3 Finite Element Model of Tree Leaf

3.1 3D Scanning

At the beginning, attempts were made to use the methods available in computer graphics literature to create 3D models of blade and leaf. But most of these models can only provide a 2D blade surface. The rest of the models can create 3D blade surface with a good amount of human interaction. The 3D effect in the blade is created by manipulating light reflection rather than actually presenting a 3D surface. As the main purpose of these models is to visualize leaf in a natural scene, they lack the capacity to precisely represent a leaf surface which is necessary for structural analysis. So, finally a 3D leaf model was developed by scanning a Bradford Pear leaf. The 3D scan data was converted into an IGES file which was later used in Abaqus to build the FE model. The 3D scan only captured the surface geometry of the leaf and was unable to retain the venation structure information of the leaf. So, a consistent method to reproduce the venation structure in that 3D model was required.

Figure 15: Bradford Pear Leaf

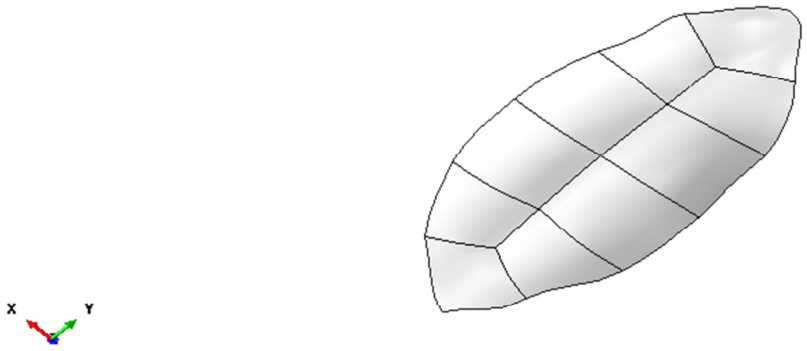


Figure 16: Scanned Leaf Blade

We studied the available literatures of computer graphics to find a consistent method for recreating leaf venation in a 3D blade surface. As the venation formation is not still well understood, the available methods only seem to perform well in specific cases. We expect, in future better generic models will be developed so that they can work well in all sorts of leaf surfaces. Finally we developed a method with minimum interaction which can approximate primary and secondary veins in a blade surface.

3.2 Creating Venation Structure

3.2.1 Initial Venation Data

A data structure was created to convey the initial venation structure information. Each vein was divided into small segments which were assumed straight lines. This line information consists of the initial node and end node on the line. The nodal reference was obtained from previously meshed blade surface. For example, if a vein segment can be represented by a straight line

between two nodes A and B respectively, on the blade surface, the initial vein data will represent this as a two element array {A, B}. The FE model created in Abaqus can be accessed with this node indexes to retrieve the coordinate information. The initial venation data is a list of these two element arrays. As of now, we are more interested to see how venation structure plays a role in structural dynamics of leaf than exactly reproducing the venation structure. So, this approximate method serves our purpose.

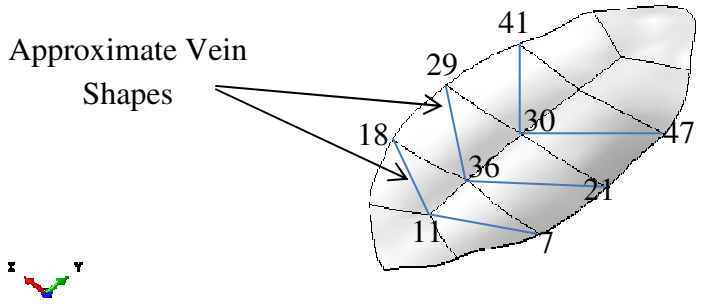


Figure 17: Representation of Initial Vein Data

In Figure 11, six vein segments are described approximately in terms of initial vein data. The numbers shown are the nodal indexes of start and end points. A list representing this initial vein data looks like the following:

$$I=\{\{11,18\},\{11,7\},\{36,21\},\{36,29\},\{30,41\},\{30,47\}\}$$

For each of these inputs from the list, an algorithm was developed to find the nodes in between the start and end nodes. Several criteria were tested to see which produce the most consistent

result and finally the accepted criteria was to find a point which makes the sum of distances from the point to end node and to its previous node minimum.

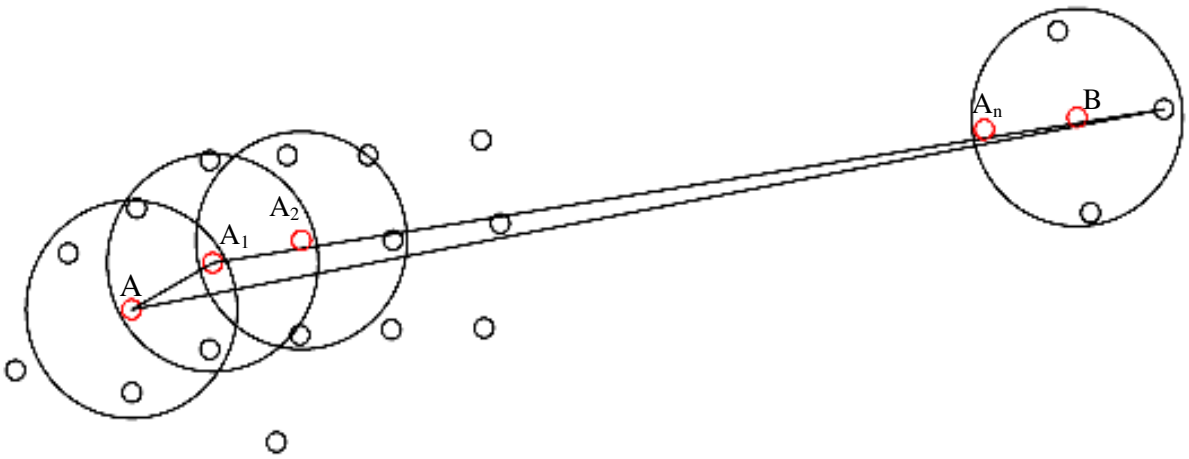


Figure 18: Vein Segment Generation

In Figure 12, A and B are the start and end node of the vein segment respectively. We create a circle of neighbor points by comparing the distance of rest of the nodes with A. This is done to reduce the computational effort in the next steps and also to constrain veins from deviating unexpectedly. Then we compare the distance from A to each neighboring points and from each neighboring points to end point B. Now, the next point in the vein segment is X, for which **AX+XB** is minimum, where X is in the neighbor points of A.

In the figure, $\mathbf{AA_1+A_1B}$ makes the minimum among all the neighbor points of A. So A_1 is the next point in the vein segment. Now, we get the next node X in the vein segment for which

A_1X+XB is minimum, where X is in the neighbor points of A_1. This gives us the next point A_2. This process in repeated until A_n, for which B is a neighbor point. Then the start point A, intermediate points A_1, A_2, A_3,… A_n and the end point B is arranged in a 1D array to be used later to draw the vein segment in Abaqus. By this method, for each node pairs in the initial vein data, a sequence of nodes are obtained that sufficiently defines the vein segment. If any irregularity is found in vein representation, the nodes can also be manually edited to correct the geometry.

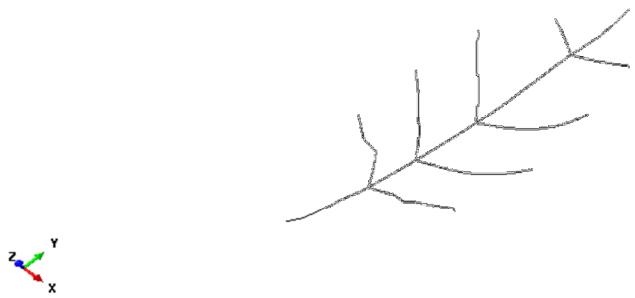


Figure 19: Vein Generated from Initial Vein Data

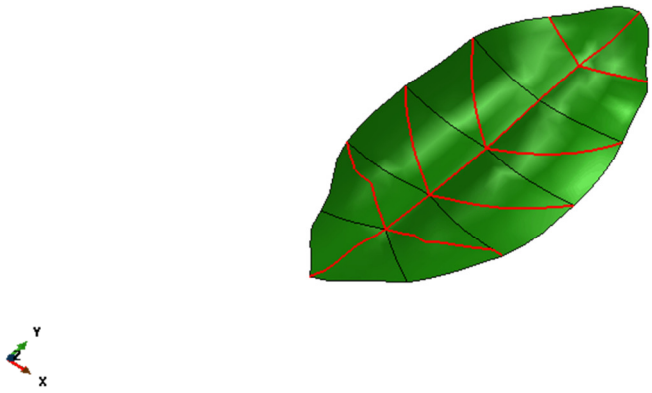
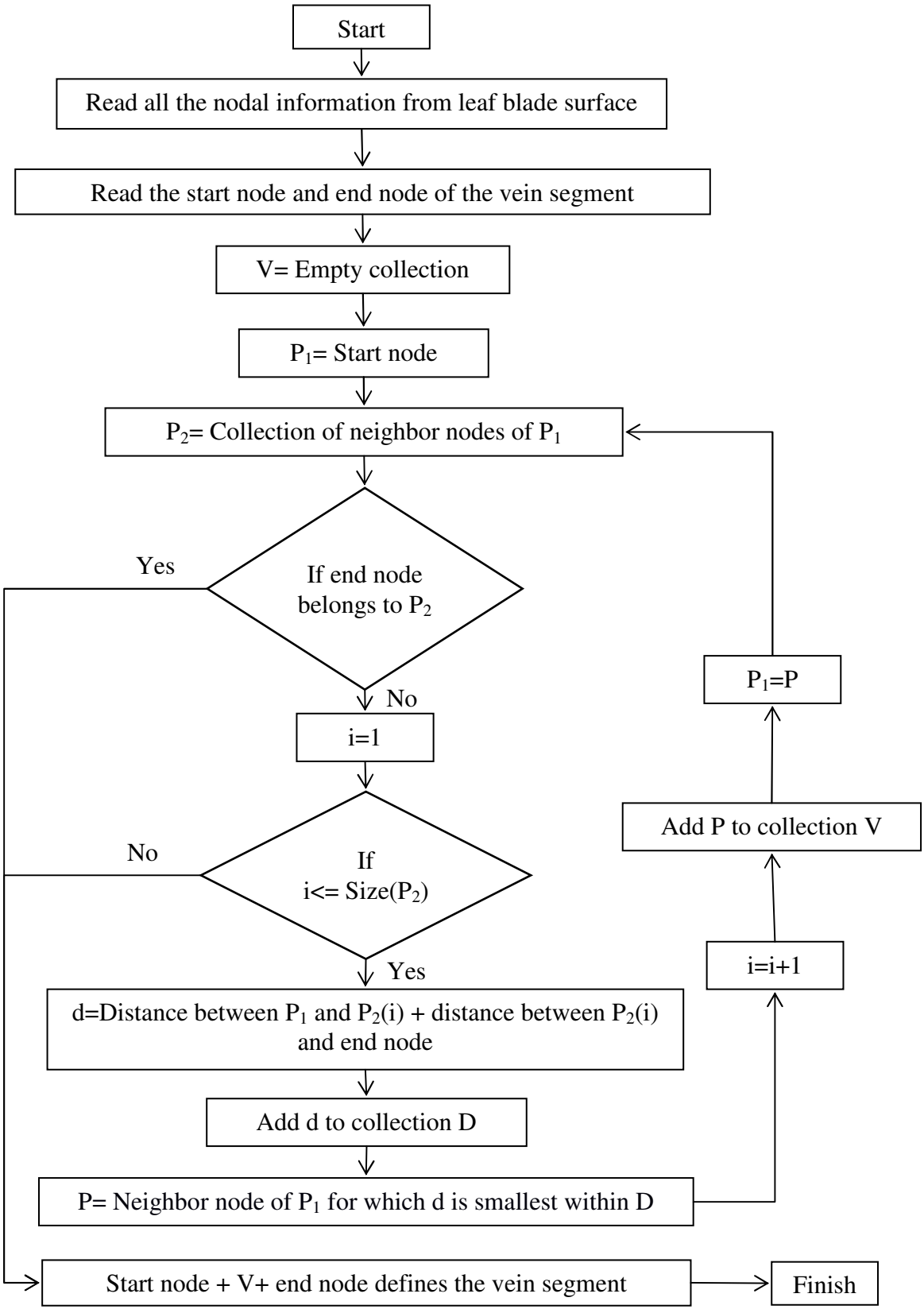
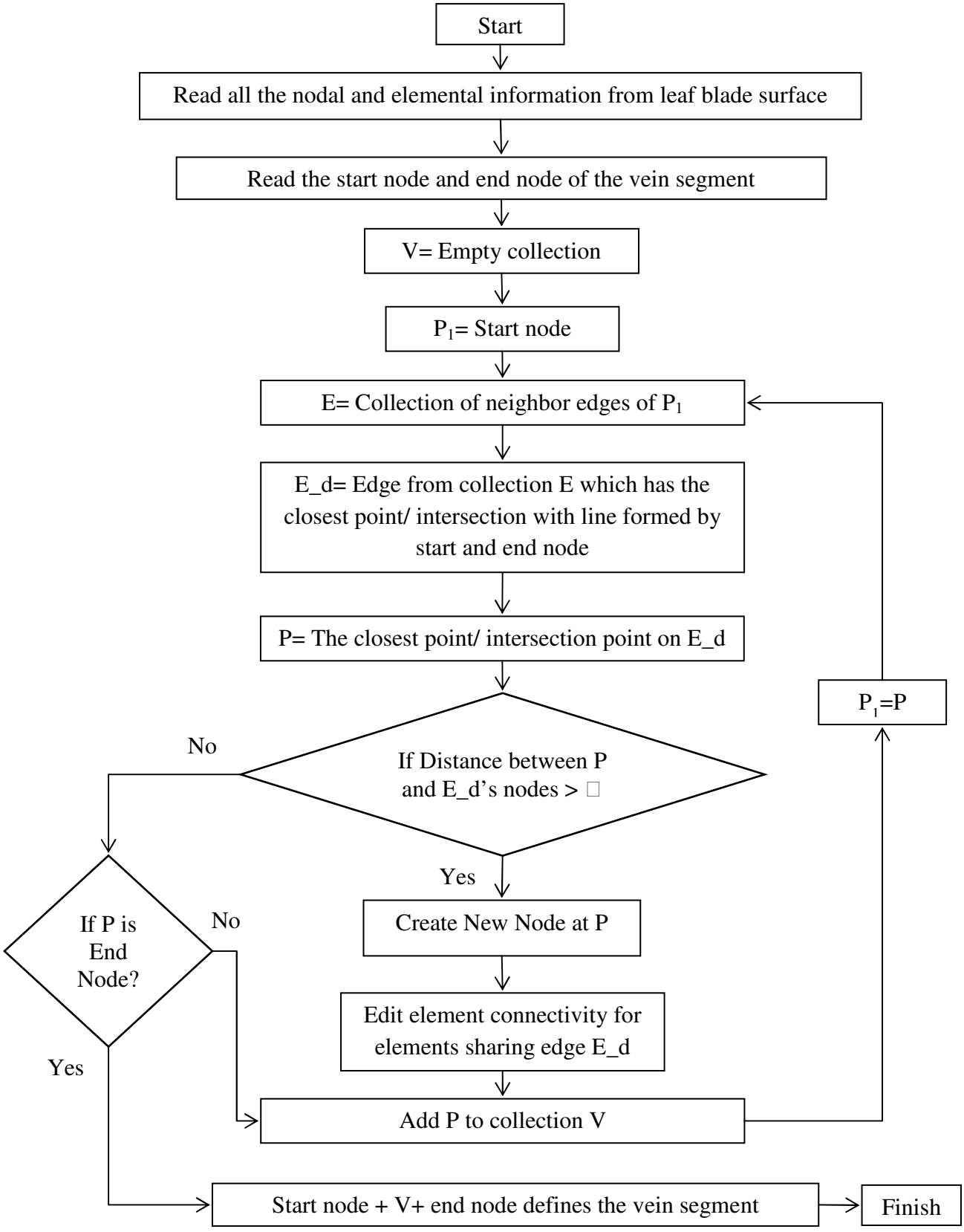


Figure 20: Veins Embedded into Leaf Blade

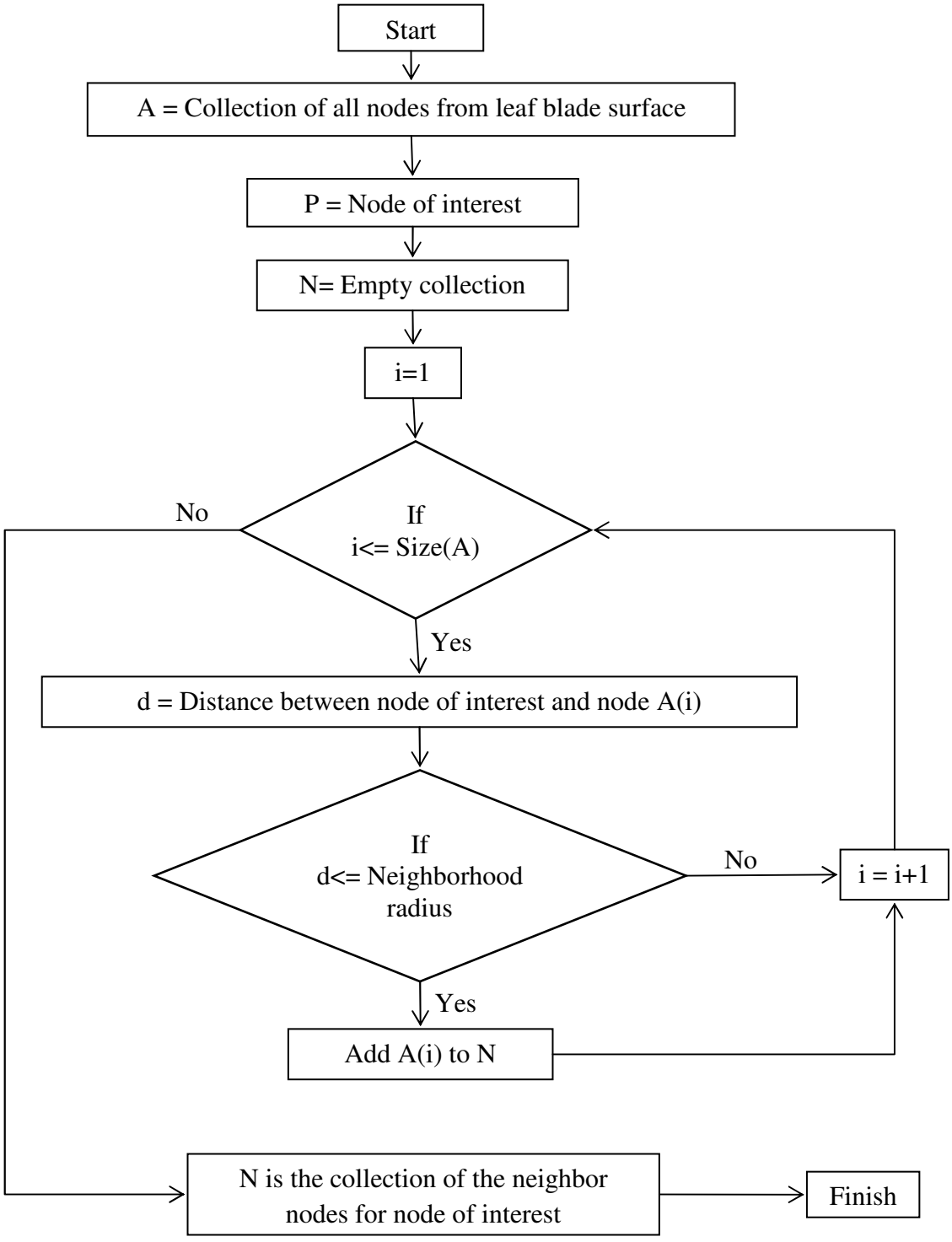
3.2.2 Algorithm 1 for Vein Generation



3.2.3 Algorithm 2 for Vein Generation



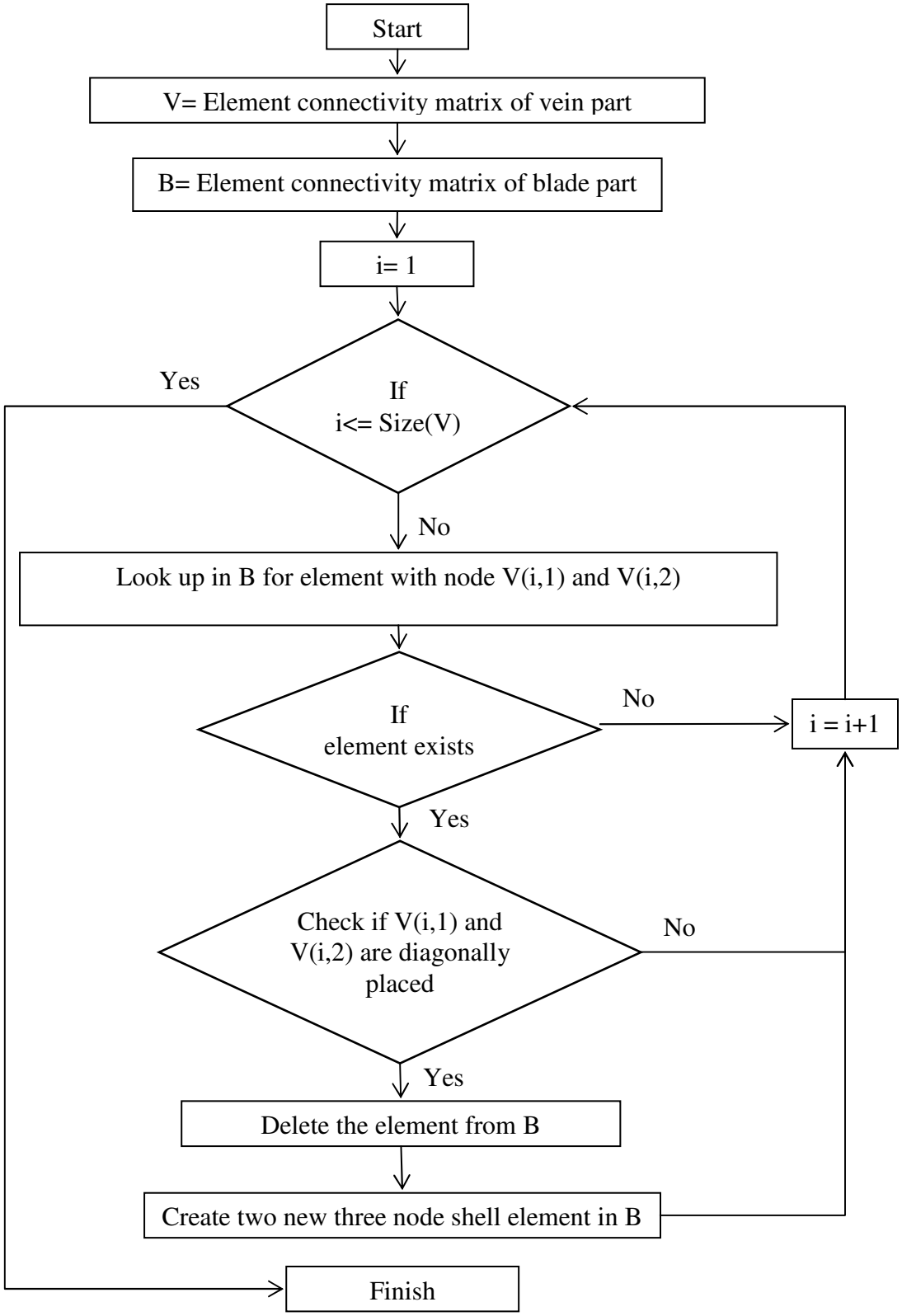
3.2.4 Algorithm for Finding Neighbor Nodes



3.3 Splitting Breached Element:

As the vein generation algorithm generates veins, some of the vein segment runs diagonally through four node shell elements which are used for modeling blade surface. This introduces inconsistency in the FE model. It happens because during deformation, the vein segment which is modeled as beam element does not stay in the same plane as the shell element. So even though the beam element and shell elements are tied at the nodes, they can go out of plane in between, which is inconsistent with the real leaf. We developed a patch test to see how this inconsistency affects the model which will be described latter. As it was found from the patch test that for those shell elements in which the beam elements runs diagonally, splitting the shell element into two three node shell elements along the beam element helps to remove the inconsistency. Now the beam element is collinear with one edge of the shell element and they stays together during deformation. The splitting of the breached element was done in two steps. First the breached elements were detected by comparing the element connectivity matrix of the blade part and the final venation structure nodal data. Later when Abaqus input file was prepared for analysis, a script was run to look for those breached elements in the input file and delete them. Then two new three node elements were added subsequently in the input file to replace each of four node breached elements.

3.3.1 Algorithm for Finding and Replacing Breached Elements



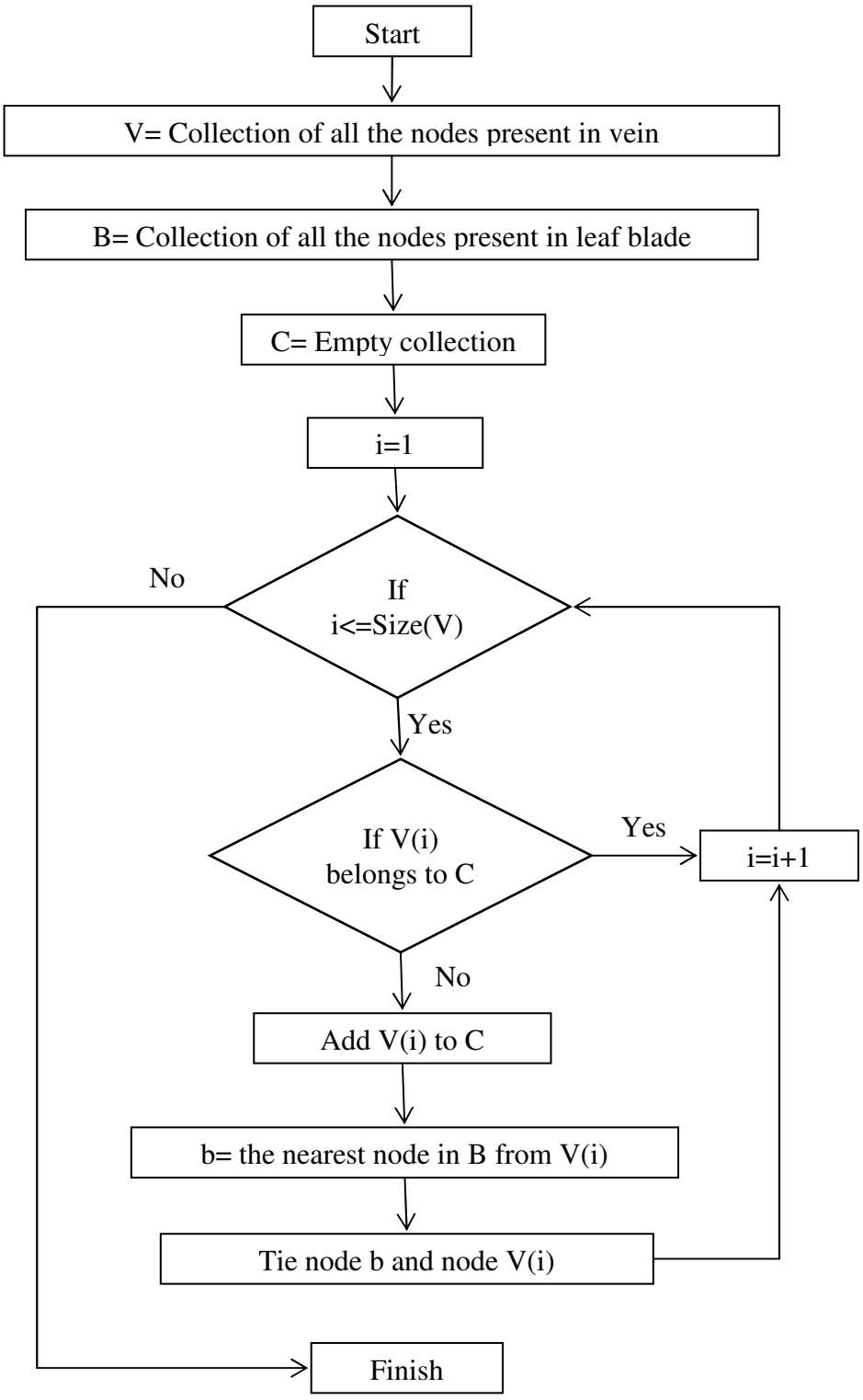
3.4 Constraining the Shell and Beam Nodes

Tie constrains were established between blade and vein nodes to make sure that they stay together during deformation. Each vein node was tied to the nearest blade node. A script was created to automate this tedious process. There are some nodes where multiple veins intersect. For those nodes, a check was done so that they get incorporated in tie constrains only once.

3.4.1 Patch Test for Tie Constrain

A patch test was developed to test different tie constrain methods between the vein nodes and blade nodes. A square domain was meshed with 9 shell element. Then beam elements were placed along the edges of the shell elements. Abaqus has three different tie constrain options. The whole blade surface can be tied to the whole venation structure surface, or all the blade nodes can be tied together with all the vein nodes, or individual vein nodes can be tied with nearest blade node. After performing modal analysis, each four node shell element was splitted into two three node shell elements. This remeshing should not affect the result if the tie method is consistent. The first two tie methods do not provide the same result before and after the remeshing while tying individual vein nodes to nearest blade nodes produces consistent result.

3.4.2 Algorithm for Setting Tie Constrains



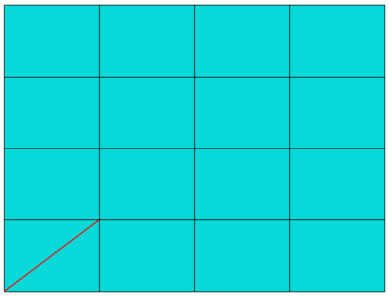
4 Result and Discussion

In this chapter, the results of vein generation algorithms, modal analysis of tree leaf are presented. The results of vein generation algorithms are first presented for a square blade and then final results for a Bradford Pear leaf is also presented. For the modal analysis of Bradford Peer leaf, first 10 mode shapes are presented. We also studied the effect of vein stiffness in the natural frequencies of the leaf structure.

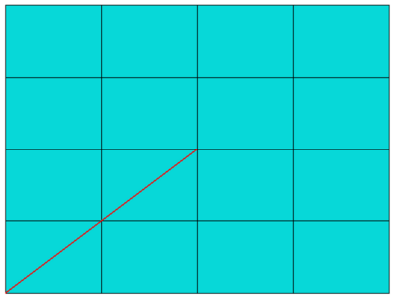
4.1 Regeneration of Vein

4.1.1 Algorithm 1

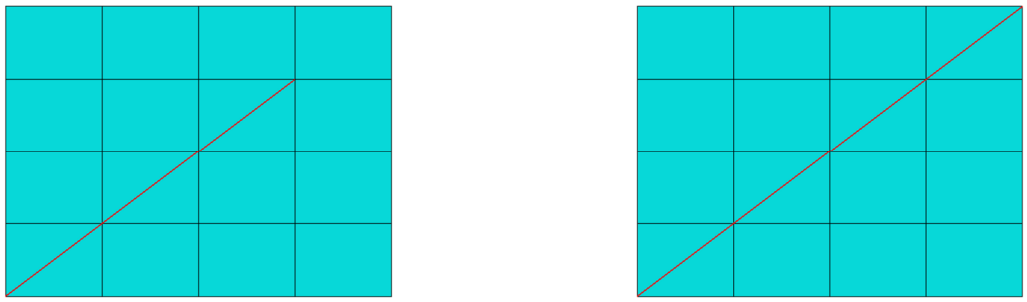
We use algorithm 1 to regenerate a vein between the two corners of a square.



(a)



(b)



(c) *(d)*

Figure 24: Vein Generation Algorithm 1 on a Rectangle with Diagonal Vein

As we see in the Figure 18, the algorithm identifies the intermediate nodes between the start node and the end nodes to regenerate an approximate vein. But, we also found the algorithm does not always produce desired result. For example, if we want to regenerate a vein between node 1 and 24, we do not get the expected result. In those cases, algorithm 2 works better.

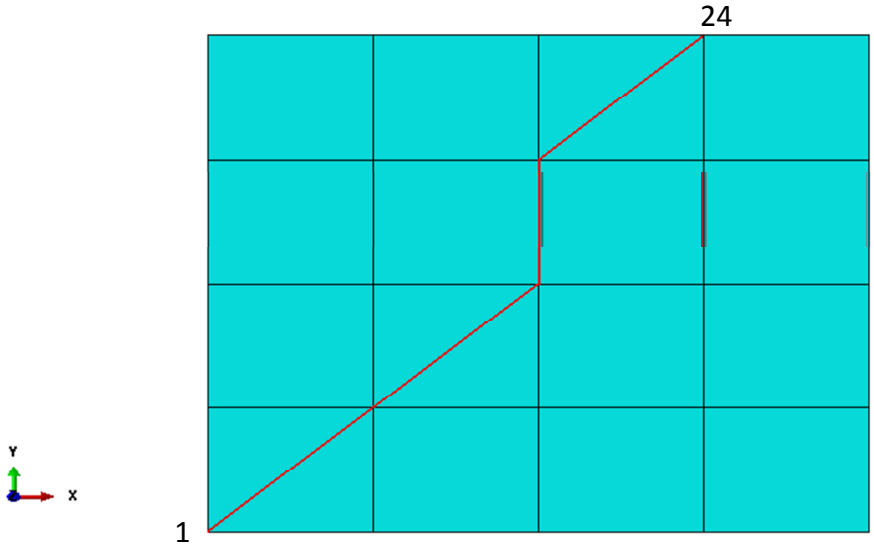


Figure 25: Poor Performance of Algorithm 1

4.1.2 Algorithm 2

As we see the insufficient performance of algorithm 1, we employ algorithm 2 in these cases. The workflow process is first to generate the vein using algorithm 1 and then if any unrealistic

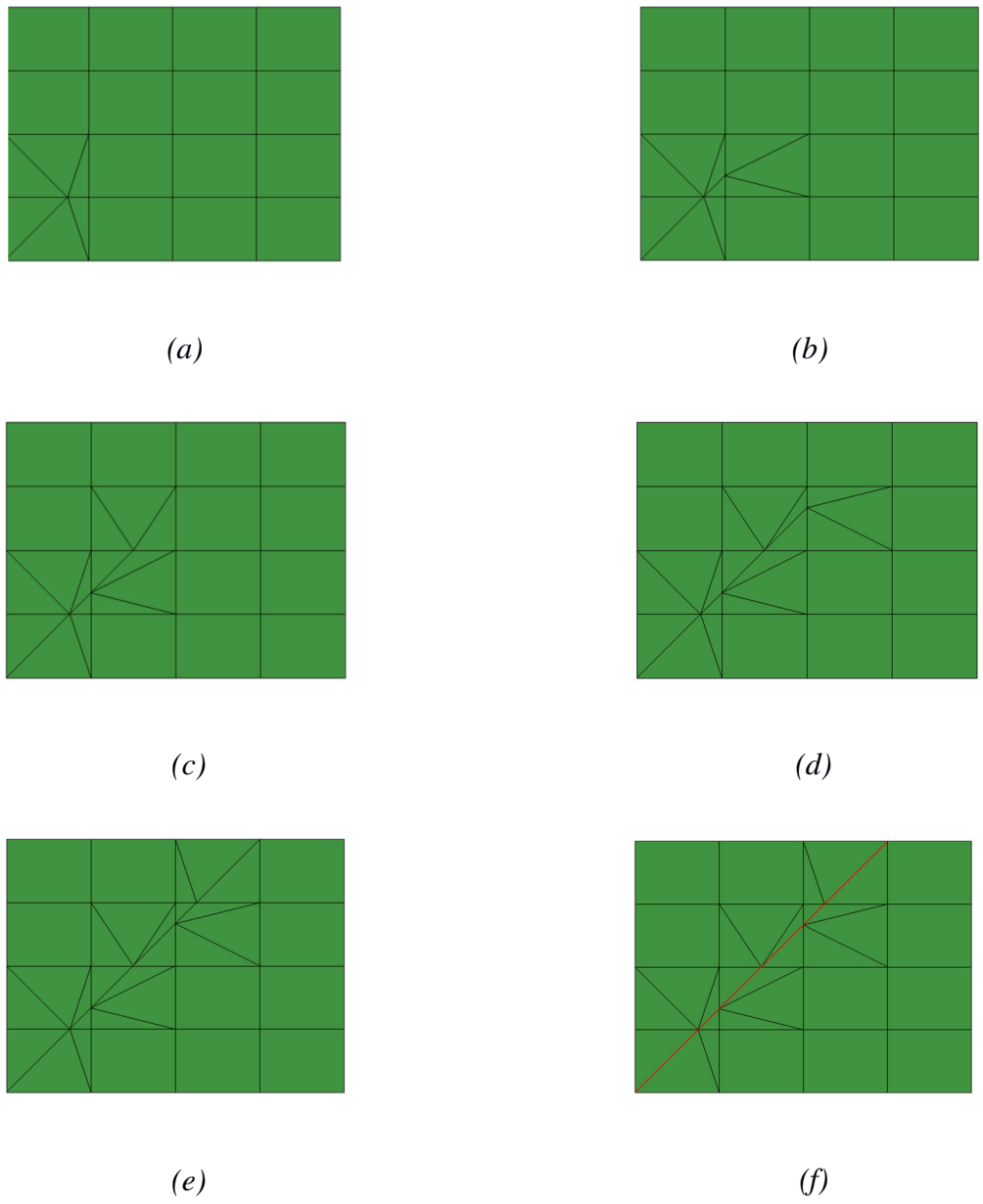


Figure 26: Application of Algorithm 2 (Step by Step)

result is found, the vein segment is corrected using algorithm 2.

4.2 Correction of Breached Element

As discussed in Chapter 3, all the shell elements, which have beam element running across, need to be splited along the beam element. We use the square plate with vein running across as the test case and perform the unit-test of the algorithm.

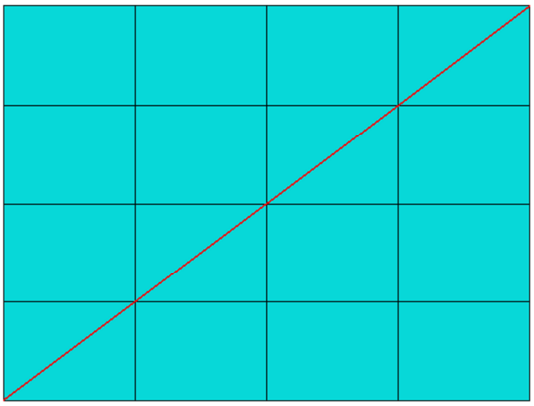


Figure 27: Rectangular Blade with Diagonal Vein

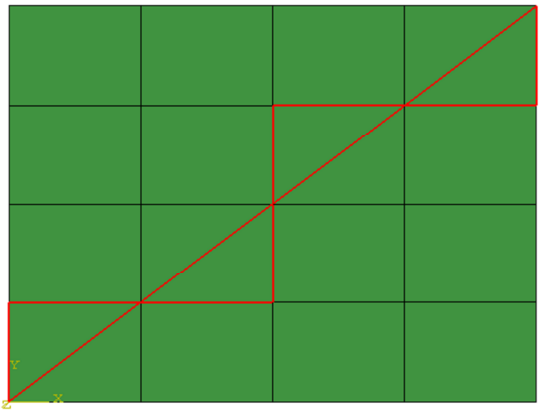


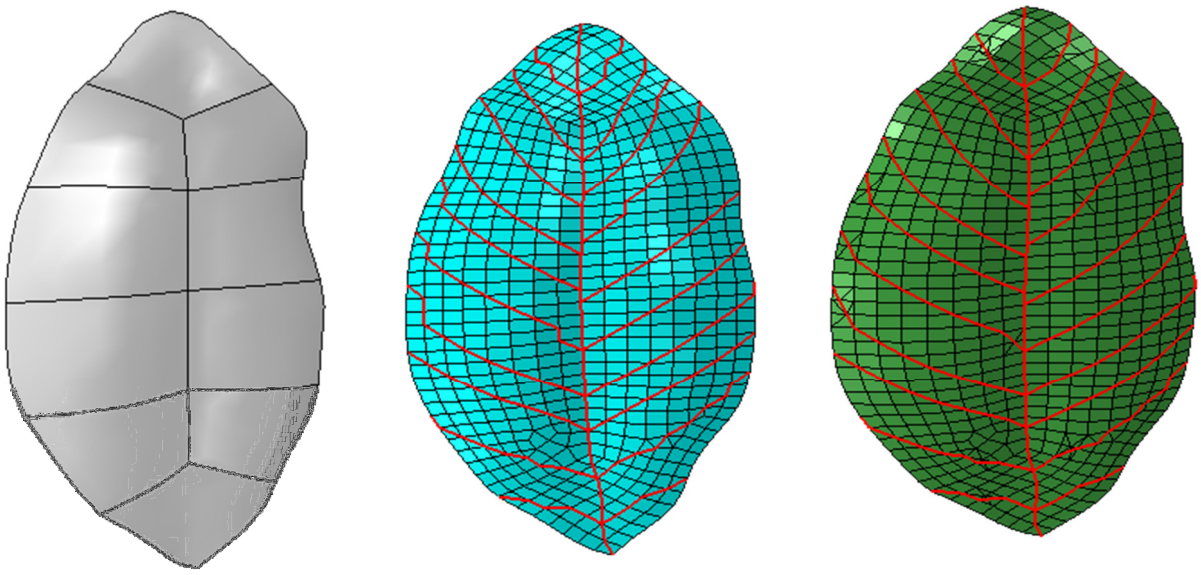
Figure 28: Splitting of Breached Elements

As we can see in Figures 21 and 22, the algorithm identifies the breached elements and splits each quadrilateral element into two triangular elements.

4.3 Application on Actual Blade Surface

All these algorithms were used in a Bradford Pear Leaf scanned model to generate a venation structure that closely resembles the actual venation structure. The length of the leaf from petiole

to tip was 108.49 mm and the maximum width was 50.69 mm.



(a) *(b)* *(c)*

Figure 29: (a) Scanned Blade Surface, (b) After Using Algorithm 1 with Approximate Venation Definition, (c) After Correcting Particular Veins Using Algorithm 2

4.4 Mesh Refinement and Convergent Study

The convergence study was performed for the model with no veins. The first five natural frequencies were monitored. The convergence plots show that we get convergence around 100000 DOFs.