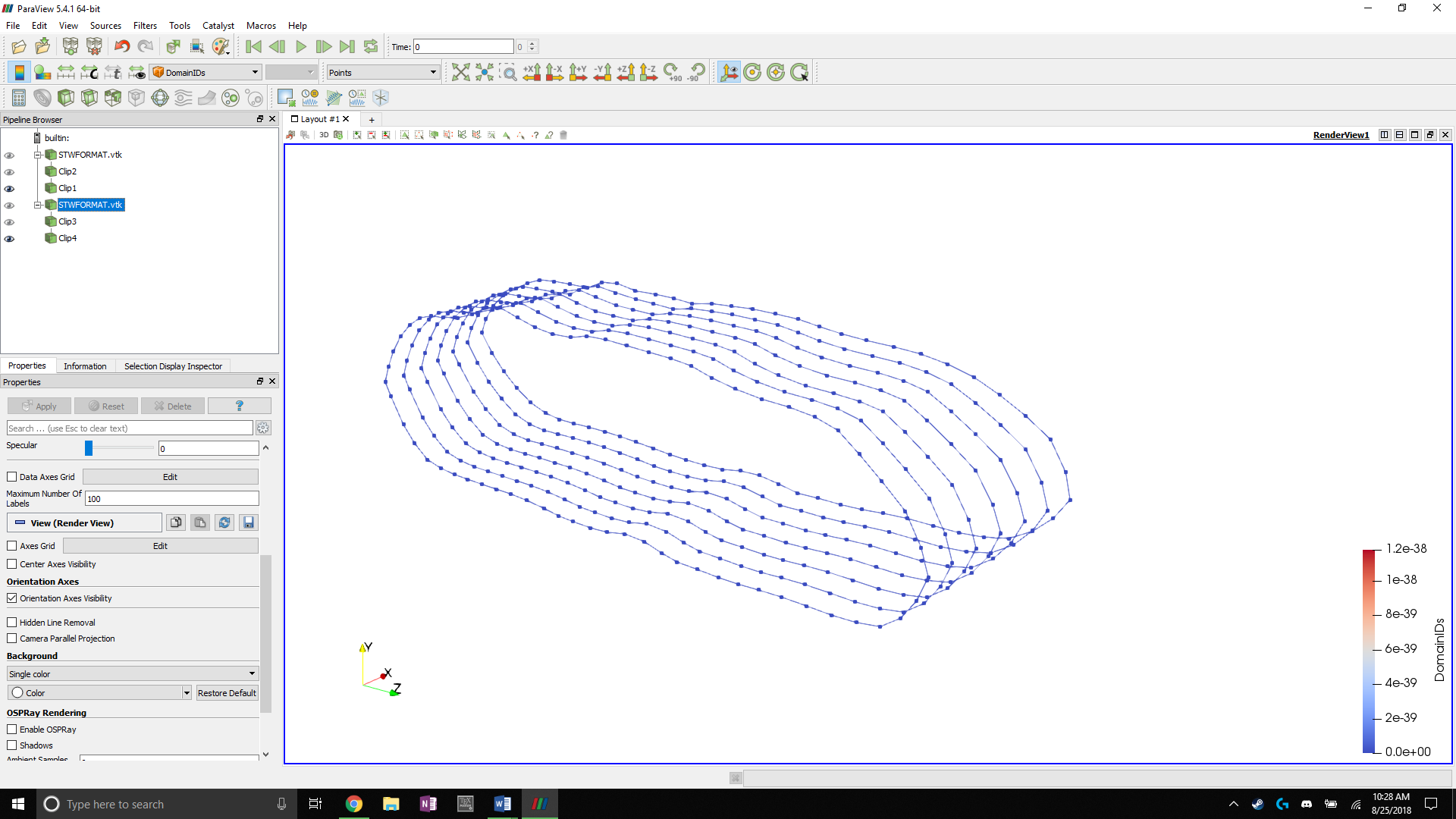
## Overview of NURBS method

The main drawback of the polygon surface interpenetration detection methods is that they operate on a point-by-point basis. They depend on only points being interpenetrating and do not conduct any other checks to verify if other parts of the surface elements are interpenetrating, such as their edges. By transforming the surfaces into b-spline surfaces, an intersection curve is created that accurately outlines the interpenetration. This requires creating an accurate b-spline surface from the pre-existing surface mesh and calculating the intersection curve between the two surfaces. This curve acts as the boundary of the interpenetration region between the two surfaces. The surface mesh is cut by the intersection curve so that the elements inside of the intersection can be removed and replaced with a mesh that is shared between the two surfaces. The details of these methods are further explained in the following sections.

## Pre-SISL data formats and SISL required inputs

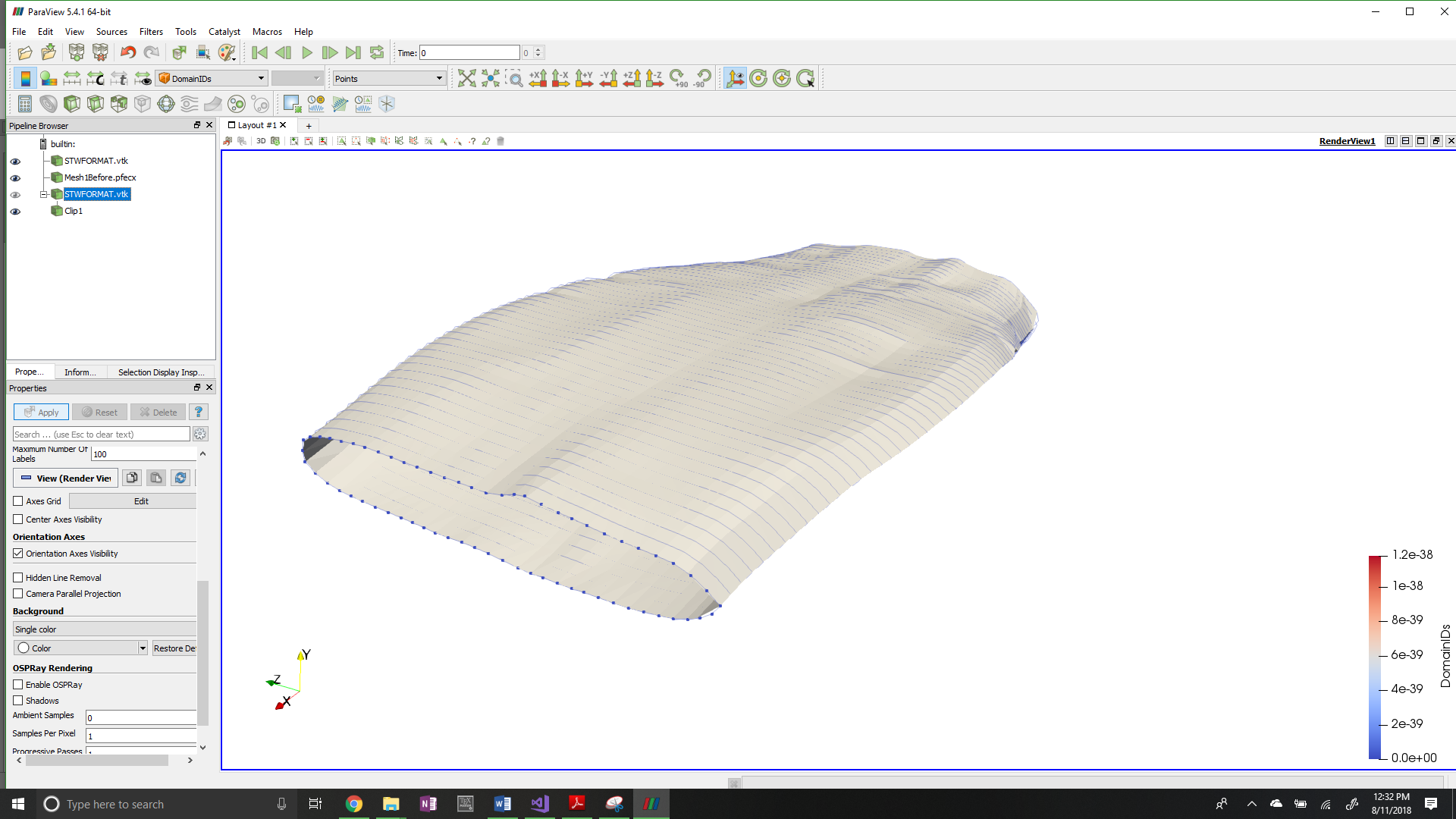
A NURBS surface is a sub-class of a parametric surface and uses a curvilinear coordinate system to describe a three-dimensional surface. NURBS use two parameter values to return the location of the surface at any point. Therefore, a way to describe the VTMS tow surface with two parameters is needed. The SISL library requires a list of all the points in the surface, and the number of points in the two curvilinear directions. These parameter requirements needed to be fulfilled to create a NURBS surface using the library.

VTMS has a format describing its surfaces that can be easily used with the SISL library. VTMS’ standard tow format (exported as the .stw file type) describes the surface as a series of polygonal cross sections where each cross section is made up of the same number of points. These cross sections are perpendicular to the path of the tow and all points of a cross section lie in the same plane. Each point that defines the surface has a cross section it belongs to and can be found using two reference values, the cross-section number it lies on and which number point it is in the cross section. The reference values are analogous to curvilinear coordinates that describe the surface. Figure A shows multiple cross sections with its points and the starting point of each stack is marked. The lines that outline the cross sections have been added to clarify which points belong to a cross section.



**Figure A: Tow surface cross sections with starting points of each stack marked**

A method was developed to read the .stw file in the order that the data was exported, which lists each cross section in order and the points that make up that cross section. Each point is stored in the order that it is read from the file. SISL is then given the number of cross sections and the number of points which are used to define the range of the curvilinear coordinate in the ***u*** and ***v*** directions (Figure B). The cross-section’s points become the control points for the NURBS surface.

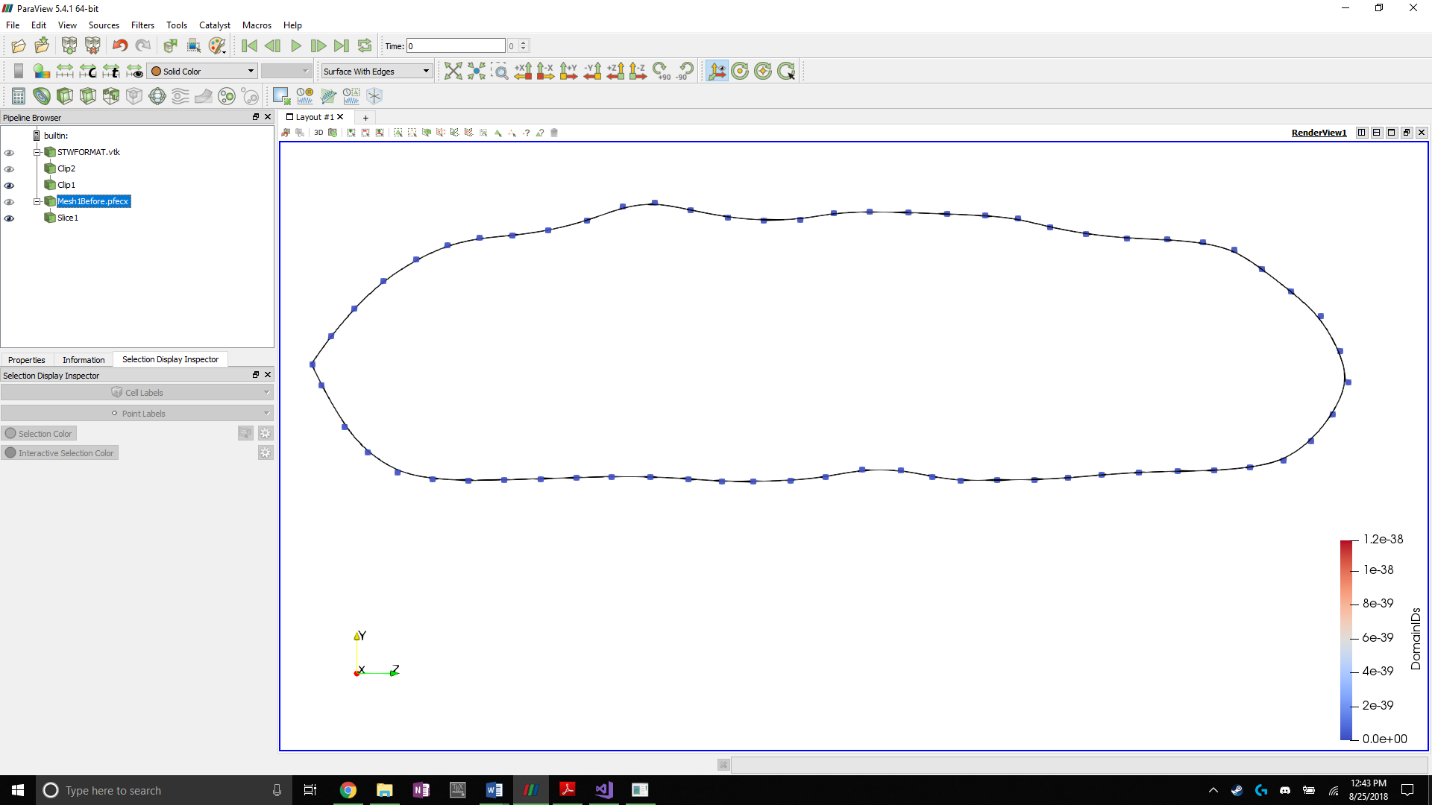
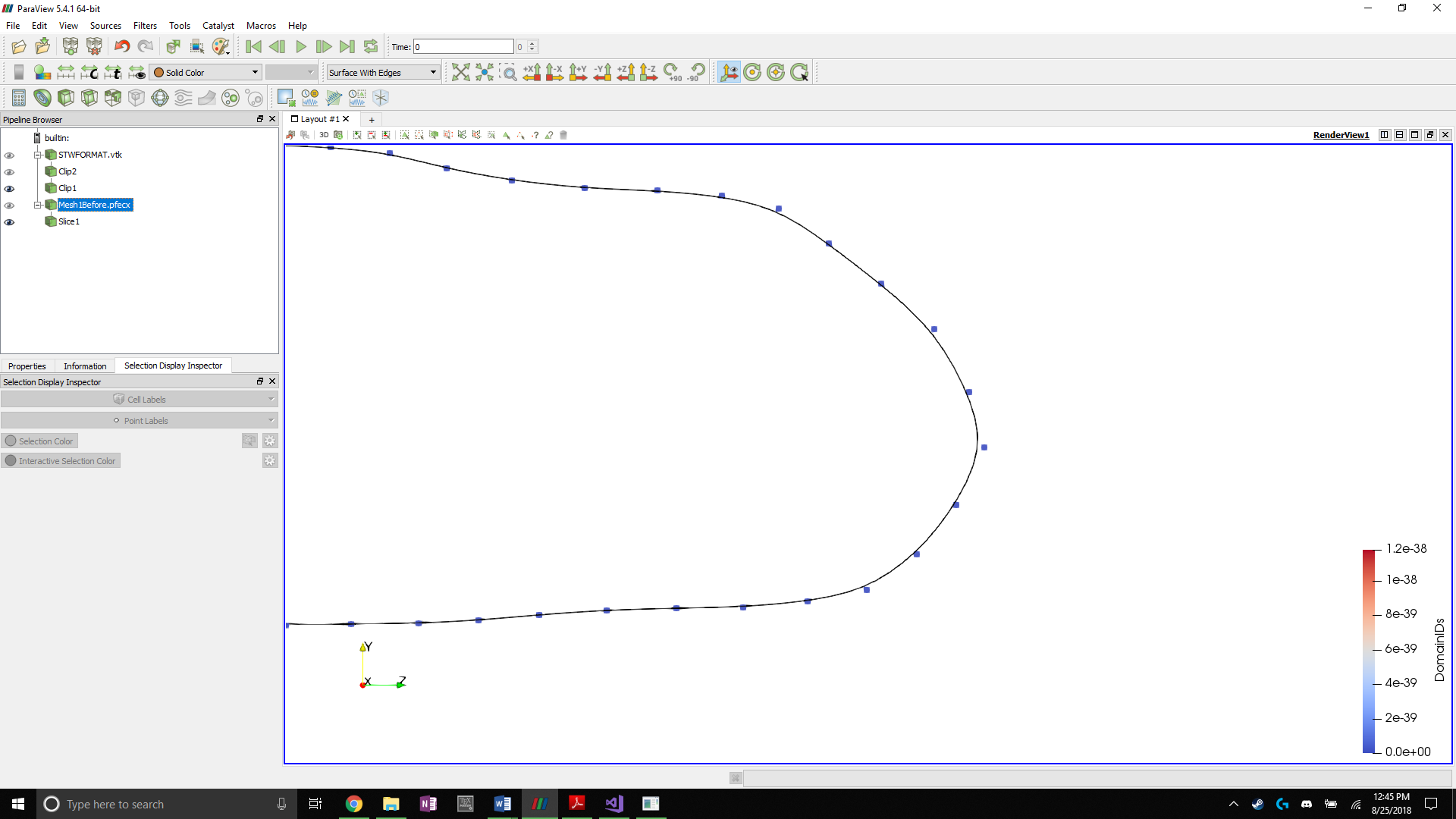


***u***

***v***

**Figure B: Parametric coordinate system for tow input data to the SISL library**

SISL has its own method that takes this data as an input and converts it to a SISL definition of a NURBS surface. It is important to note that the resulting surface approximates the data sent from VTMS. This is because of how NURBS are formed by their control points. The curves that make up the surface are not required to run through the control points. In general, the control points rarely lie exactly upon the b-spline curve it defines. However, for a non-idealized tow surface produced by VTMS, the NURBS approximation is very good because the surfaces produced by the SISL library fit the original data very well. This relationship is clearly shown when the surface created by SISL is compared to the original VTMS surface data. Figure C shows the original VTMS points of a single cross section compared to the resulting NURBS curve. The figure shows that in the region where the surface arcs significantly, the points that are used to control the surface do not lie on the surface but are close. Where the surface does not bend significantly, the points lie close to or on the surface. By observation, the SISL library captures nearly all of the tow volume as well as keeps many of the topological features (peaks and valleys of the surface) that are important for detecting surface interpenetrations. Therefore, the NURBS surface approximations are an accurate approximation of the VTMS surfaces and will result in accurate detection of the interpenetration regions.



**Figure C: NURBS approximation of tow cross section with original VTMS data as control points**

Once the NURBS surfaces have been created by the SISL library, the surfaces are then used by the library to detect the interpenetration regions and return the intersection curves that outline the regions where two surfaces interpenetrate. The terms intersection curve and boundary curve can be used interchangeably. The curve that bounds the interpenetration regions is the same curve that traces where the two surfaces intersect and cross into each other. The library returns these interpenetration boundary curves as b-splines. To view the results, multiple methods were developed to export the SISL data into file types to be used in a data analyzing and visualization software.

## NURBS and B-Spline viewing algorithms

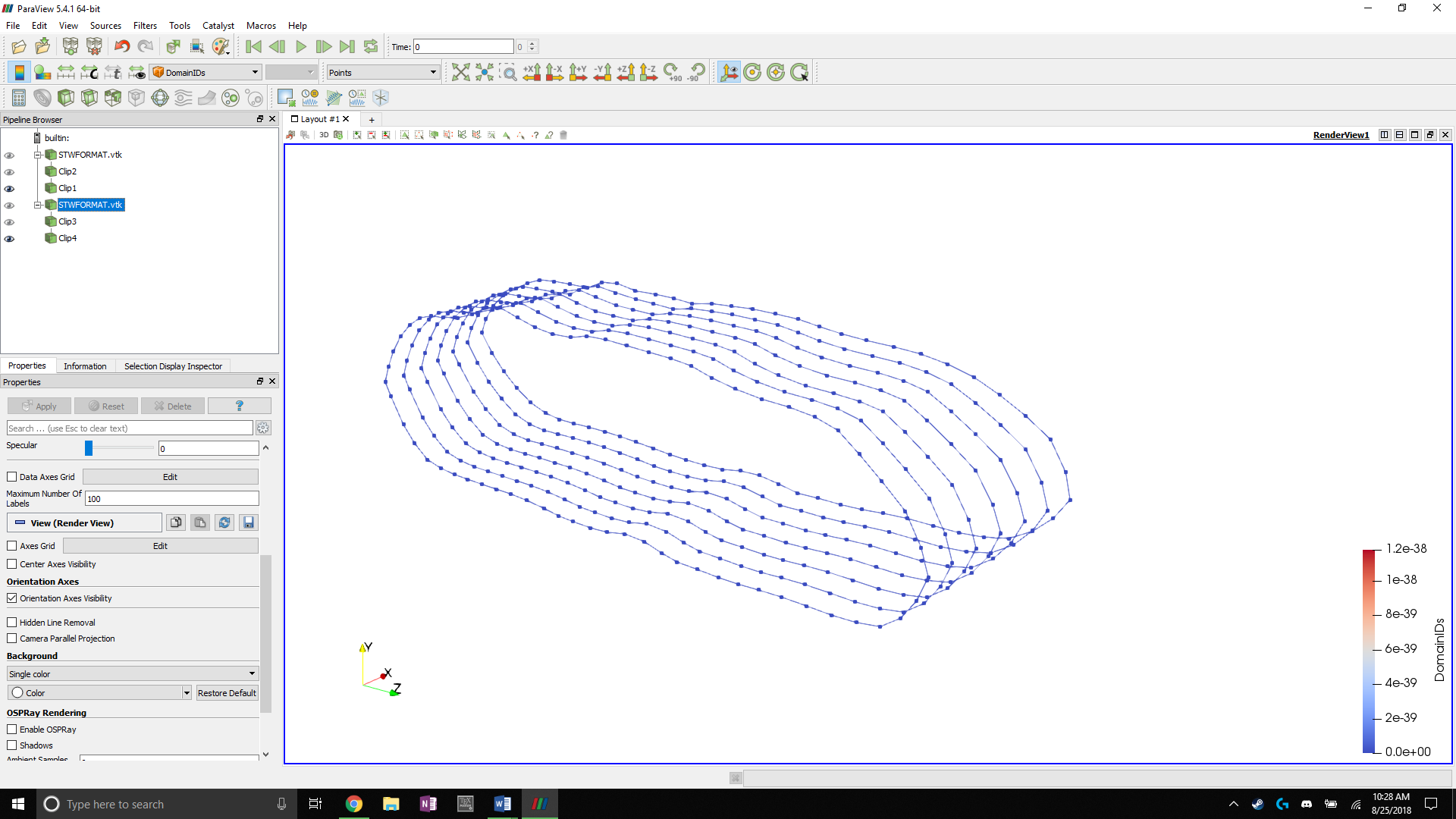
One drawback of using NURBS and B-Splines to describe a surface is visualizing the surfaces and curves. Most visualization software suites use polygons to approximate the surface of complex shapes to render them for viewing. Therefore, methods to adapt the b-splines to visualization data types had to be developed to use the chosen viewing software, Paraview. Paraview was chosen as the main visualization software due to previous experience using the software. SISL comes with its own visualization suite, but it does not have data analyzation methods and lacks important functionality available in Paraview that is used to show interpenetrations between surfaces. Another set of software, called BetaMesh, was instrumental in many of the methods developed. BetaMesh is a mesh framework that is well suited for domain discretization and manipulation. BetaMesh also has modules that can conduct a wide variety of finite element analyses. However, BetaMesh is primarily used for its mesh framework. BetaMesh was developed at Texas A&M University and is currently maintained by Dr. John Whitcomb and his students. BetaMesh also comes with methods that allow it to directly export mesh data to Paraview for visualization. Therefore, it is well suited for the task of visualizing NURBS as a polygonal surface mesh.

To use BetaMesh as a mesh framework, the NURBS surfaces and b-splines from the SISL library must be discretized. This was accomplished by using a surface (or curve) point sampling function supplied by the SISL library. The function returns the cartesian coordinates of the surface at a chosen set of curvilinear coordinates. The maximum and minimum curvilinear coordinates for both parametric directions of the NURBS surface were found to make sure the surface is sampled at its boundaries. These ranges are were used to evenly divide the parameter space to reflect the desired amount of surface refinement.

The refinement of the surface is controlled by increasing the number of times the surface is sampled in the respective parametric directions. The same level of refinement as the original VTMS surface data was chosen because the refinement is a good compromise between accuracy and coarseness. The surface was sampled corresponding to the number of cross-sections in the ***u*** direction (axial direction of the tow) and the number of points per cross-section in the ***v*** direction (circumferential direction of the tow), similar to figure A. The surface was sampled by starting at one end of the tow surface (minimum coordinate value in ***u*** direction) and sampling around the circumference of the surface (stepping from the minimum to the maximum coordinate value in ***v*** direction). The method then steps to the next coordinate value in the ***u*** direction and samples the entire circumference again. Each time the surface was sampled, the function returns a set of coordinates that was used to create a BetaMesh node object. These node objects store node data such as coordinates, local and global number, owning partitions, and multiple functions that can edit and assign the node the various attributes. These nodes were stored in a list in the order they were sampled so that the order in which they are stored is known. This is important when forming the surface elements that are used for the visualization. Once the surface is sampled, according to the refinement chosen, a method creates the surface mesh using the vector of nodes.

The mesh was easily created because the order that the surface was sampled and how the nodes in the vector were ordered was known. The algorithm begins by connecting two adjacent nodes on a cross-section and doing the same for the proceeding cross section. Then, the same indexed points on the two cross sections are connected, creating a quadrilateral element.

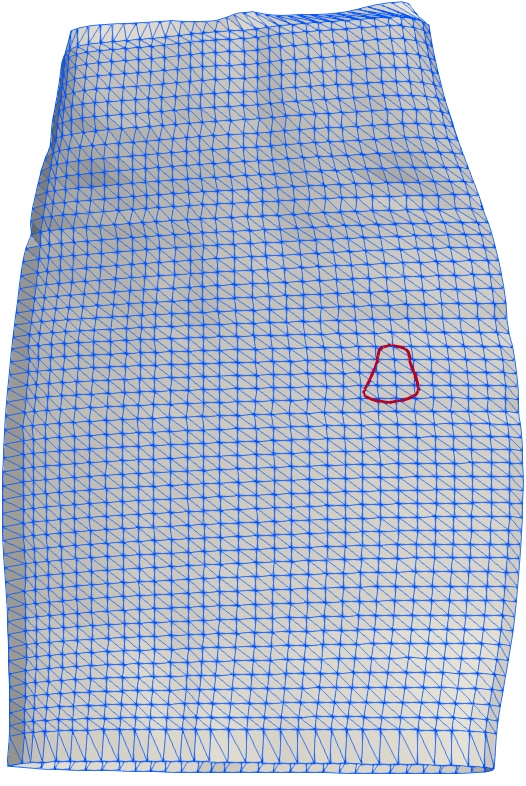
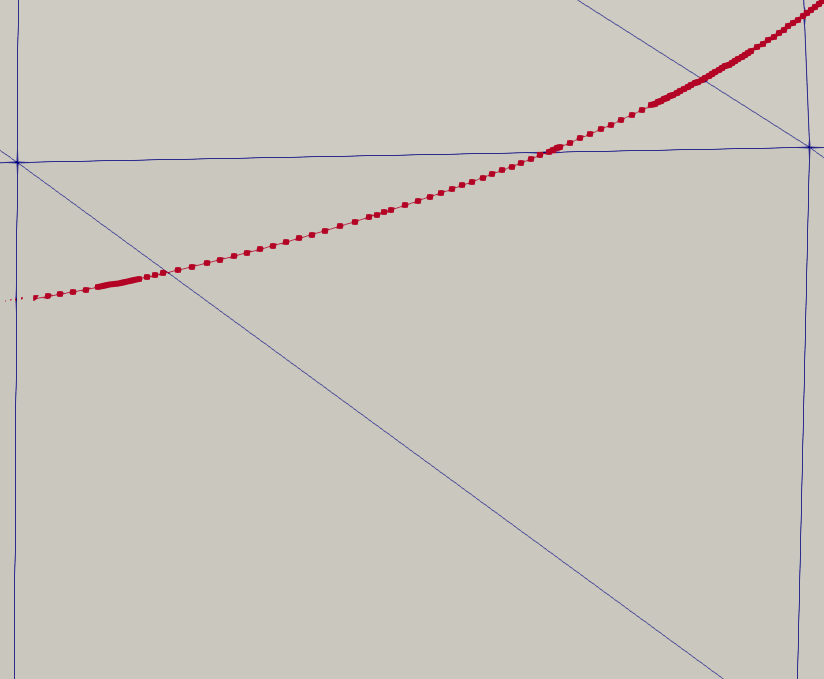
Two opposite corners of the quadrilateral are connected to create two triangles, as shown in figure D. These triangles are stored as BetaMesh triangular elements. All of these elements are assigned to a surface mesh that is exported by a BetaMesh function to file formats that are used in Paraview.



**Figure B: Cross-section polygons with connecting surface elements**

The SISL library also returns the intersection curves as b-splines, which were converted so that they can be viewed by Paraview. The same function that samples the NURBS surface at regular intervals is used and coordinates of the curve are returned. For each node that is created from the returned coordinates, a line element is created connecting it to the previous node. A line element is defined as a three-dimensional, two node line segment that can be manipulated using any element functions in the BetaMesh library. The curve was sampled at a high refinement so that the meshed curve closely resembles the b-spline. Figure E shows an interpenetration boundary curve and an example of the refinement level of the boundary curve.

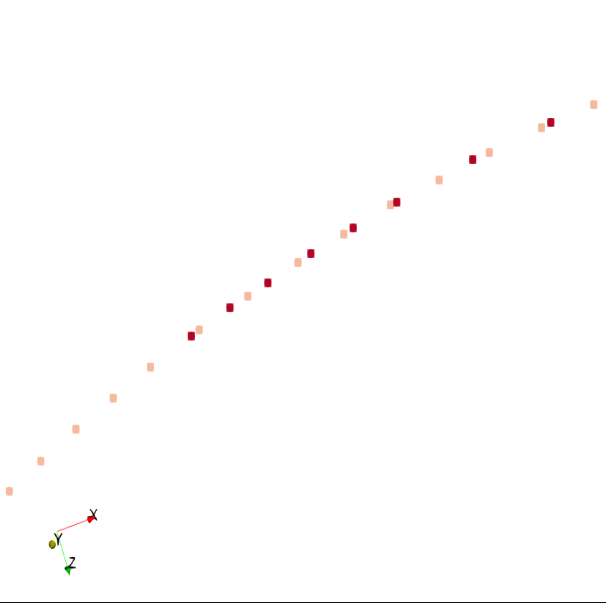
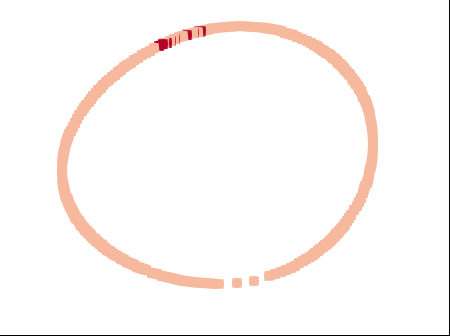
The result of these methods is the ability to visualize NURBS surfaces and curves from the SISL library. The surfaces shown in Paraview are approximations of the NURBS surface due to the discretization process that was used to visualize them. However, the points that define the surface elements were directly sampled from the surface to reduce the error in the approximation. If the refinement of the sampling is increased, the relative error between the approximation and the actual surface is reduced. The same relationship applies to the intersection curve. The higher refinement of the interpenetration boundary curve relative to the surface ensures that all interpenetrating nodes from the surfaces lie inside the intersection curve. Once the surfaces and intersection curves are converted, the surfaces are ready to be cut by the intersection curves.



**Figure E: Surface mesh of NURBS surface with boundary curve and relative refinement of curve**

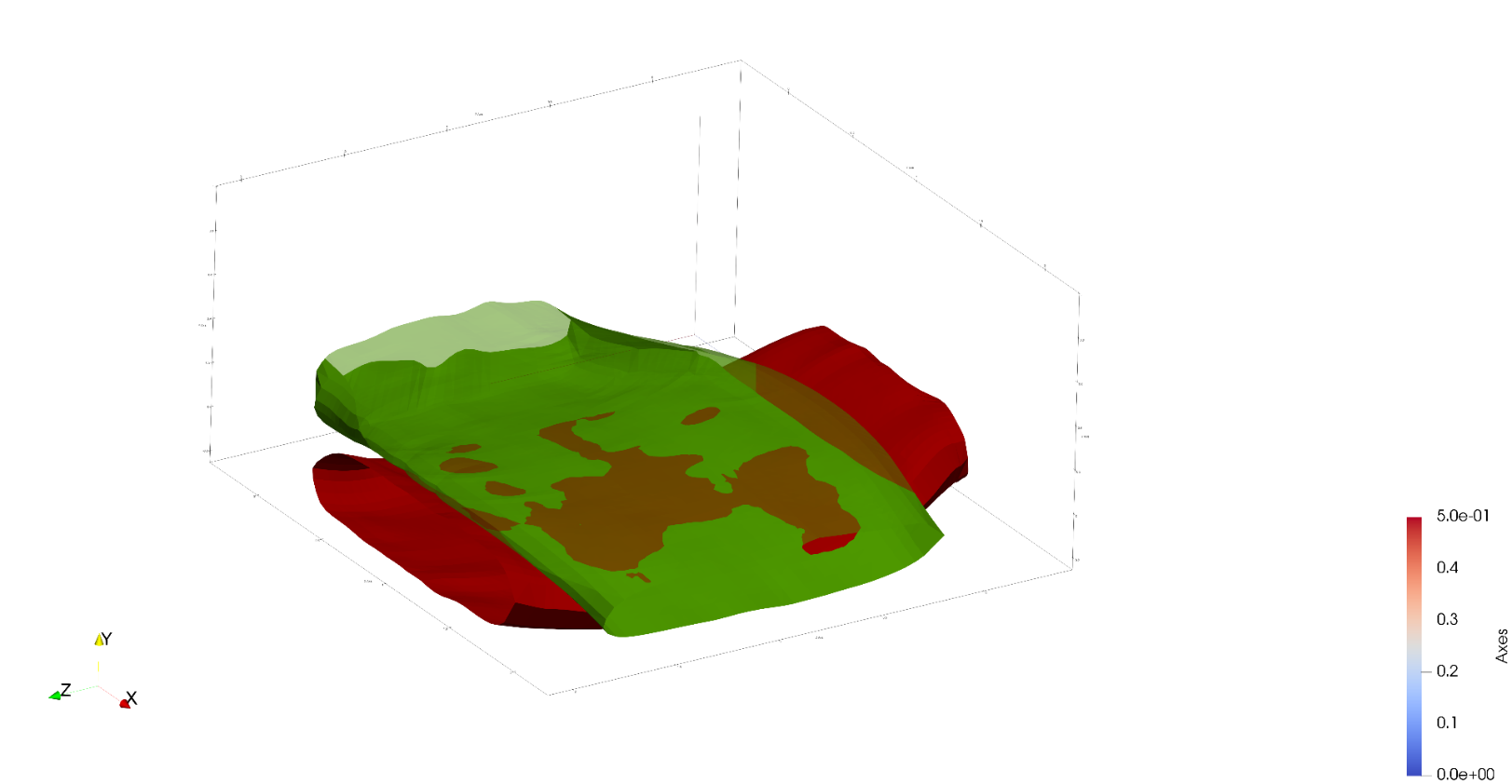
## Correction of intersection curve meshes

The main drawback of using the SISL library is that the intersection curves returned from the methods are not guaranteed to be unique, closed, or have shared nodes between their converted meshes due to how the sampling method works. Therefore, the meshed curves need to be corrected before they are added to the surface mesh. Figure F shows an example of two surface intersection curves that were returned by SISL and converted into a line element mesh. Only the nodes that make up the curves are shown for clarity. They are described as non-unique because the smaller, open curve (dark in color) duplicates a region of the larger, closed curve (light in color).



**Figure F: Non-unique meshed boundary curves with duplicated curve data**

There are multiple scenarios where two curves either overlap and duplicate data, or connect to create a closed boundary curve. An assumption is made that if two surfaces interpenetrate then any interpenetration region that they create can be described by a boundary that is closed. This is a valid assumption because if two surfaces interpenetrate in a relatively small region then one surface should both enter and exit the opposing surface, creating a closed region of interpenetration that is bounded by a closed boundary curve. Figure G can be used as a reference for two surfaces interpenetrating. Using this assumption, it is required that all the boundary curves, or intersection curves, from this method are unique and closed.



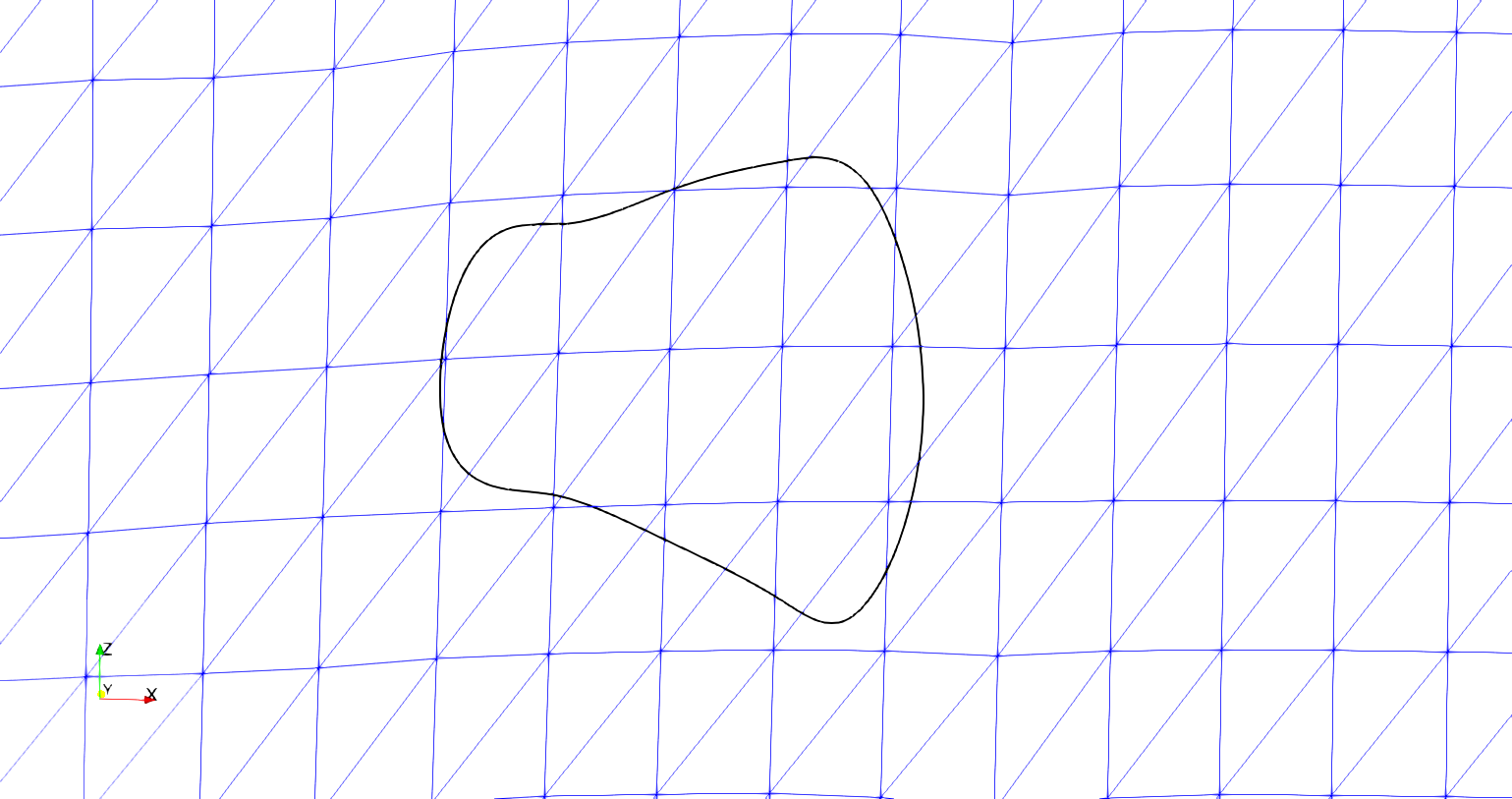
**Figure G: Two tow surfaces with interpenetrating regions**

The first step of this method identifies which surface intersection curves are closed and which are open. When the NURBS curves were converted into line element meshes, line elements were created between every pair of adjacent nodes sampled from all of the curves. The line elements were not linked together, but instead added to a list and ensured that each linear element was unique. These elements were also added to one large BetaMesh mesh object which allowed the usage of mesh manipulation operations, such as the ability to determine how many elements a node belonged to. Most nodes belong to a pair of line elements because they are the connection point between two individual line elements. However, some nodes only belong to a single line element, which indicates that it is the beginning or end of a curve. One of these nodes is identified as the starting point for a search loop that builds a set of boundary curves that are known to be open or closed. Using this node, the element it belongs to is selected and added to a temporary mesh object that is used to record the current curve being defined. The second node on the element is used to find the node’s remaining parent element. The second parent element is added to the curve mesh and the process is repeated, maintaining connectivity definitions. As each new element is added to the curve mesh, it is removed from the list of line elements and checked to see if the second node of the element is at the same location as the original single parent node. If the two nodes coincide, then the curve is defined as closed and the algorithm loop is exited. The closed curve is then added to a boundary curve mesh list for later use. If the algorithm finds a node that has no second parent element and does not coincide with the original starting element, then it is an open curve. The curve is labeled as open and saved as well. This process continues until all of the boundary curve line elements have been removed from the original list of unique line elements. The end result is a curve list containing both open and closed boundary curve meshes. This is required so that duplicate and open curves could be removed.

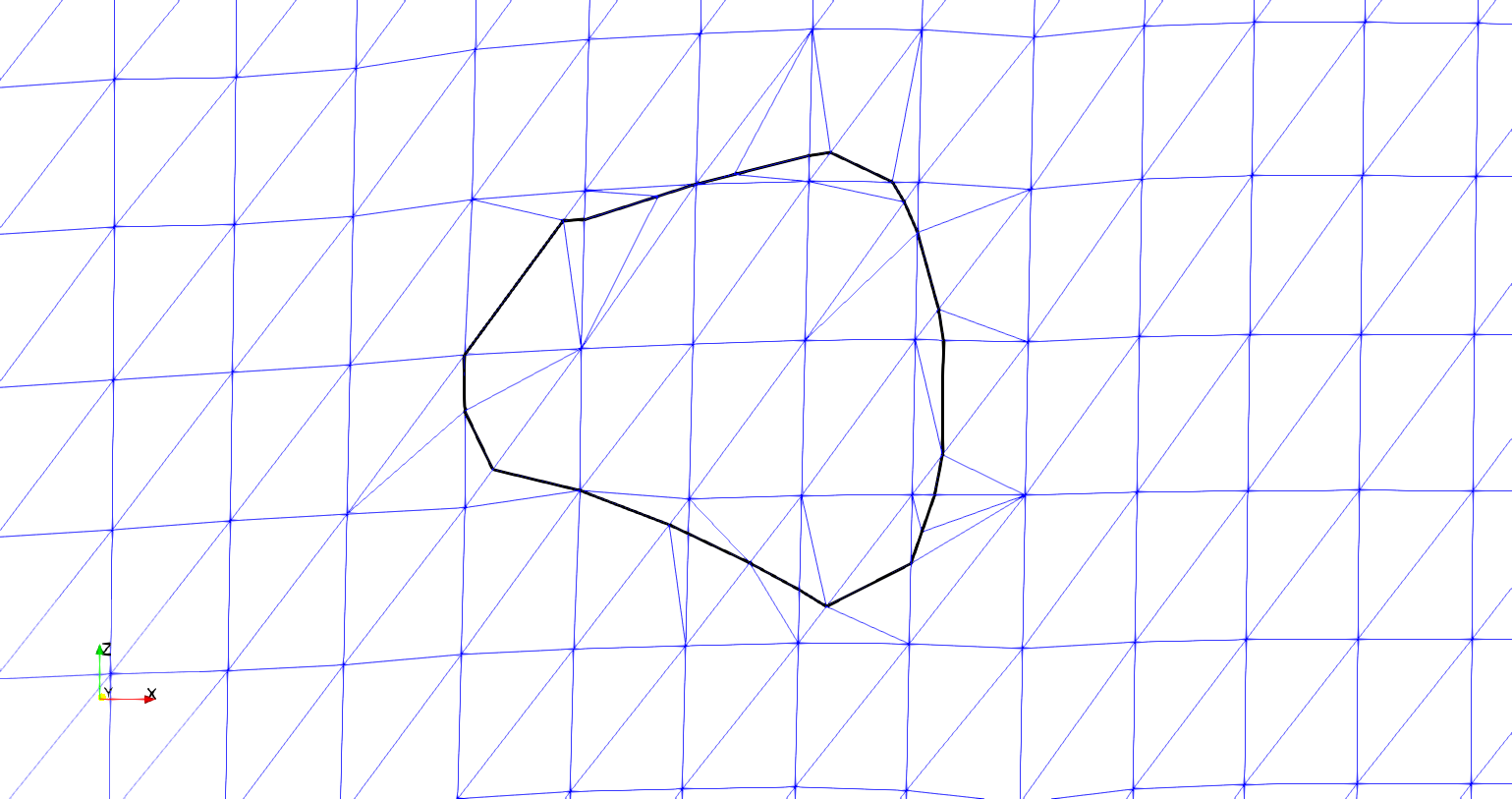
To determine if any of the curves duplicate data, a method compiles all of the curves into a list and performs a dual-loop iteration. These loops choose the first closed curve (denoted curve **A**) in the list and compare the remaining boundary curves against it. If the curve being compared (curve **B**) lies on curve **A**, whether closed or open, it is removed from the list. Overlaps are determined by an algorithm that chooses both the start and end node of curve **B** and whether it lies on any line element from curve **A**. The method that calculates this uses the two end points of the line element (nodes **a** and **b**) and the node to be checked (node **c**). A vector is created to connect **a** and **b**, as well as a vector from **a** to **c**. The cross product between these two vectors is calculated and if it is below a certain tolerance, node **c** is determined to lie on the line that goes through **a** and **b**. The dot product is then calculated between the two vectors. If the result is greater than zero but less than one, it is known that point **c** lies between the points **a** and **b**. These two checks verify that node **c** is on the line segment formed by **a** and **b**. This identifies an overlap and the curve which the node belongs to is removed. If the test does not result in an overlap for the beginning or end node of the curve, the curve is kept in the curve list. The result of this method is a set of unique, closed curve meshes that are used to subdivide the surface meshes along the boundary curves.

## Detecting surface elements intersected by a boundary curve

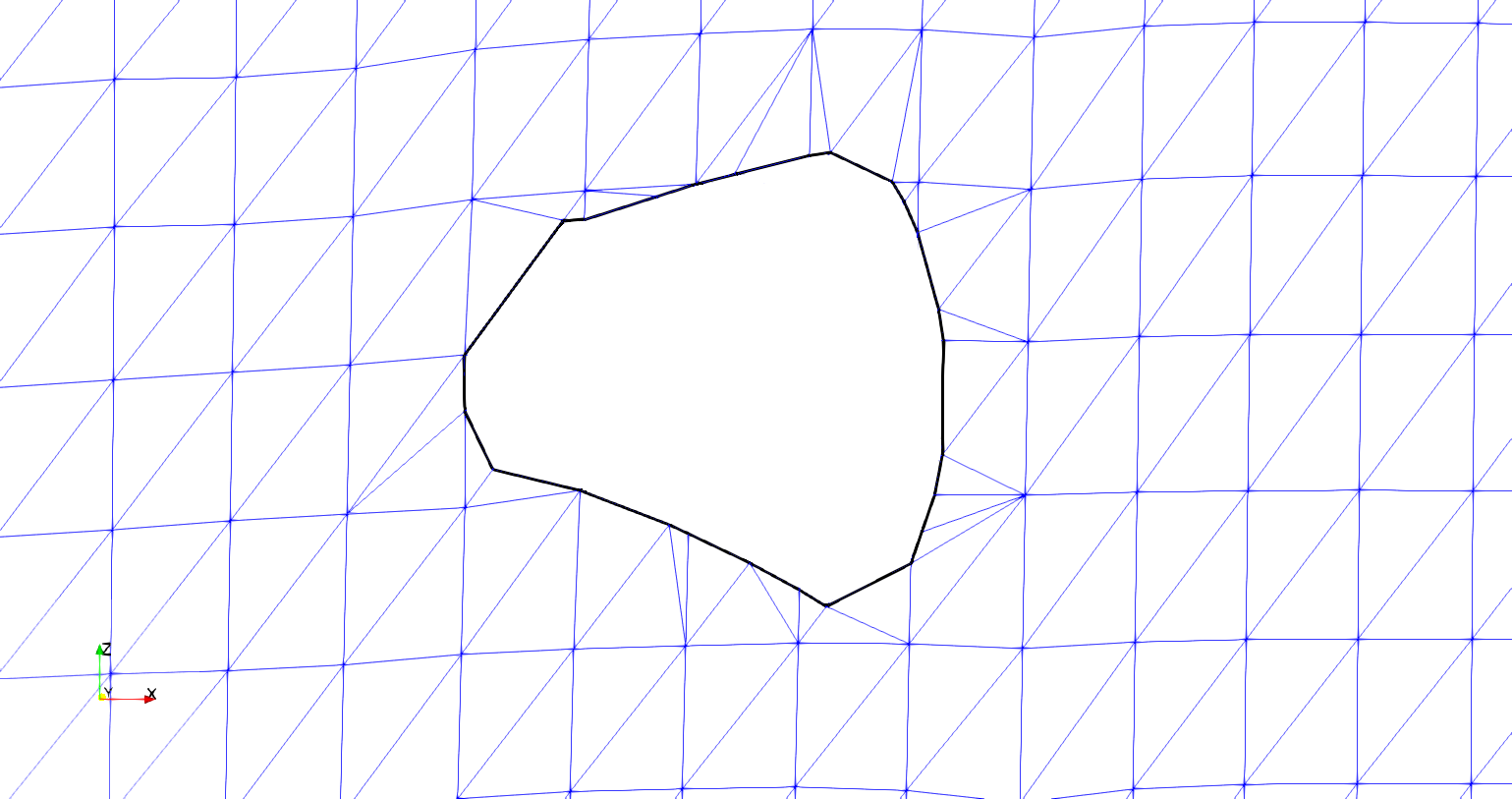
Once the surface intersection curves have been made unique and closed, the curves are used to cut the tow surface elements they intersect. Figure H shows an example of an intersection curve that is used to cut the tow surface mesh and the removal of the interpenetrating elements.



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

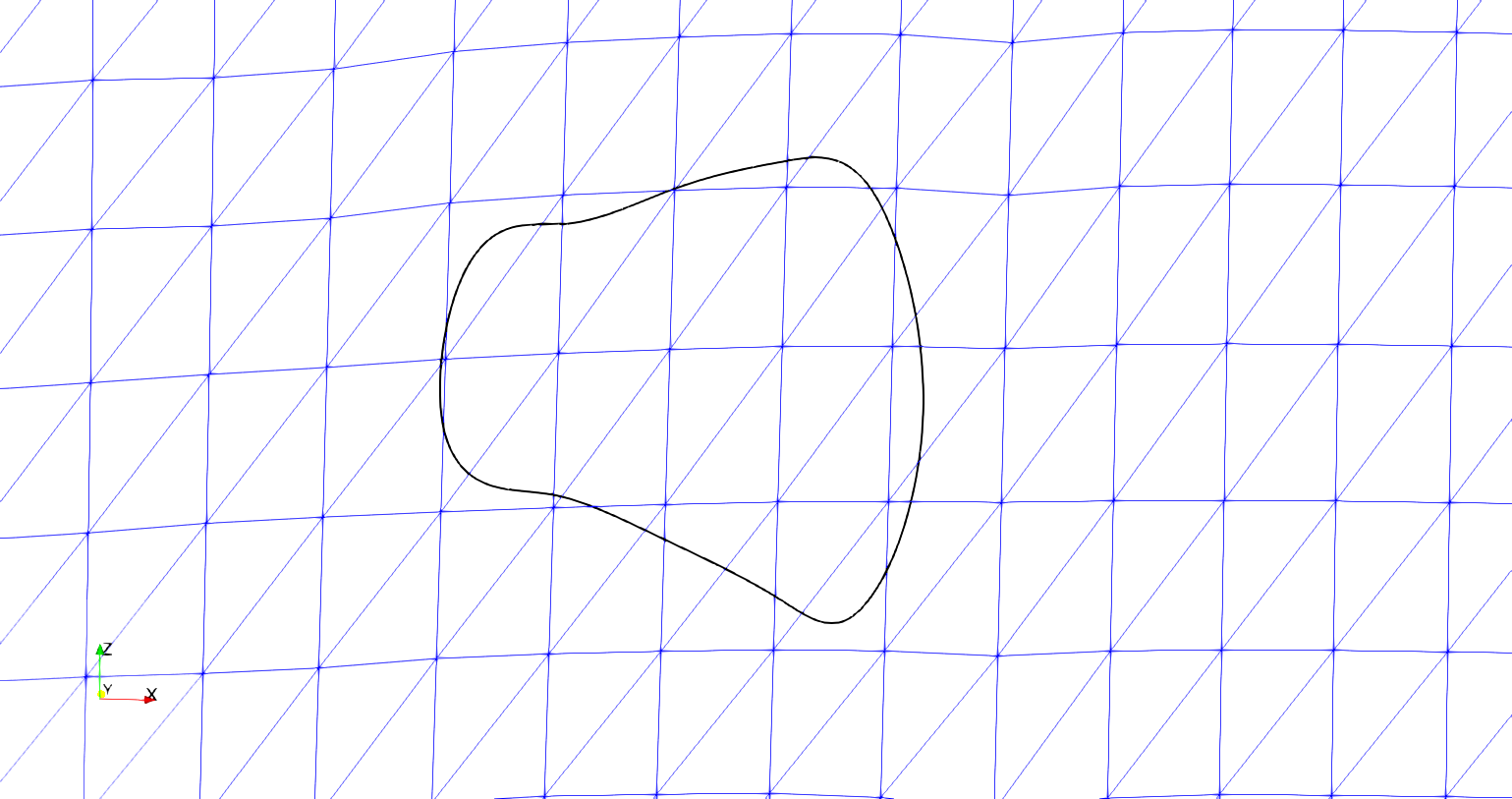
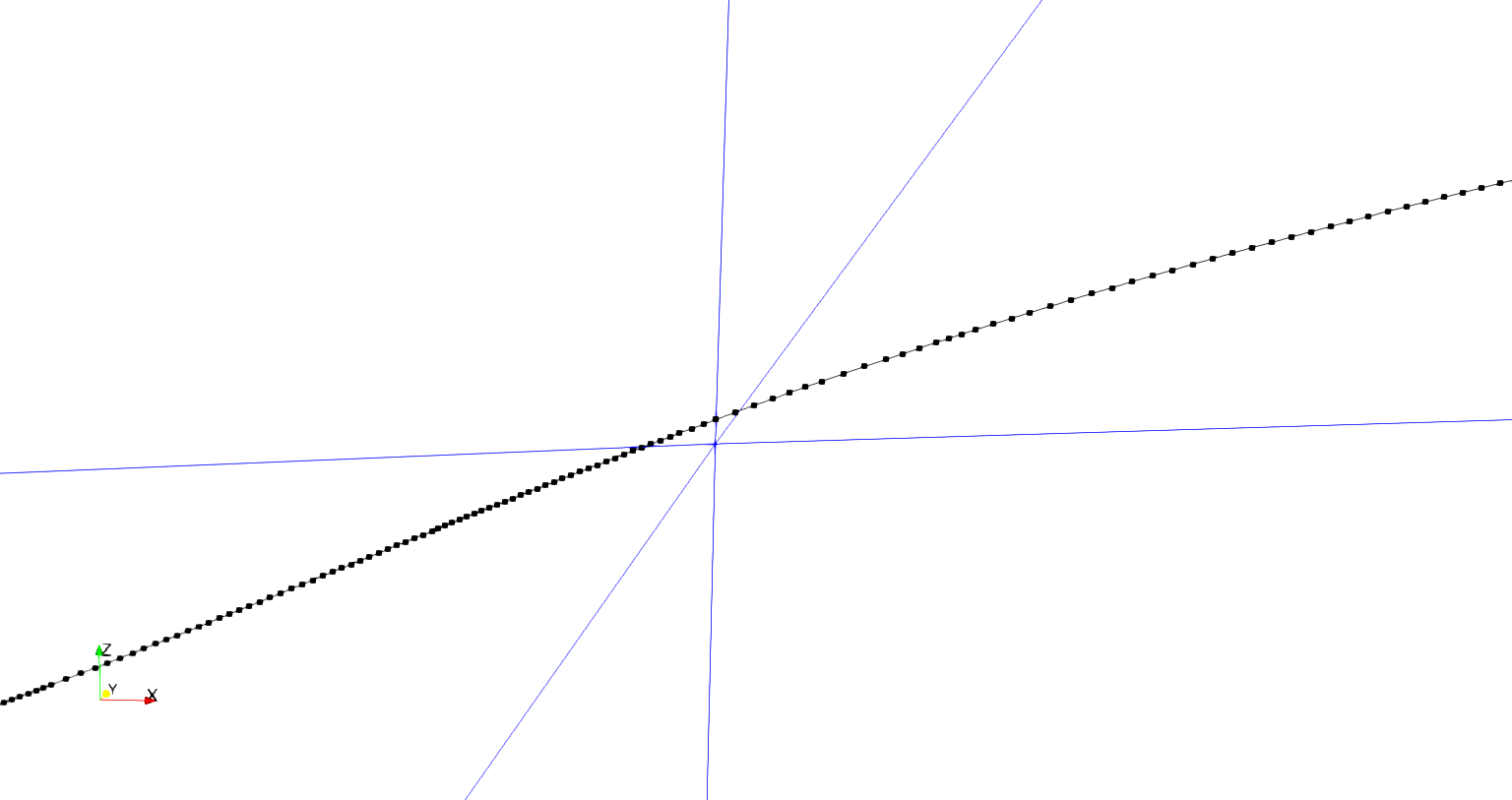


1. **Initial intersection curve and surface mesh**
2. **Intersection curve and resulting mesh after cut**



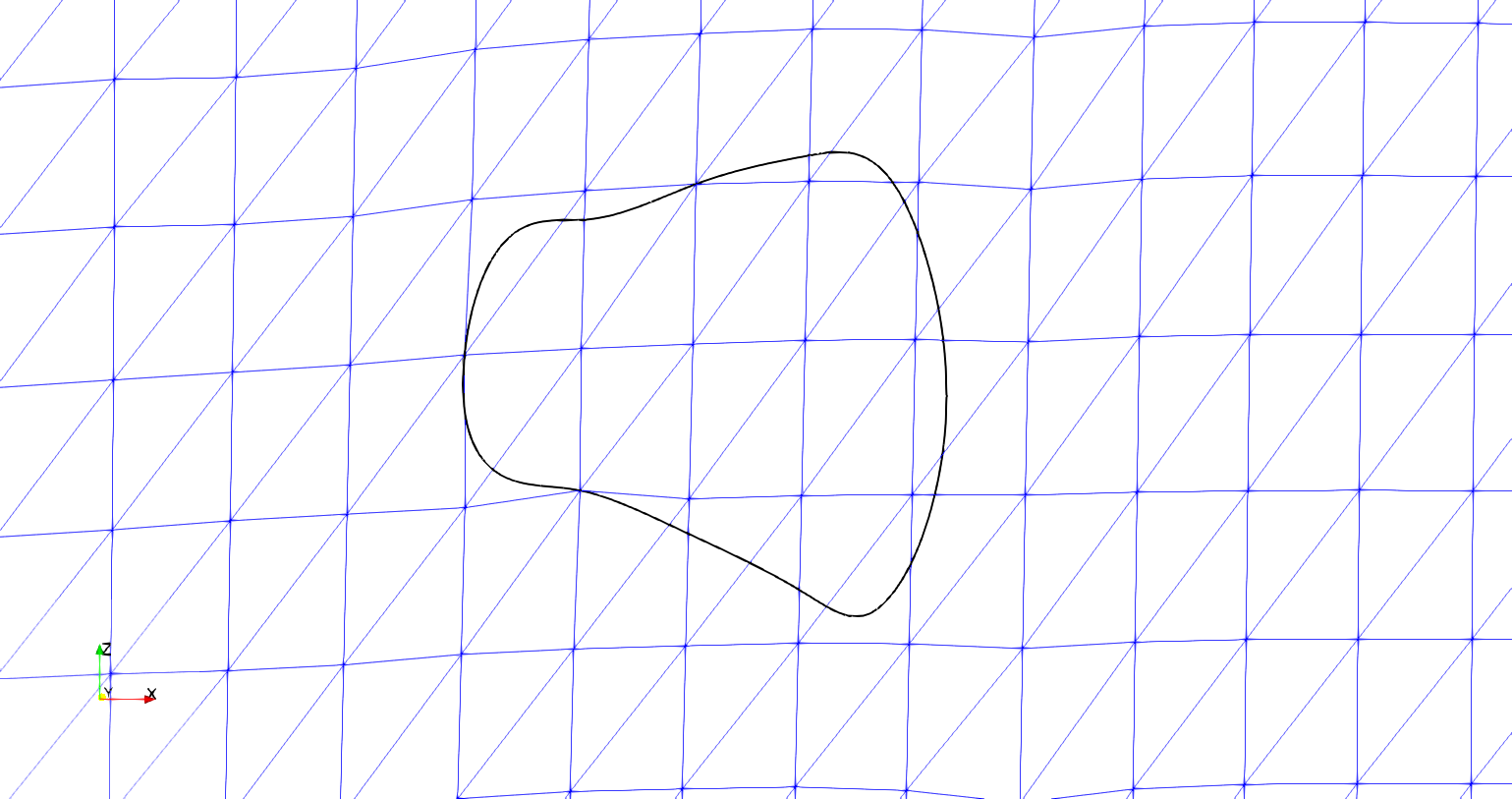
1. **Intersection curve and removed elements after cut**

The curve must be represented in a form so that the interpenetrating elements from the surfaces can be corrected or removed. The chosen solution is to cut the surface elements where they are intersected by the intersection curve and remove the interpenetrating elements from the surface mesh. Once the curve is used to cut 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 of the intersection curve to a node on the curve itself. Figure I shows a section of the upper part of the curve in figure H that is very close to a surface node on the surface mesh.

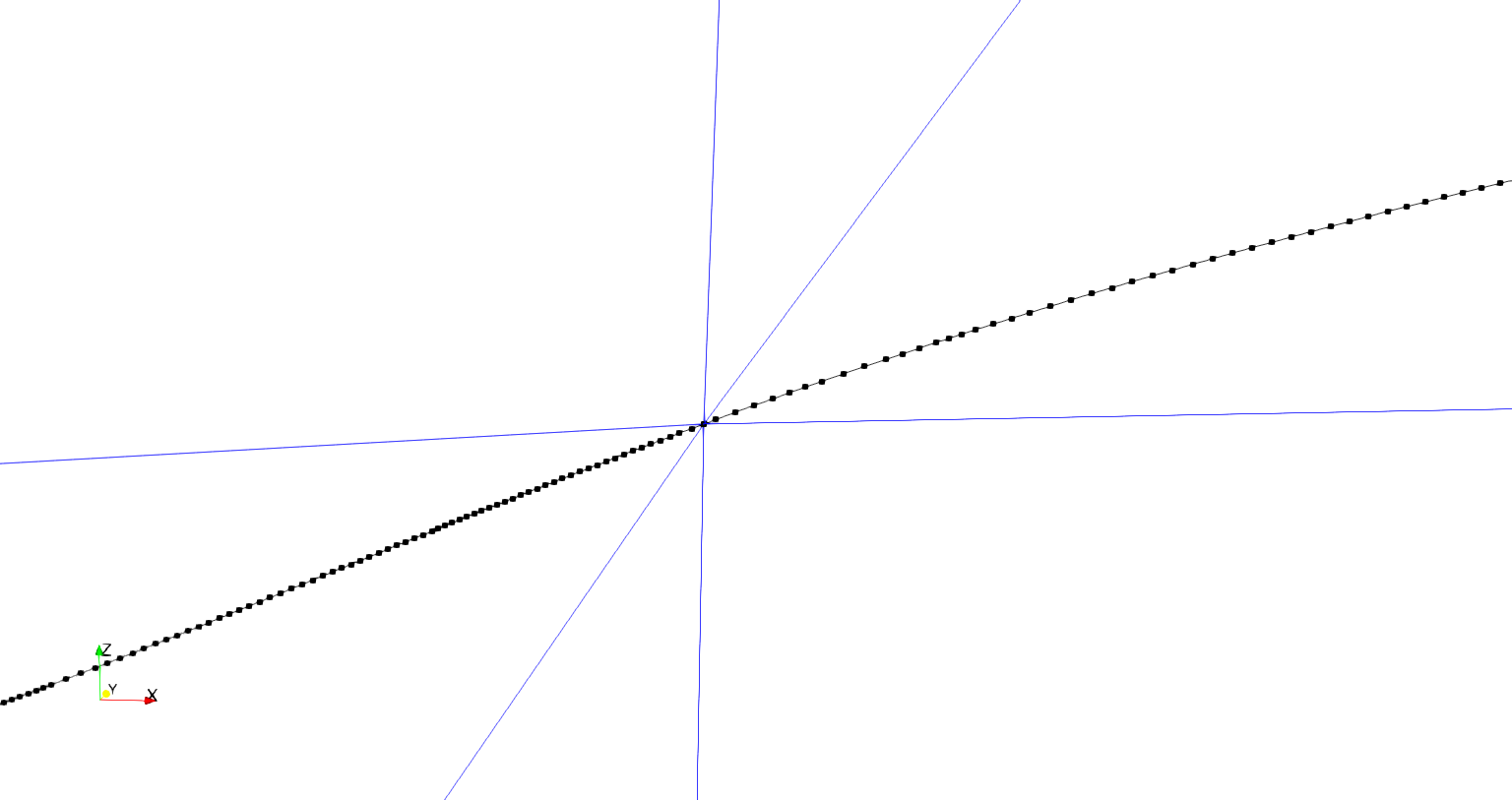


**Figure I: Intersection curve in close proximity to a surface mesh node**

When the intersection curve comes close to a surface node, the resulting mesh from cutting the surface has a much higher refinement in the area surrounding the surface node. This has negative consequences when performing an analysis. The solution is to move any surface nodes that are within a chosen tolerance to an existing node on the boundary curve, as in figure J. Because the relative refinement of the intersection 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 removes the need to cut the element. 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 an 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 a surface mesh whose surface nodes that lie close to the intersection curve have been moved to the curve. This eliminates the localized high refinement that is caused by a small corner of the element being cut by the intersection curve.



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



Once the surface nodes have been moved, the intersection points between the boundary curve and the surface mesh are calculated. The Separating Axis Theorem (SAT) is implemented to detect if an intersection curve segment intersects with a surface element. The SAT is discussed more fully in Appendix A. Before the SAT can be used, the intersection curve segment and the surface element must lie in the same plane. An assumption is made that any segment that intersects a surface element is nearly planar with the element because the intersection curve lies on the intersection of both surfaces. Therefore, if an intersection curve segment intersects an element edge or lies within the element edges, the boundary curve segment should lie on the same plane as the element. However, this is not guaranteed because of the approximation of the NURBS surface as a faceted surface mesh, which changes the intersection point of the meshes slightly. Need advice for a figure here showing a vertical separation of intersection curve and surface mesh. The intersection curves do not lie perfectly on the surface but it is difficult to get a comprehensible angle showing exactly what is happening. Therefore, when evaluating if part of the intersection curve intersects a surface element, the curve segment is verified to be in proximity of the element and then projected onto the surface element. Intersection curve proximity is verified by creating a vector between a point (**p**) in the plane of the element and the mid-point (**m**) of the curve segment (**s**) being checked, as in figure K. If the result is within user defined tolerance, the segment midpoint is verified to be in or near the plane of the element. Another check is made to verify that the segment is within a certain vertical distance by taking the dot product of the previously mentioned vectors and verifying it is within a set tolerance (**Dot product** in figure K). The last check is to project the segment (**s’**) onto the plane and complete the SAT. Projecting the segment onto the element plane is accomplished by determining the distance to the plane for each segment node and then moving the node along the element plane normal vector to the calculated distance. This shown in figure K by moving the segment **s** to **s’**. The SAT is used to determine if the segment, when projected onto the surface element, intersects or lies within the element. I know that we have the SAT in the appendix but I wonder if an additional figure for the intersection could be useful here. Once the intersected elements are identified, the points where the boundary curve segments intersect the edges of the surface element can be calculated.

Dot product

**m**

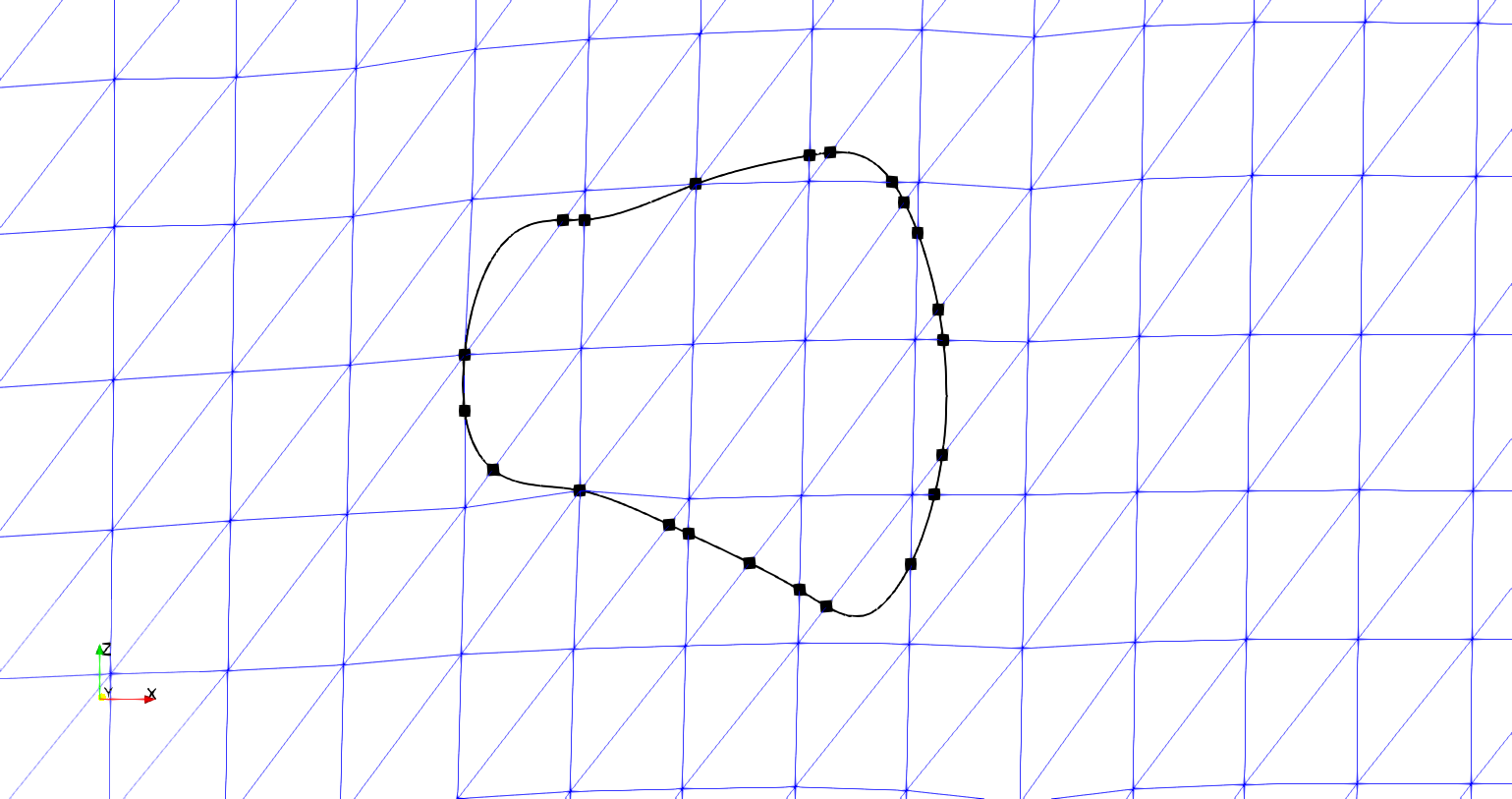
**p**

**Figure K: Illustration of proximity check algorithm**

**s**

**s’**

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

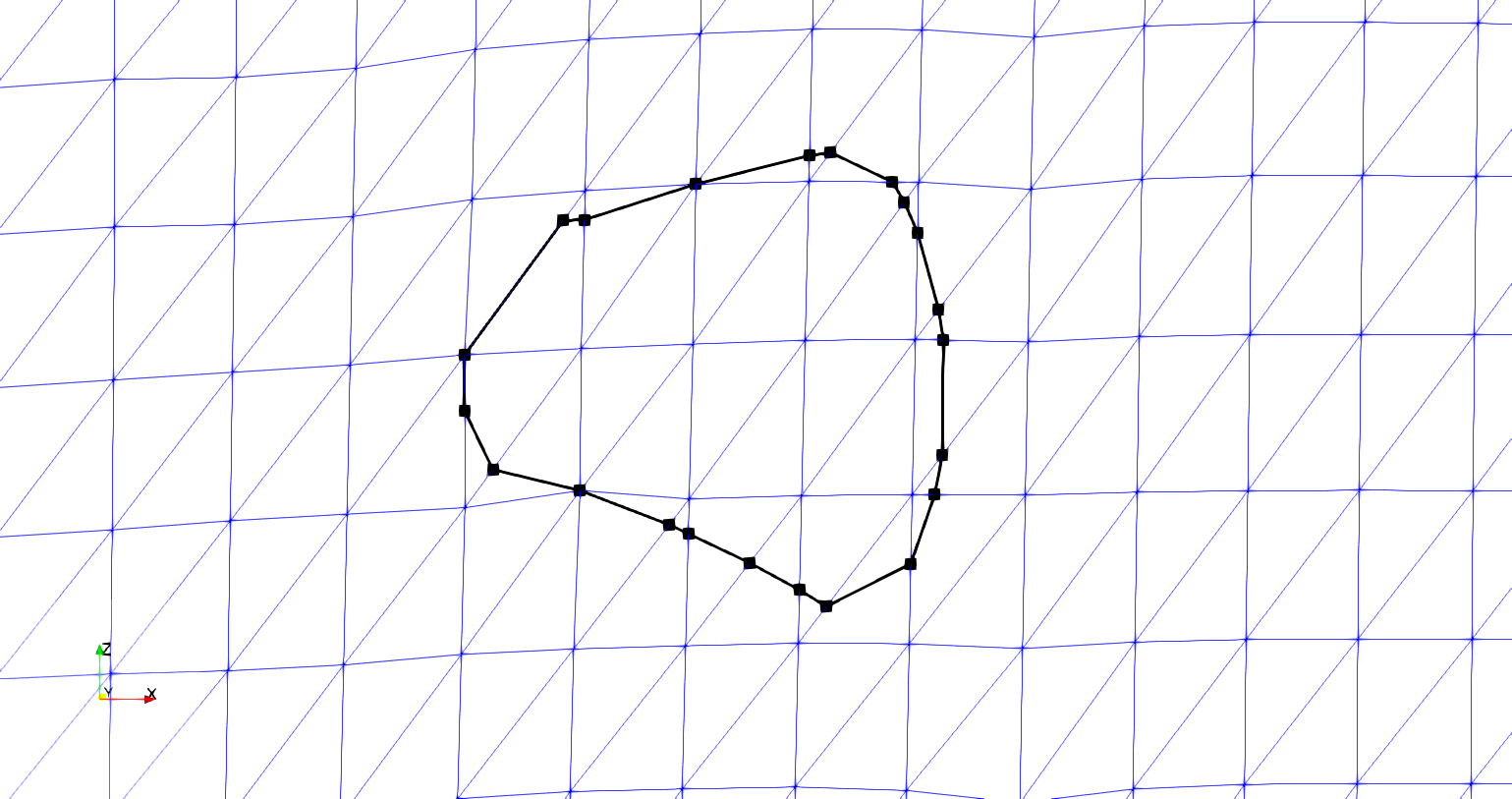


**Figure L: Boundary curve with marked surface element intersection points**

## The purpose and use of intersection points

The main purpose of dividing the intersection curves of surface at each element intersection point is to establish a basis for compatibility between the two surface meshes. The algorithm calculates the intersection points for both tow surfaces whose interpenetrations are bounded by the curve. 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 curve will have a set of points on the curve that their edges will line up with, which is required in traditional finite elements. The intersection points are inserted into the interpenetration boundary curve by first finding which curve segment the intersection point lies on. Then the segment is cut at the intersection point and the point is inserted into the curve. This occurs with every calculated element intersection point and results in a basis for compatibility of the surfaces along this intersection curve between the tow surfaces. Once this compatibility is created along this boundary curve, the curve can be used to cut the surface mesh and define the region in which surface elements should be removed. The element intersection points also serve a secondary purpose. 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 calculated before the boundary curve refinement is reduced, there is no loss of accuracy when removing interpenetrating nodes and elements. To reduce the intersection curve’s refinement, an iterative loop is run to remove any curve points between two consecutive intersection points. The result of the method can be seen in figure M.

It can be seen from figure M that all of the surface nodes inside the perimeter of the original intersection curve from figure L are still in the interior of the curve. Also, the relative shape of the curve is the same, indicating that the lower refinement does not overly affect the interpenetration boundary curve. Figure M is the result of capturing the interpenetration points of just one surface for illustrative purposes. However, once the intersection points from both curves have been added to the intersection curve, there will be intersection points that line within the elements in figure M as well. I am not sure how to discuss the effect of both surfaces in tandem. I could potentially show both results simultaneously and discuss them but I am afraid it may add more confusion.



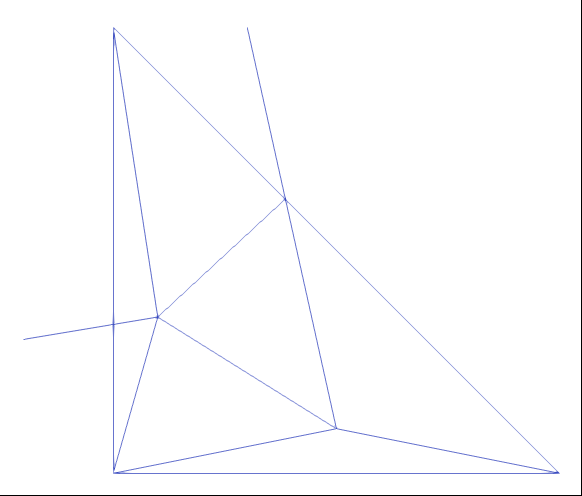
**Figure M: Reduced refinement boundary curve with marked intersection points**

Once the intersection points between the surface elements and the intersection curve of the surfaces have been used to reduce the curve’s refinement, the surface meshes are cut by curve. To cut the surface mesh with the intersection curve, the interpenetrating elements are first re-meshed to include the intersection curve. 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 specific element. Only elements intersected by the current curve being used to cut 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 involving the Separating Axis Theorem is used to collect intersection curve segments that lie within the current surface element being evaluated. Once the segments have been collected, they are checked against the elements 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 intersection curve segments.



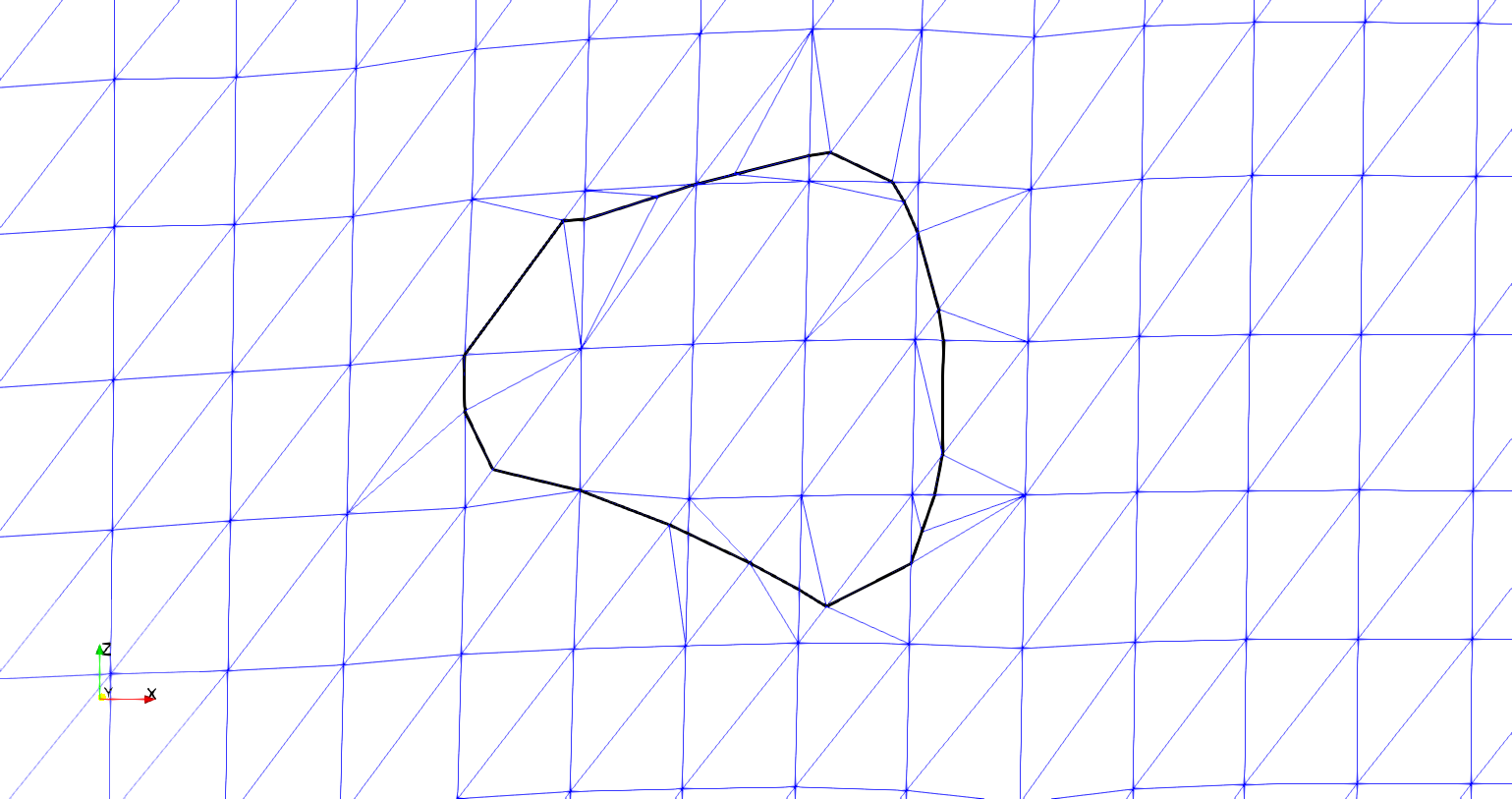
**Figure N: Example surface element with three boundary curve segments intersecting**

Figure N shows two boundary curve segments with endpoints lying on the surface element edges and one curve segment completely contained in the element. A mesh generation library called Triangle is used to create a new mesh of the surface element in figure H that includes the surface intersection curve. The library requires that all line segments, referred to as boundary segments by the library, that define the required boundaries to be included in the mesh be given to the library. These segments include all surface element edges and the intersection curve segments that lie within the element. Once the boundary segments are given to the library, a mesh of the single element and the intersection curve segments is returned, as shown in figure O.



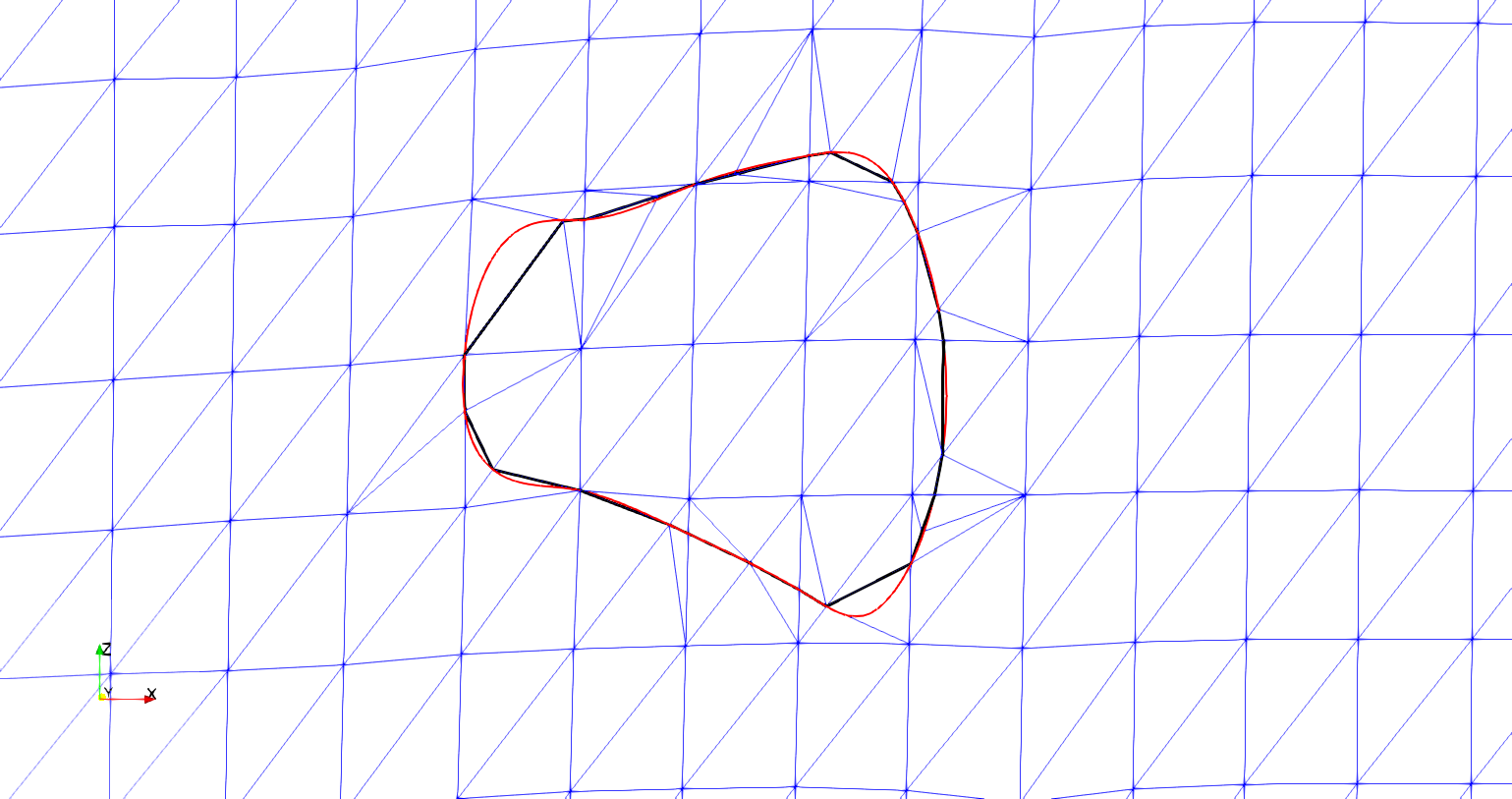
**Figure O: Sub-mesh of surface element with boundary curve**

The newly meshed element is then be added back into the surface mesh, replacing the original element. Figure P shows the result of the algorithm for the test case shown. Figure P shows both the relative refinement of the newly 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 Q shows the original boundary curve compared to the new curve used to re-mesh the surface.



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

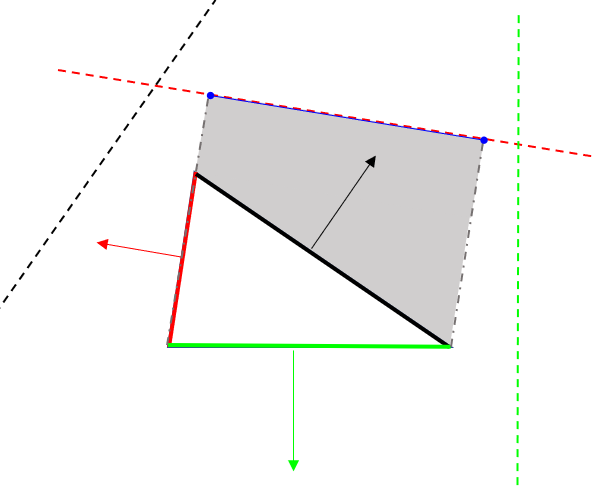
Figure Q shows how the overall shape of the boundary curve is still maintained and that no features such as surface nodes or elements are left out from the reduced refinement boundary curve. Some sections of the curve are not captured fully when the curve has been coarsened but this does not affect the ability of the algorithm to capture the correct interpenetrating elements or nodes between the two surfaces. The result is a mesh that includes the curve along which the surface mesh will be cut to remove the interpenetrating region. When the curve has been used to cut both surfaces, they will then share a curve along which there is 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 intersection curve between the two tow surfaces is also where the two surfaces are connected in a compatible manner. Using this curve, a compatible mesh can be created in the interior of the intersection curve that will be used to replace both surfaces elements that have been removed. The result is region of compatible mesh that both surfaces share, creating a connecting surface between the tow meshes. I think I could end with the same figure with the inner elements removed from the mesh like weve discuss as well but im not sure if it is redundant.



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

# Appendix

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 a shape onto pre-determined axes. The projection can be thought of as the shadow of the shape on an axis. The axes are created by taking the normal direction of an shapes edge and creating an imaginary infinite line in the same direction. Figure 1 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 1. Three axes are identified, one corresponding to each side of the triangle.



**Figure 1: Projection of a triangle on an axis**

**RA**

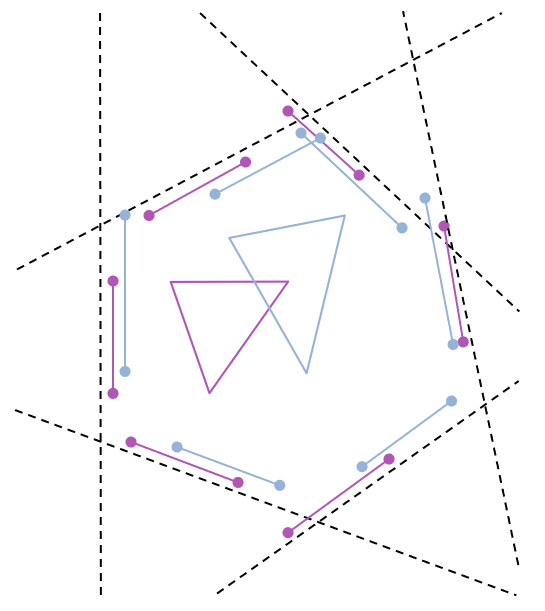
**REN**

**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 2.

In Figure 2.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 2.b shows the case when the two shapes do not intersect. The axes (dotted lines) are the same in 2.a and 2.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 2.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 a curve.



**a) SAT in which triangles overlap**

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

**Figure 2: Two cases for testing the Separating Axis Theorem**