SLICING ALGORITHM FOR COMPLEX CONTOUR GENERATION FROM COARSE MESHED ISO-SURFACE DATA

By

DIVYA HARESHKUMAR SHUKLA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

Of the Requirements

For the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

Acknowledgements

I would like to express my gratitude to my advisor Dr. Robert Taylor for his motivation and continuous support. His immense knowledge and guidance helped me a lot during the tenure of my research work. I highly acknowledge his patience and inspiration that helped me overcome crucial challenges in the research.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Kent Lawrence and Dr. Bo P Wang for their valuable time, encouragement, and insightful comments.

Finally, I would also like to thank Dr. Nomura and Mechanical & Aerospace Department of the University of Texas at Arlington for this valuable opportunity.

November 17, 2016

Dedication

I would like to dedicate my work to my father- Late Mr Hareshkumar Shukla and my mother Mrs. Bharati Shukla for their emotional and financial support. I would also like to thank my grandparents for their blessings. Thanks to my brother and my all friends who have played an important role during my academic journey. I would like to take a moment and thank few very important people without whom this would have been impossible- Harshal Patel, Prawal Sharma and Romanch Patel.

November 17, 2016

Abstract

SLICING ALGORITHM FOR COMPLEX CONTOUR GENERATION FROM COARSE MESHED ISO-SURFACE DATA

Divya Hareshkumar Shukla, MS in Mechanical Engineering

The University of Texas at Arlington, 2016

Supervising Professor: Name: Robert Taylor

Finite element based topology optimization is used to develop complex geometric configurations for additive manufacturing. Complex coarse-meshed Iso-surface data is extracted from topology optimization results and smooth mathematical geometry must be fit to that data in order to accomplish further design development. To accomplish this geometry fitting, cross-section data must be extracted at specific locations. The stereo lithography (STL) file format is a triangular mesh format which can be developed using 3D CAD Software and is widely used for additive layer manufacturing processes. The main aim of this research work is to generate cross-section contours using a nearest tail-to-head distance algorithm between the intersection of a plane and triangles from the Iso-surface. This work defines a function to find intersection points when an Iso-surface object is sliced by a given plane. After identification of intersection points, the function removes coincident points to form a closed, piecewise linear contour. This approach uses the first point in the matrix and identifies the nearest point by using distance analysis. The results show successful implementation of the head-to-tail distance analysis algorithm. Contours are generated for different geometry and cutting plane position and orientation.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

In this chapter, Concept of Design optimization, Topology optimization and Additive Manufacturing are discussed. Research objective and motivation are introduced in detail.

1.1 Design Optimization

Optimization is a process that provides us a better design that a human being may not be able to find through experiments. Design optimization is defined as a problem in which design variables can be determined to achieve objective function at given constraints.

The design optimization process is illustrated by the following flow chart. Initially design variables, objective function and constraints are defined by the designer. Design variables are the parameters that can be controlled by the designer. Constraints are the conditions which must be satisfied in order to achieve the objective function. The Objective function is a value which is minimized or maximized. For example, in a manufacturing process, we may want to maximize the profit or minimize the cost. If the objective function does not converge to a solution, the design variables have to be changed.

$$minimize\ f(x) = 0 \tag{1}$$

$$where\ g(x) = 0 \text{ and } h(x) = 0 \tag{2}$$

In the above equation f(x) is the objective function, x is the design variable and g(x) and h(x) is constraints. [1]

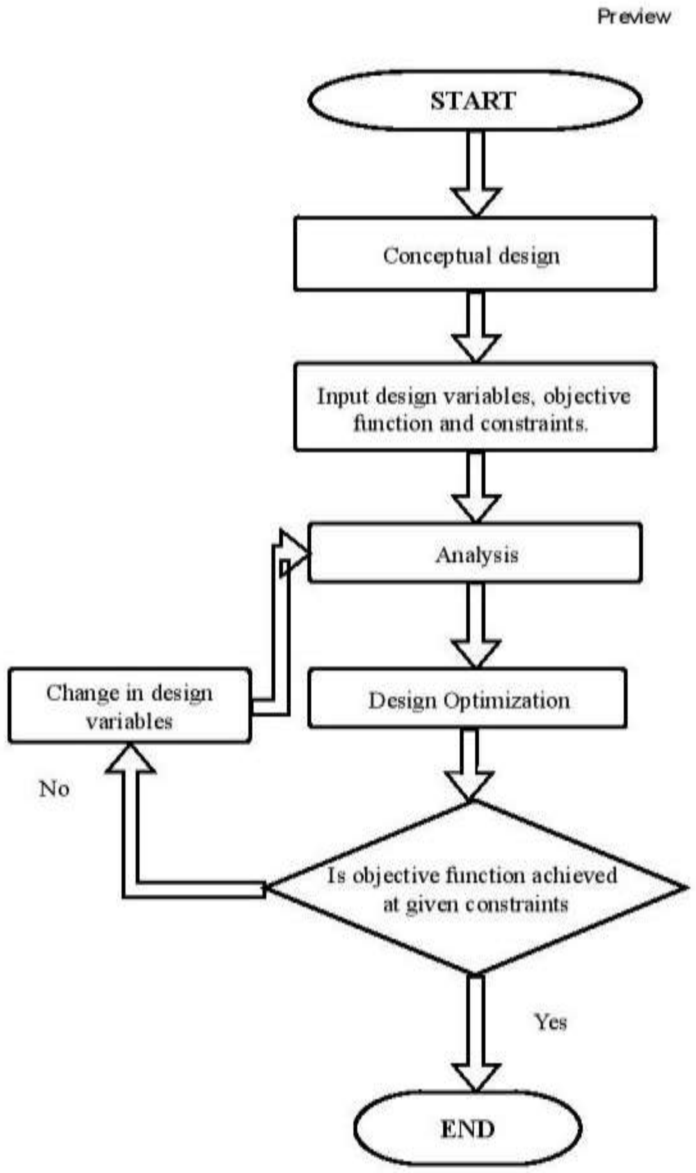
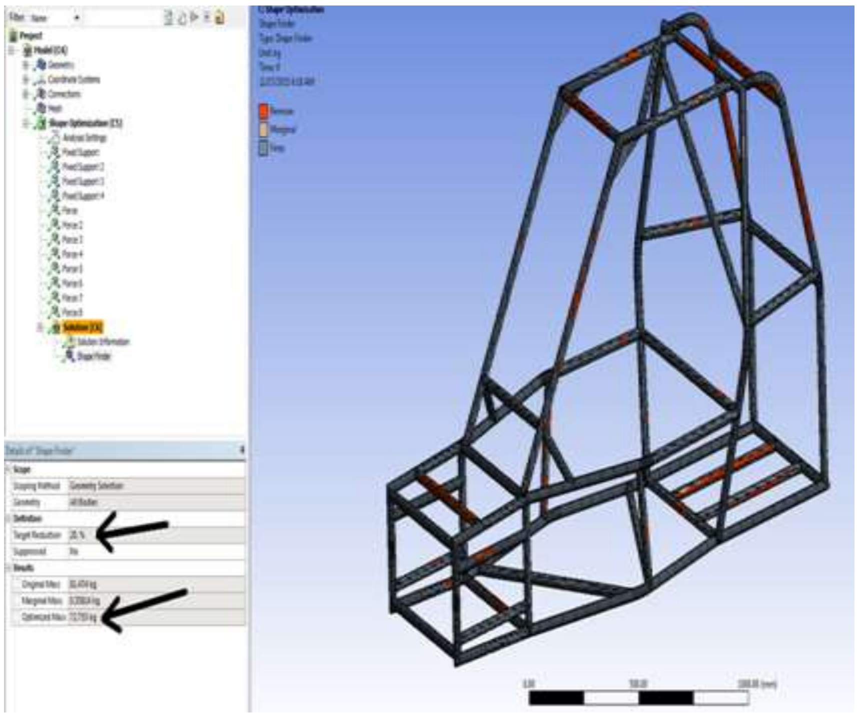

Preview
START
Conceptual design
Input design variables, objective function and constraints.
Analysis
Change in design variables
Design Optimization
No
Is objective function achieved at given constraints
Yes
END

Figure 1-1 Flow chart Model for the Design Optimization Process

1.1.1 Shape Optimization

Shape Optimization is defined as the process that obtains the shape which is optimal in order to minimize the objective function while satisfying given constraints. This method uses a variable controlling shape of the object to improve overall design. In shape optimization, voids which contribute to the initial design layout should be constant.

Here, the shape optimization tool is used to optimize the original roll cage of Mini Baja car. The objective function is defined as minimizing the weight and is shown in the below figure 1-2. Maximum yield stress is a constraint and density is the design variable of this project. [2]

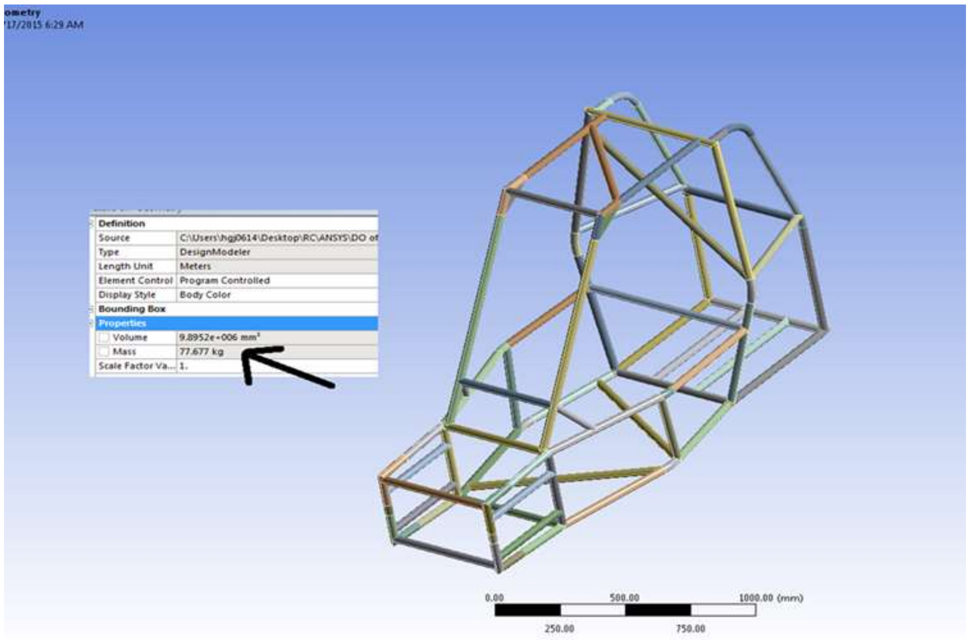


Details of "Shape Finder"

Scope	
Scoping Method	Geometry Selection
Geometry	All Bodies
Definition	
Target Reduction	20 %
Suppressed	No
Results	
Original Mass	30.404 kg
Marginal Mass	0.20814 kg
Optimized Mass	23.793 kg

Figure 1-2 Optimized Roll Cage with a reduced mass [2]

The optimized roll cage is shown in the figure 1-2. The members that do not affect the maximum stress in the roll cage are removed. Thus, the objective function is optimized at given constraints. The Optimized roll cage with the analysis is show in the figure 1-3.



ometry
17/2015 6:29 AM

Definition	
Source	C:\Users\hgj0614\Desktop\RC\ANSYS\DO of
Type	DesignModeler
Length Unit	Meters
Element Control	Program Controlled
Display Style	Body Color
Bounding Box	
Properties	
Volume	9.8952e+006 mm³
Mass	77.677 kg
Scale Factor Va...	1.

Figure 1-3 Weight Reduction in Roll Cage with Shape Optimization [2]

1.1.2 Size Optimization

Size optimization involves changing the parameters of the design such as material properties, thickness, cross-sections or operating conditions to achieve better design.

Size Optimization mainly involves varying physical properties such as thickness of the material and dimensions. It is widely used to manufacture composite components as it provides ideal layout of the composite ply thickness and helps to reduce time in the manufacturing process. [3]

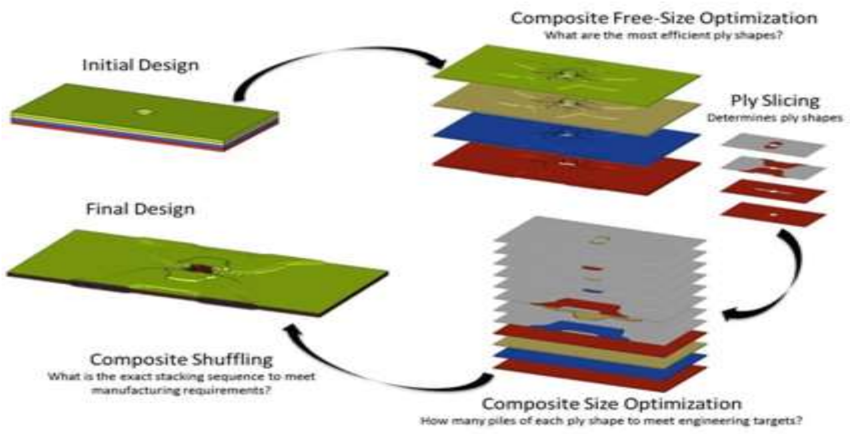

Composite Free-Size Optimization
What are the most efficient ply shapes?
Initial Design
Ply Slicing
Determines ply shapes
Final Design
Composite Shuffling
What is the exact stacking sequence to meet manufacturing requirements?
Composite Size Optimization
How many plies of each ply shape to meet engineering targets?

Figure 1-4 Size Optimization for the Composite materials [3]

1.1.3 Topology Optimization

Topology optimization is a computational method which optimizes the material distribution layout for the structure without any predefined baseline shape within a design domain. In topology optimization, Material distribution is obtained for given loads and boundary conditions by minimizing or maximizing the objective function within design space. Topology optimization is useful to find innovative and high-performance design layouts. [4]

Topology Optimization process can be divided as following:

- Problem Definition
- Material Distribution
- Optimize material distribution for better results and to achieve objective function, domain and boundary conditions
- Evaluate results

Topology optimization problem is also known as the binary programming problem. Formulation of the binary programming problem is to find solid and void which is referred to as “black and white” layout to minimize work done by external force at given volume. The binary compliance problem is known to be ill-posed (Kohn and Strang 1986a, b, c). This approach reduces the structure’s compliance. To understand this process, assume a design that has one big single hole that can be replaced by n number of holes at the same mass with lower compliance. [5]

Density-based approach

Density based approach is commonly used to solve topology optimization problem. In this approach, material distribution is parameterized by the material density distribution. [5] The result of topology optimization in design domain is entirely material or no material. Thus, density function approximation varies between 0 (void) and 1 (solid). [6] If an intermediate density of a structure is 0 or 1, the design is a “black and white design” for which performance has been evaluated with correct physical model. [6]

Commercial topology optimization tools like inspire, hyper mesh etc. generate surfaces from CAD model that are too noisy and has many defects. As illustrated in the figure 1-5, recent topology optimization tools are not suitable for interpretation of complex geometries with advance surfacing. For complex geometry from topology optimization results to be produced with Additive Manufacturing process, proper interpretation and tools are needed to create smooth geometry from the results.

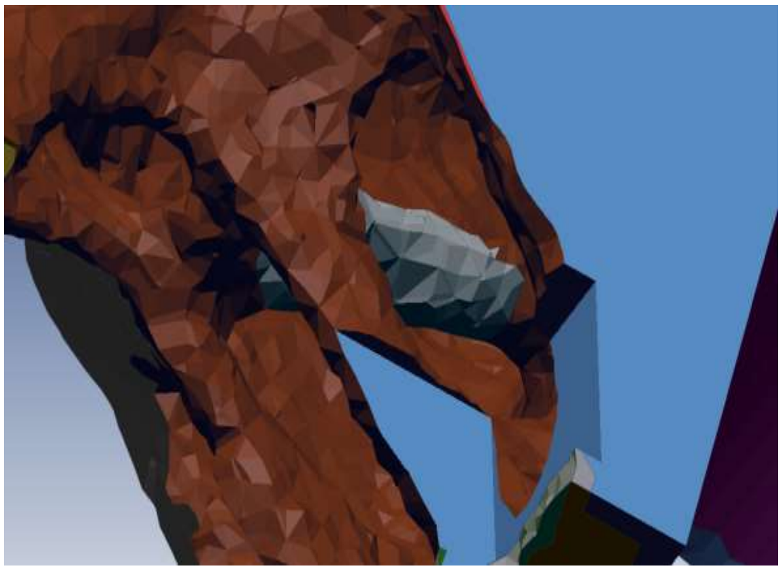


Figure 1-5 Noise and defects from the Topology Optimization results

1.2 Additive Manufacturing

Additive Manufacturing technology develops 3D objects by adding material layer upon layer. Each cross section has finite thickness which is derived from CAD geometry data. AM has certain advantages over conventional manufacturing processes. For example: AM eliminates the use of fixtures, process planning, tool selection and setup. This technique uses layer based approach where layers are made up of different thickness as required by the desired finish of the final product. The final product is an approximation of the original 3D CAD model and layer thickness parameter determines the accuracy as defined. For better accuracy, the layer thickness has to be minimized. The different type of AM processes are segregated by the degree of accuracy, final post processing time, overall printing time, Material etc.

Additive Manufacturing process involves several steps. It starts from 3D CAD modeling software where 3D model represents the final geometry to be manufactured.

STL file is widely used in AM process. Before transferring output to AM machines, 3D Solid or surface output is converted into STL file. Next, the machine is setup and certain parameters like layer thickness, material, and support structure are set. AM Machines build the object automatically. In the final step some cleaning of support structure is required before putting it into the use.

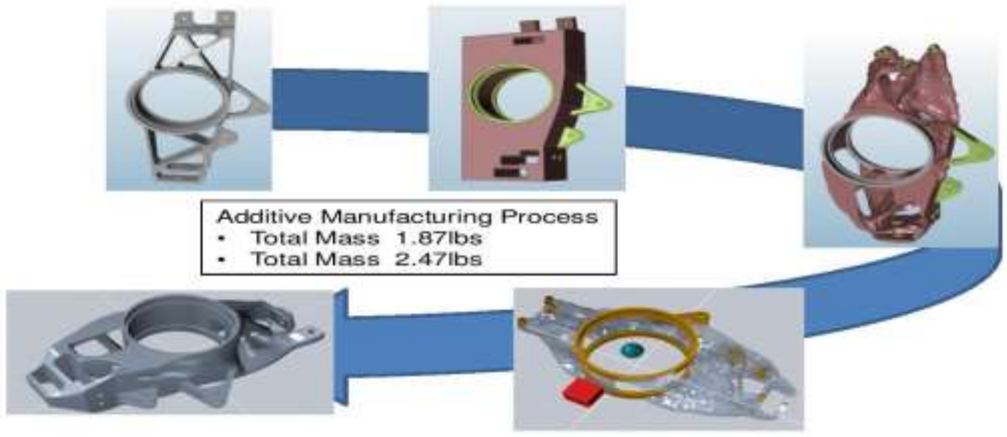


Figure 1-6 Topology Optimization Results for the Additive Manufacturing Process [14]

Additive manufacturing technique examples are as given below:

- SLA

  SLA (stereo lithography) models utilize laser technology to cure layer-upon-layer of photopolymer resin due to its high accuracy and advanced surface finish makes it the preferred option for designers.

- FDM

  In Fused deposition modeling, 3D CAD files slice and build a path to extrude thermoplastic and supporting material. After that, thermoplastic is heated to a semi-liquid state by 3D printer and deposits it in ultra-fine beads and removable material where support is needed. The user removes support material by dissolving in water or breaking in order to use final part.

- MJM

  Multi-Jet Modeling is similar to an inkjet printer. MJM uses hundreds of small jets to apply a layer by layer of thermo polymer material.

- 3DP

  3D printing process can be achieved in 3 sub steps: Modeling, Printing and Finishing.

  <u>Modeling</u>: 3D objects can be modeled by any 3D CAD Software.

  <u>Printing</u>: To print 3D CAD model, it needs to be solved in stereo lithography file format. STL files from CAD software have errors like facet intersection, noise shells and holes. To overcome these errors, STL files need to be processed by slicer software that converts them into printable thin layered format for 3D printer.

  <u>Finishing</u>: Some 3D printable objects require material removal, additional support and smoothness to get better performance.

- SLS

  Selective laser sintering technique uses high power laser to fuse small particles of plastic, metal, ceramic or glass powder into a mass of a desired three dimensional shape. [8]

1.3 Motivation

The overall aim of this research work is to automate the process of smoothing the noisy surfaces of topology optimization result and create a link that can be useful directly in additive manufacturing without significant efforts from the CAD designer. To achieve this overall objective, this process can be divided into several steps.

- After Topology Optimization results, cross-section and point cloud data are obtained.
- Contours are generated using cross section and point cloud data.
- The next step is the definition of the curve and fitting of that into nubs to generate surface.
- The final step involves intersection of surfaces at joints.

This research aids in contour generation and slicing of 3D complex geometry which is the most important step to achieve overall objective. STL files have errors like inconsistency in normal vectors, holes and over laps. Existing typical Slicing algorithm generally used to repair STL files and to generate 2D closed errorless contours. If CAD geometry is complex and it is sliced at an angle by arbitrary plane, typical slicing algorithm to develop closed contours is not applicable. In this research work, STL file has been sliced by an arbitrary plane at horizontal and at some angle with respect to z height. Generation of closed contours for 3D complex meshed Iso-surface using Tail to Head distance algorithm and Separation of contours has been proposed.

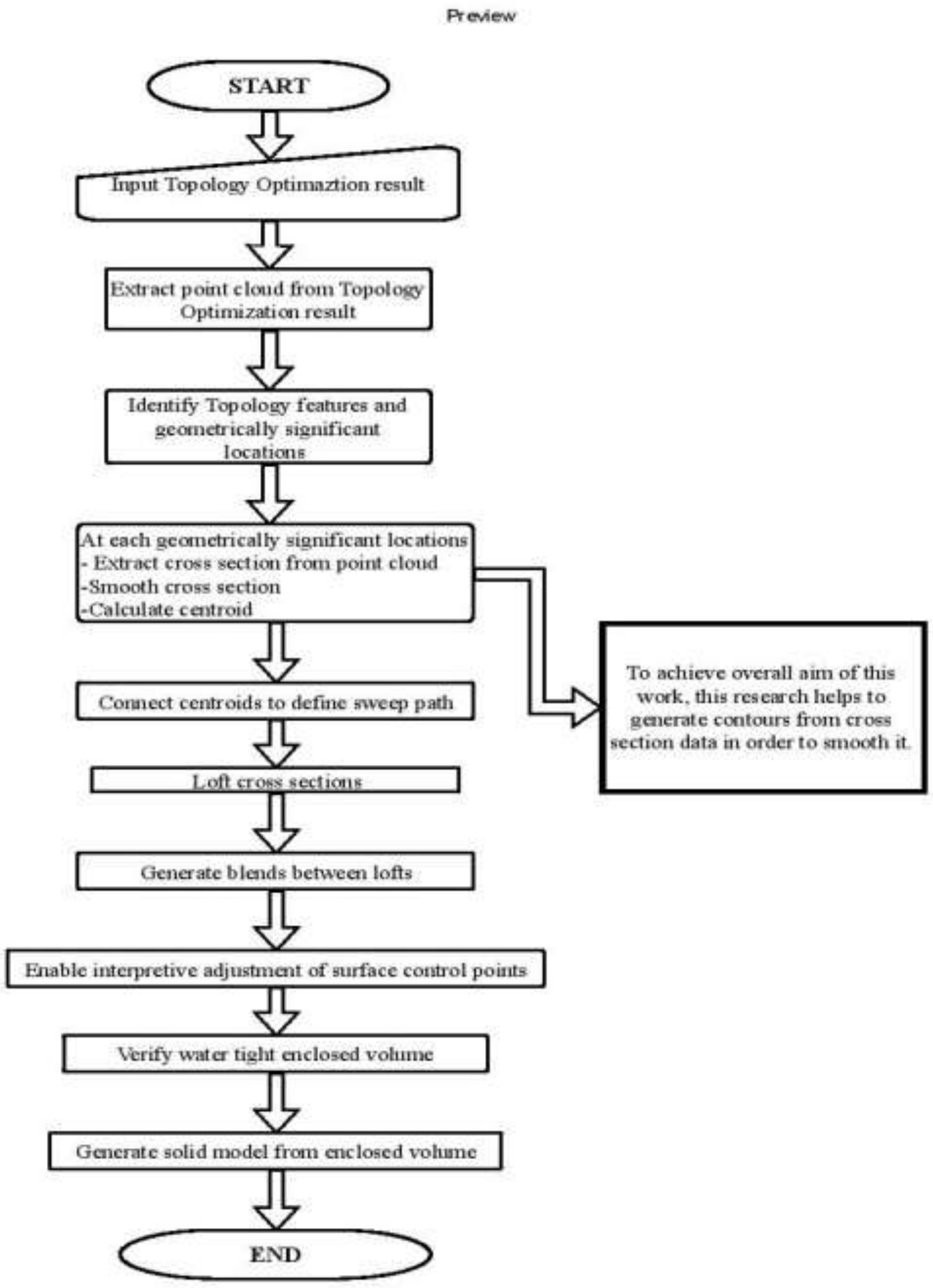

Preview
START
Input Topology Optimaztion result
Extract point cloud from Topology Optimization result
Identify Topology features and geometrically significant locations
At each geometrically significant locations
- Extract cross section from point cloud
-Smooth cross section
-Calculate centroid
To achieve overall aim of this work, this research helps to generate contours from cross section data in order to smooth it.
Connect centroids to define sweep path
Loft cross sections
Generate blends between lofts
Enable interpretive adjustment of surface control points
Verify water tight enclosed volume
Generate solid model from enclosed volume
END

Figure 1-7 Overall aim of the Research Work flow chart

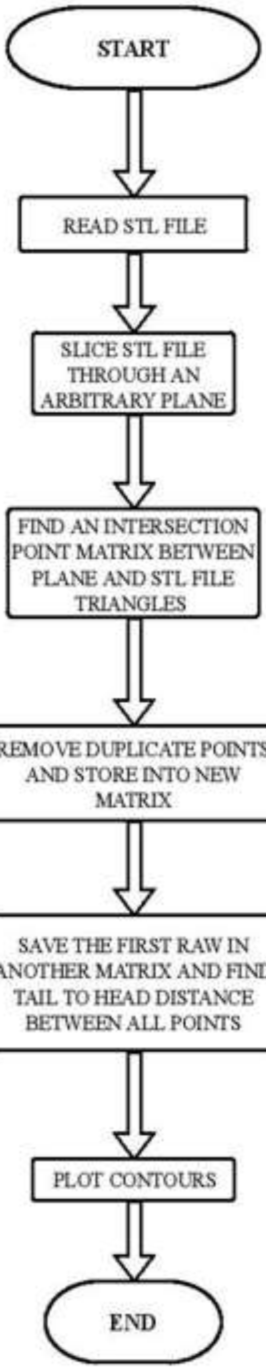

START
READ STL FILE
SLICE STL FILE THROUGH AN ARBITRARY PLANE
FIND AN INTERSECTION POINT MATRIX BETWEEN PLANE AND STL FILE TRIANGLES
REMOVE DUPLICATE POINTS AND STORE INTO NEW MATRIX
SAVE THE FIRST RAW IN ANOTHER MATRIX AND FIND TAIL TO HEAD DISTANCE BETWEEN ALL POINTS
PLOT CONTOURS
END

Figure 1-8 Thesis work flow chart

Chapter 2

Theoretical Background

In this section, Feature Recognition method, Stereo Lithography file and typical Slicing Algorithm are discussed in detail. As mentioned in chapter 1, STL file has a large impact in this research work's approach. Format of STL file and its significance are given in this chapter.

2.1 Feature Recognition Method

Feature Recognition method is used to detect "feature" which has a wide range of implications in different engineering areas. In CAD geometry, this method provides information of holes, pocket, rim, loft, sweep, swept cut etc. and plays an important role in the manufacturing. Feature Recognition is an active research area in solid modeling.

This method gives a platform to detect important features from the CAD geometry to assist in manufacturing process. Most Feature Recognition algorithm uses existing entity or pattern which however is not accurate while working with intersecting features.

As discussed in previous chapter 1, STL file needs to be repaired due to errors such as holes overlaps etc. to create a printable format for 3D printers or Additive manufacturing. Feature Recognition method is useful to detect void and solid part, which helps in repairing STL files. The main backdrop of this method is that it needs to be initiated by the user and depends on the user to provide inputs for time optimization. [9]

In the computer vision, this method is known as the Feature Detection. Feature Detection involves image processing which detects edges, corners, lines, circles etc. This method can be used to develop Feature Recognition in solid modeling as well. The user can assign different pixel values to distinguish between feature, solid and void. [10]

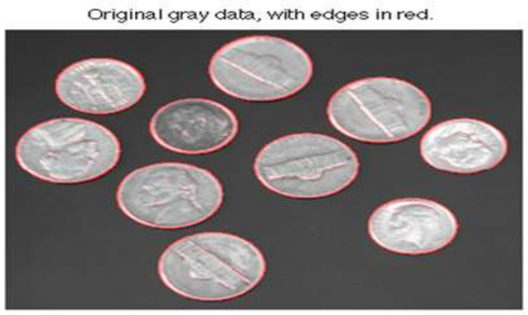


Figure 2-1 Feature Detection method using Image processing [10]

2.2 Stereo Lithography File

Stereo lithography file or an STL file is developed by 3D CAD software, which is in the form of triangles developed by the normal vector of facets and its vertices. The main advantage of STL file is its easiness to read and write in ASCII format. Though, STL files have some drawbacks too as they do not provide any information about material's color or texture. [11]

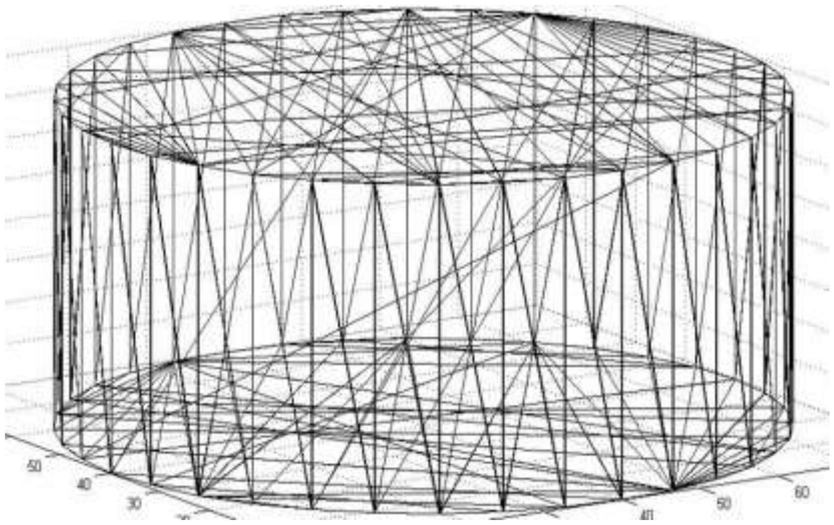


Figure 2-2 Cylinder STL file example

2.2.1 ASCII format of STL file

ASCII STL file format is given as below:

solid name

facet normal n_i n_j n_k

outer loop

vertex $v1_x$ $v1_y$ $v1_z$

vertex $v2_x$ $v2_y$ $v2_z$

vertex $v3_x$ $v3_y$ $v3_z$

endloop

endfacet

endsolid name

Above represents ASCII STL file format. ASCII file starts from “solid” to distinguish it from Binary file. In this file, “name” is an optional string but for “solid” to be initialized, we must leave a space. In this format facet represents the number of triangles. Normal represents the normal vector which is perpendicular to the plane of triangle and has three coordinates. Each coordinate of normal has three vertices. [12]

2.2.2 Binary format of STL file

BINARY STL file format is given as below:

UINT8[80] – Header

UINT32 – Number of triangles for each triangle

REAL32[3] – Normal vector

REAL32[3] – Vertex 1

REAL32[3] – Vertex 2

REAL32[3] – Vertex 3

End

The one thing that differentiates ASCII and BINARY is the start term “solid” and hence, a binary file should never start with solid. Next is a 4 byte unsigned integer that indicates the number of triangles facet in binary file. For each triangle there are three floating point integer for the normal and three for x, y and z coordinates of each vertex which make a total of 12 32 bit floating point number. [12]

2.3 Slicing Algorithm

In the typical Slicing algorithm, all facet data from the STL file are stored into the matrix form and the normal vectors of each facet are ignored to reduce the data size and memory.

[solid name

facet normal n_i n_j n_k

outer loop

vertex $v1_x$ $v1_y$ $v1_z$

vertex $v2_x$ $v2_y$ $v2_z$

vertex $v3_x$ $v3_y$ $v3_z$

endloop

endfacet

endsolid name]

$\Longrightarrow$

[facet Fn

vx1 vy1 vz1

vx2 vy2 vz2

vx3 vy3 vz3

.

.

facet Fm

vx1 vy1 vz1

vx2 vy2 vz2

vx3 vy3 vz3]

This method uses the following equation (3) to find an intersection between the line segment of facets and arbitrary horizontal plane.

Slope equation:

$$\frac{x - xa}{xb - xa} = \frac{y - ya}{yb - ya} = \frac{z - za}{zb - za} \tag{3}$$

In this equation xa, ya, za and xb, yb, zb are points of end facets. For an arbitrary plane, the height Z must satisfy the following equation (4). Here, Z1 and Z2 are the height of facet vertex data.

Condition equation:

$$Z1 \leq Z \leq Z2 \tag{4}$$

According to the slope equation, x, y and z are intersection points between the triangle facet and the cutting plane. Here, the height of a plane, 'Z 'is known. Thus, intersection points are calculated and stored into the line matrix for contour generation.

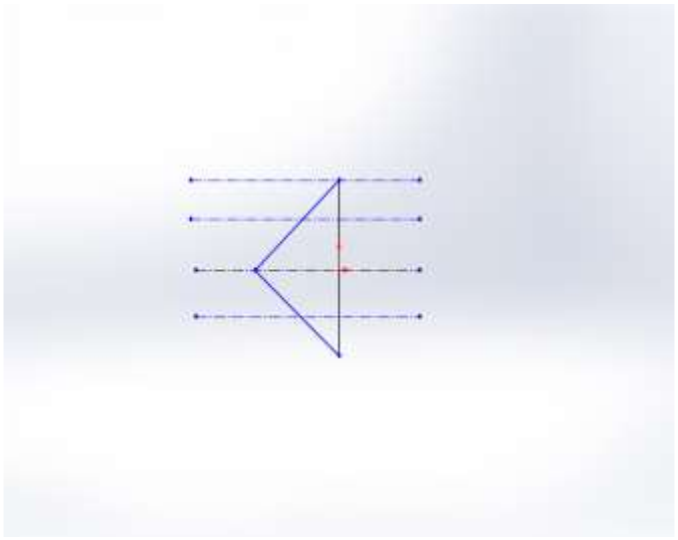
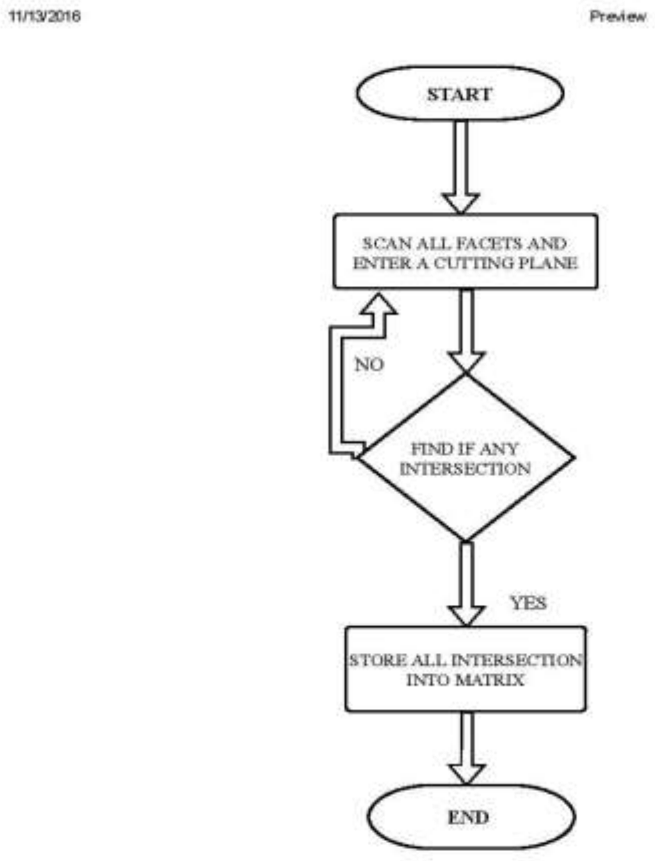


Figure 2-3 Intersection Cases between Facets and Cutting plane

The above figure 2-3 shows that the triangle‘s top most case does not satisfy equation (4), therefore, condition is not considered for contour generation. The remaining three line cases meet the requirement of equation (4) and are considered to find an intersection to generate contours.

However, the typical Slicing algorithm has some drawbacks because it cannot be applied to generate contours for 3D complex geometry and do not provide any information when STL file is sliced by an arbitrary plane at some angle. [13]



11/13/2016
Preview
START
SCAN ALL FACETS AND ENTER A CUTTING PLANE
NO
FIND IF ANY INTERSECTION
YES
STORE ALL INTERSECTION INTO MATRIX
END

Figure 2-4 Slicing algorithm flow-chart

Chapter 3

Methodology

In this section, the method to find intersection points between triangular facet and arbitrary plane are discussed in detail along with Tail to Head distance Algorithm and Separation of Contours Algorithm.

3.1 Method to find intersection points between STL file facet and arbitrary plane

As discussed in chapter 2, Existing typical Slicing algorithm has some limitations and is not applicable for complex CAD geometry at inclined arbitrary plane. In this chapter new methodology has been proposed to find intersection points of the line of each facet which is sliced by an arbitrary plane at horizontal or inclined plane. The key to understand this method is to understand line and plane equations in detail.

To solve the general plane equation, we need at least one point on the plane and normal vector. If normal vector is unknown, we must have three points to find the normal vector of a plane.

According to equation (5) and (6), if we have normal vector and one point on the plane, we can get the value of “d” and the general plane equation. STL file has triangle format and when it is sliced by an arbitrary plane; its facet line is intersecting with the plane. In 2D we can easily find intersection points through the slope equation but for 3D line we cannot use the same method.

Plane equation:

$$ax + by + cz = d \tag{5}$$

Normal vector equation:

$$n(a,b,c) = (p1x - p2x)i + (p1y - p2y)j + (p1z - p2z)k \tag{6}$$

Line equation:

$$X = x1 + t(x2 - x1) \tag{7}$$

$$Y = y1 + t(y2 - y1) \tag{8}$$

$$Z = z1 + t(z2 - z1) \tag{9}$$

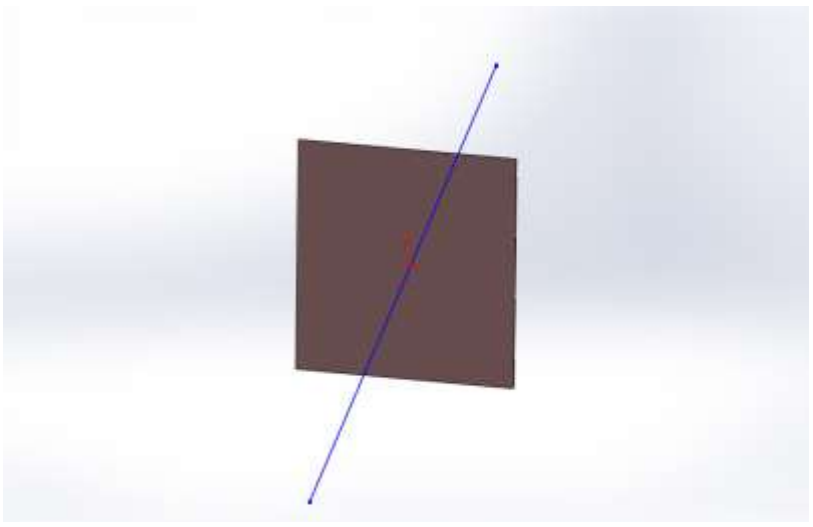


Figure 3-1 Plane and Line Intersection

The general plane equation and line's parametric equation with t variable is known. To find line's parametric equation we need to use (x1, y1, z1) and (x2, y2, z2) as the end points of facet.

Intersection points must satisfy both equation for plane and line. Thus, to find intersection points, Put equation (7), (8) and (9) into equation (5) which gives the value of "t".

By using the value of "t", we get intersection points X, Y and Z that are stored into the matrix form.

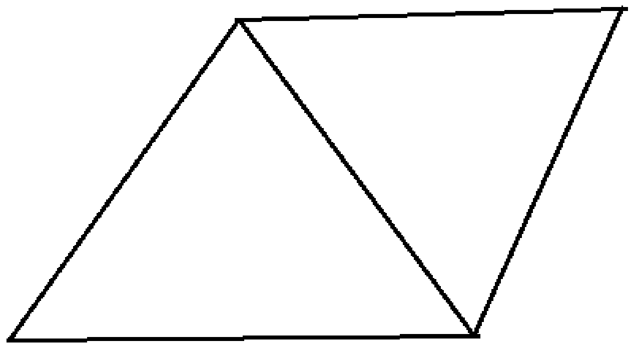


Figure 3-2 Adjacent Triangles in an STL File Format

Figure 3-2 shows a common edge shared by two triangles. There are "n" numbers of triangles which share a common adjacent line in STL file. When common edge triangles are sliced by the plane, a number of similar coordinates are obtained that are in the same or opposite direction. These coordinates need to be removed to generate closed and errorless contours.

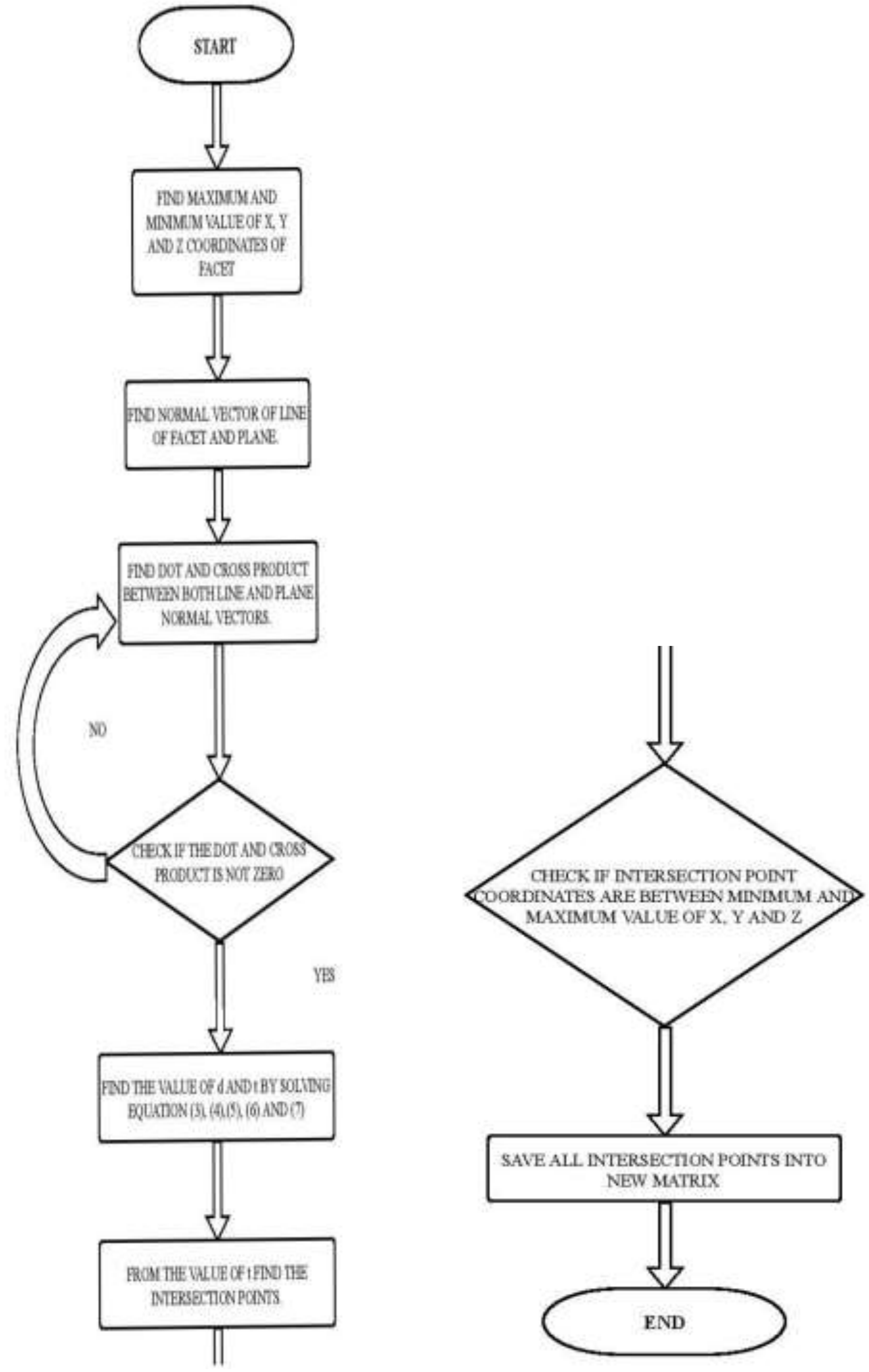

START
FIND MAXIMUM AND MINIMUM VALUE OF X, Y AND Z COORDINATES OF FACET
FIND NORMAL VECTOR OF LINE OF FACET AND PLANE.
FIND DOT AND CROSS PRODUCT BETWEEN BOTH LINE AND PLANE NORMAL VECTORS.
NO
CHECK IF THE DOT AND CROSS PRODUCT IS NOT ZERO
YES
FIND THE VALUE OF d AND t BY SOLVING EQUATION (3), (4),(5), (6) AND (7)
FROM THE VALUE OF t FIND THE INTERSECTION POINTS.
CHECK IF INTERSECTION POINT COORDINATES ARE BETWEEN MINIMUM AND MAXIMUM VALUE OF X, Y AND Z
SAVE ALL INTERSECTION POINTS INTO NEW MATRIX
END

Figure 3-3 Step Flow chart to find Intersection points

Remove duplicate points:

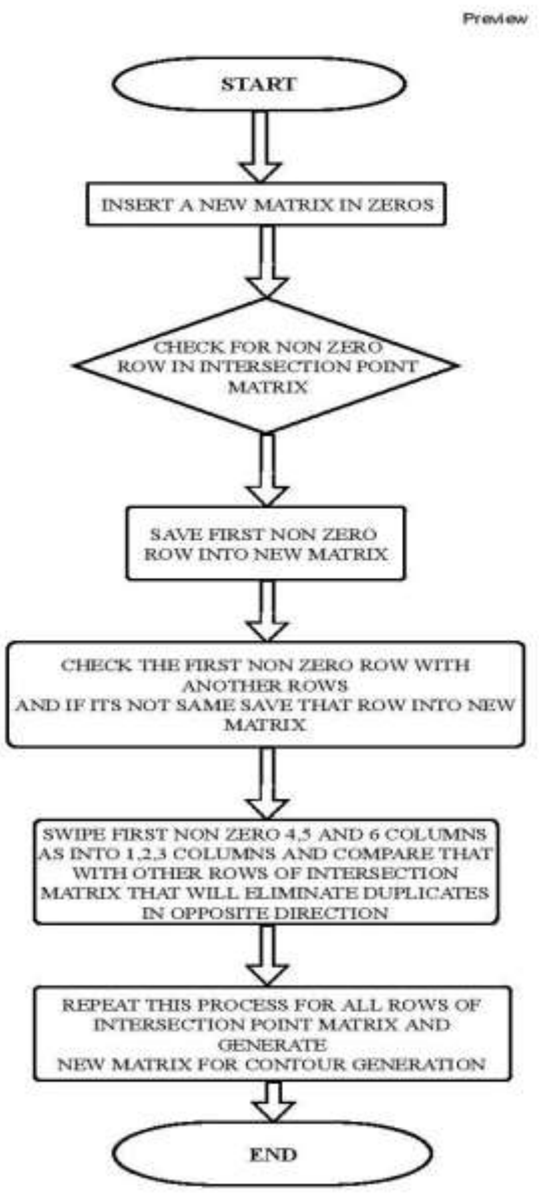

Preview
START
INSERT A NEW MATRIX IN ZEROS
CHECK FOR NON ZERO ROW IN INTERSECTION POINT MATRIX
SAVE FIRST NON ZERO ROW INTO NEW MATRIX
CHECK THE FIRST NON ZERO ROW WITH ANOTHER ROWS AND IF ITS NOT SAME SAVE THAT ROW INTO NEW MATRIX
SWIPE FIRST NON ZERO 4,5 AND 6 COLUMNS AS INTO 1,2,3 COLUMNS AND COMPARE THAT WITH OTHER ROWS OF INTERSECTION MATRIX THAT WILL ELIMINATE DUPLICATES IN OPPOSITE DIRECTION
REPEAT THIS PROCESS FOR ALL ROWS OF INTERSECTION POINT MATRIX AND GENERATE NEW MATRIX FOR CONTOUR GENERATION
END

Figure 3-4 Step Flow chart to remove duplicate points

3.2 Tail to Head Distance algorithm

To develop errorless closed contours, we need to understand Tail to Head Distance algorithm in detail. All intersection point data are stored randomly in the result matrix which should be sort out in order to remove inconsistency.

After getting intersection points and removing duplicates, the first non-zero row from the intersection point matrix is stored into the new contour point matrix. The intersection point matrix has six columns, out of which columns one to three and columns four to six represent the head and tail of intersection points respectively. From the Tail of the first non-zero row, all Head and Tail distance are calculated using the following formula.

Distance Formula:

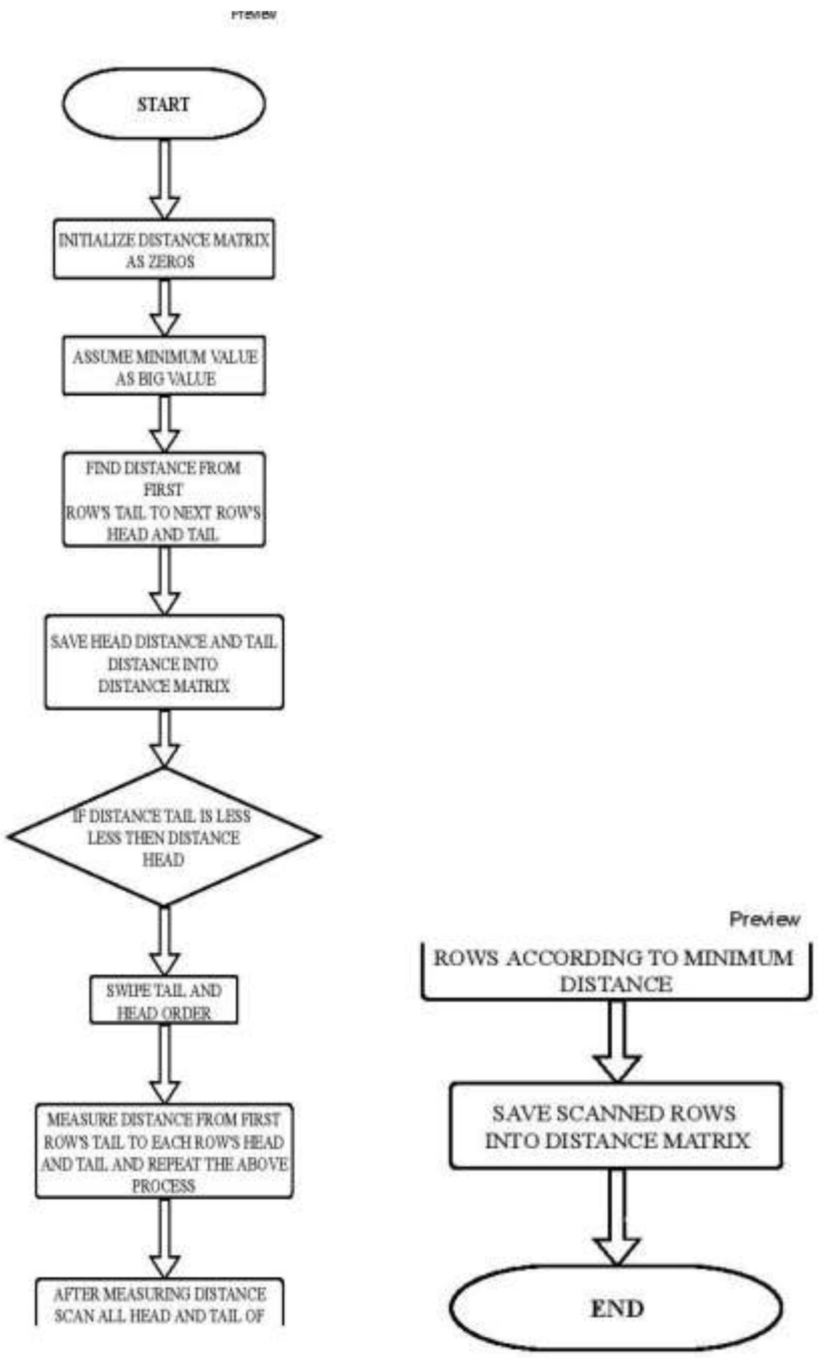
$$D = \sqrt{(x2 - x1)^2 + (y2 - y1)^2} \tag{10}$$

$$tail < head \tag{11}$$

As per equation (11), if Tail distance is less than the Head distance, intersection data points of Head and Tail should be interchanged with each other. After comparing distances of all rows with respect to the Tail of the first non-zero row, the row with minimum distance should be saved as consecutive row.

The above process is repeated for the newly saved row’s Tail points to other rows below it. Thus, by following the same process we will get Tail to Head distance analysis result matrix.

3.2.1 Steps of tail to head distance algorithm



Preview
START
INITIALIZE DISTANCE MATRIX AS ZEROS
ASSUME MINIMUM VALUE AS BIG VALUE
FIND DISTANCE FROM FIRST ROW'S TAIL TO NEXT ROW'S HEAD AND TAIL
SAVE HEAD DISTANCE AND TAIL DISTANCE INTO DISTANCE MATRIX
IF DISTANCE TAIL IS LESS LESS THEN DISTANCE HEAD
SWIPE TAIL AND HEAD ORDER
MEASURE DISTANCE FROM FIRST ROW'S TAIL TO EACH ROW'S HEAD AND TAIL AND REPEAT THE ABOVE PROCESS
AFTER MEASURING DISTANCE SCAN ALL HEAD AND TAIL OF
Preview
ROWS ACCORDING TO MINIMUM DISTANCE
SAVE SCANNED ROWS INTO DISTANCE MATRIX
END

Figure 3-5 Step Flow chart of Tail to Head Distance algorithm

- In complex 3D geometry, when STL file has some void, contours need to be separated.
- When the first row of the result of Tail to Head Distance algorithm is repeating, rows contained in between are stored in the new matrix that represents as a new page in order to separate contours.
- This method also performs Separation of Contours based on user's need.
- User can input X, Y and Z coordinates and by using distance analysis between those coordinates and contour points, we can separate contours and plot them.

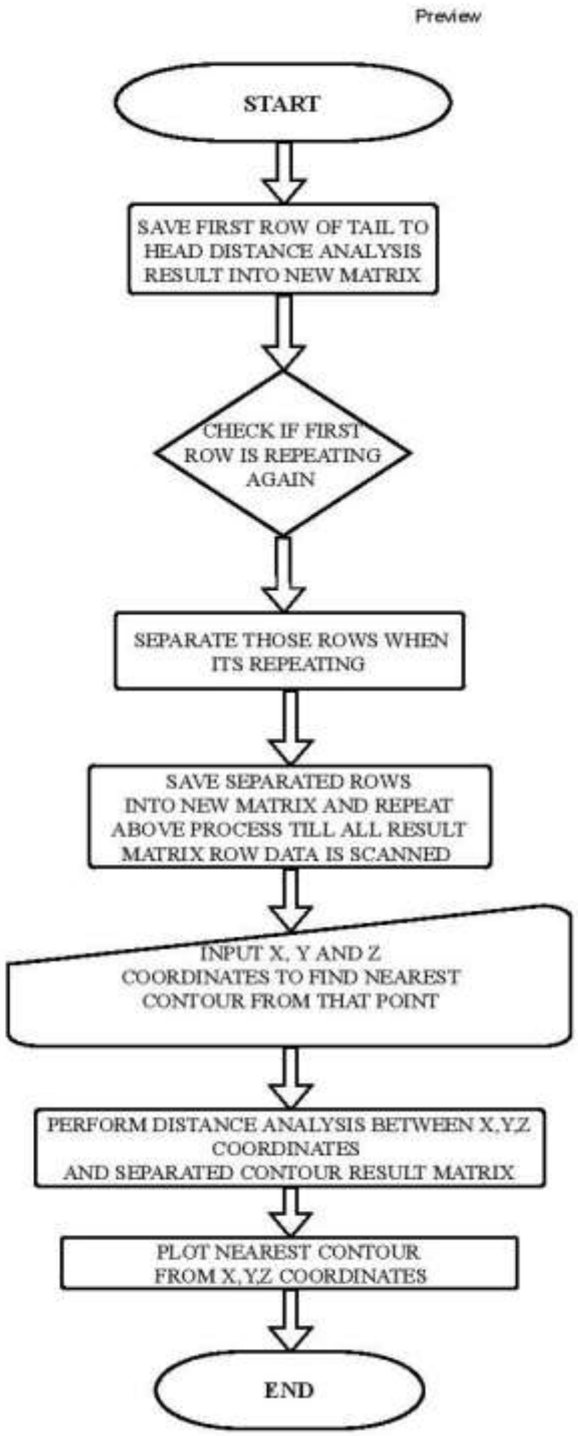

Preview
START
SAVE FIRST ROW OF TAIL TO HEAD DISTANCE ANALYSIS RESULT INTO NEW MATRIX
CHECK IF FIRST ROW IS REPEATING AGAIN
SEPARATE THOSE ROWS WHEN ITS REPEATING
SAVE SEPARATED ROWS INTO NEW MATRIX AND REPEAT ABOVE PROCESS TILL ALL RESULT MATRIX ROW DATA IS SCANNED
INPUT X, Y AND Z COORDINATES TO FIND NEAREST CONTOUR FROM THAT POINT
PERFORM DISTANCE ANALYSIS BETWEEN X,Y,Z COORDINATES AND SEPARATED CONTOUR RESULT MATRIX
PLOT NEAREST CONTOUR FROM X,Y,Z COORDINATES
END

Figure 3-6 Step Flow chart of Separation of Contours

Chapter 4

Results and Discussions

In this chapter, Implementation of method to find an Intersection between facet and given arbitrary plane, Tail to Head Distance algorithm and Separation of Contours are discussed in detail for four different CAD geometry cases.

4.1 Cylinder case

To implement the methodology which is discussed in chapter 3, a cylinder's case is tested. Geometry of the cylinder is created in SOLIDWORKS and exported in ASCII STL file format in MATLAB.

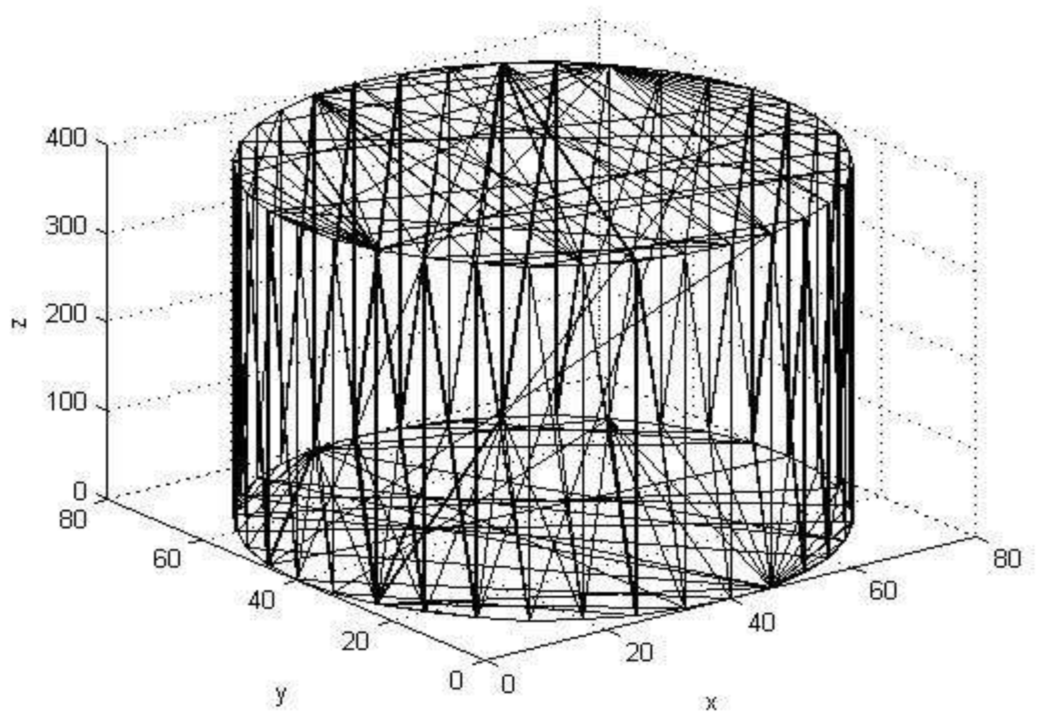


Figure 4-1 Cylinder STL file

4.1.1 Cylinder test case for the horizontal plane

Following figure 4-2 shows an STL file of the cylinder that is sliced by the horizontal plane. Table 1 displays the plane coordinates and the number of facet data which are used to develop a closed contour. Contour generation for the Cylinder case is achieved successfully without any errors.

Table 1: Parameters of the horizontal plane and STL file for the cylinder

Cylinder case	Horizontal plane
Number of facets	140
Plane coordinates p1	[0 0 200]
Plane coordinates p2	[80 0 200]
Plane coordinates p3	[80 80 200]

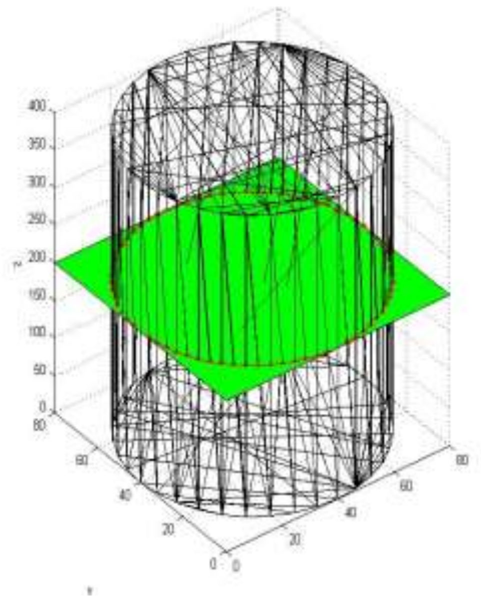


Figure 4-2 Contour generation for the Cylinder at the horizontal plane

4.1.2 Cylinder test case for an inclined plane

Table 2 illustrates the parameters to generate the plane at an inclined angle for the Cylinder STL case. Figure 4-3 displays successful implementation of the proposed method and contour generation. Figure 4-2 and 4-3 implies that the proposed method to find intersection points at any arbitrary plane is successfully obtained without any errors.

Table 2: Parameters of an inclined plane and STL file for the cylinder

Cylinder case	Inclined plane
Number of facets	140
Plane coordinates p1	[0 0 0]
Plane coordinates p2	[80 0 200]
Plane coordinates p3	[80 80 200]

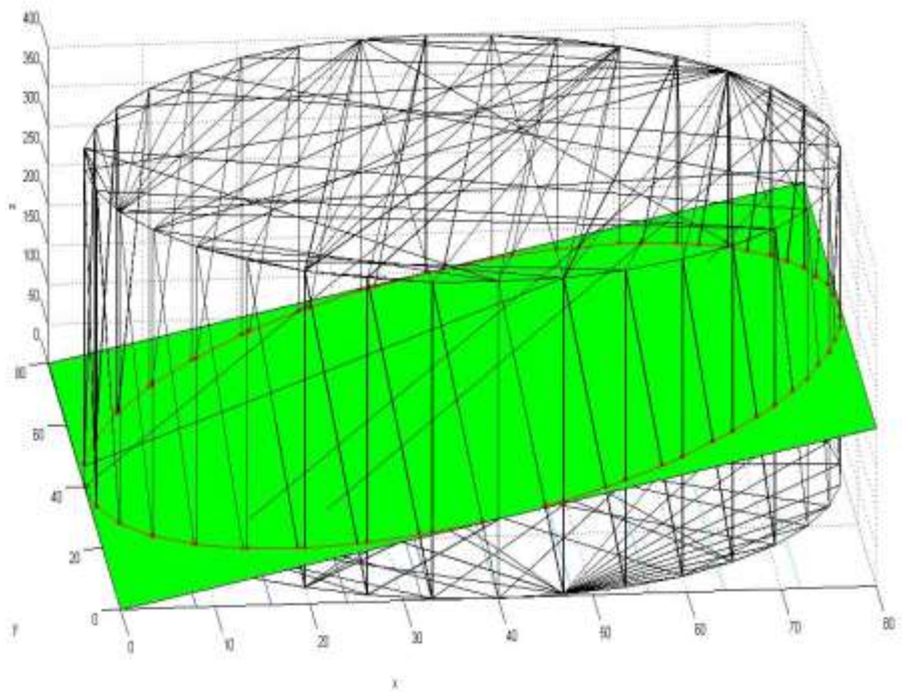


Figure 4-3 Contour generation for the cylinder at an inclined plane

4.2 Table with two legs case

. The STL geometry of the table with two legs case has been given in the following figure 4-4 which displays some noises in the STL file. When this geometry is sliced by the plane, it generates few random points.

There is a gap between two legs which gives random lines in the STL file that implies noise defects.

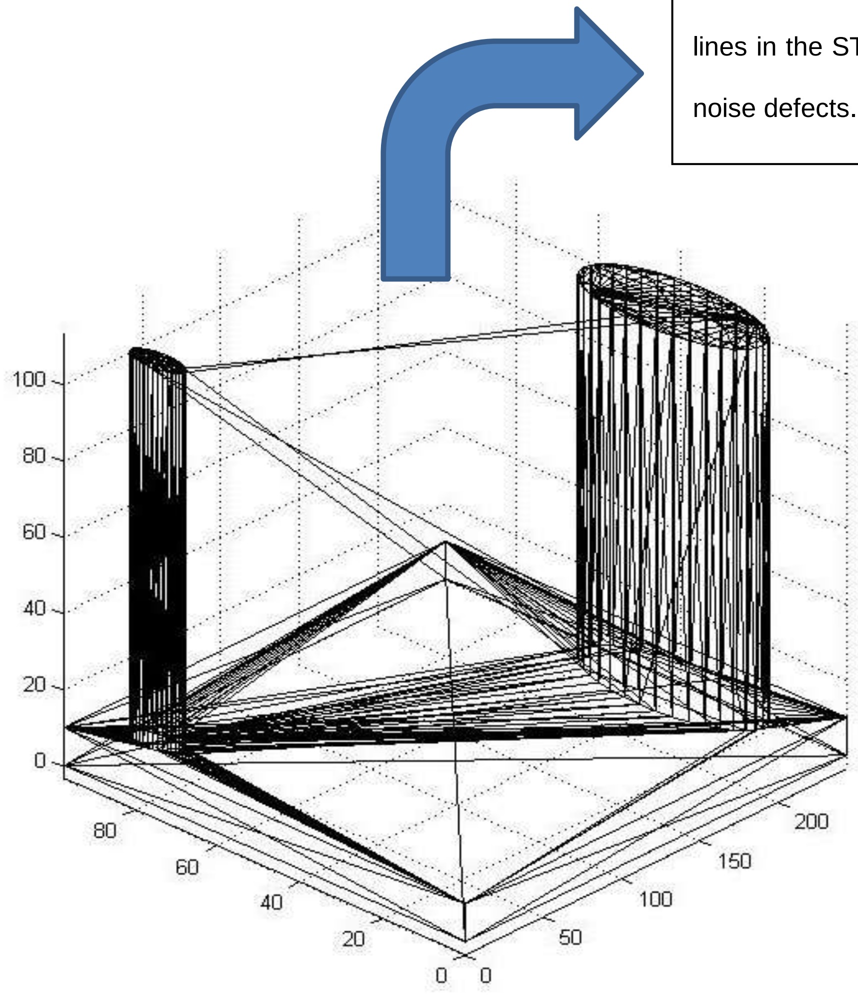


Figure 4-4 Table with two legs case STL file

4.2.1 Table with two legs case for the horizontal plane

Table 3 displays the plane coordinates and the number of facet data which are used to develop closed contours. Here, X Y and Z coordinates are used in order to separate contours. Figure 4-5 shows the red and yellow contours obtained, where the red contour is the nearest contour from X, Y and Z coordinate.

Table 3: Plane coordinates and X Y and Z coordinates which are used to separate and generate contours for the horizontal case

Table with two legs	Horizontal plane
Number of facets	300
Plane coordinates p1	[0 0 100]
Plane coordinates p2	[250 0 100]
Plane coordinates p2	[250 150 100]
X coordinate used to separate contours	0
Y coordinate used to separate contours	0
Z coordinate used to separate contours	100

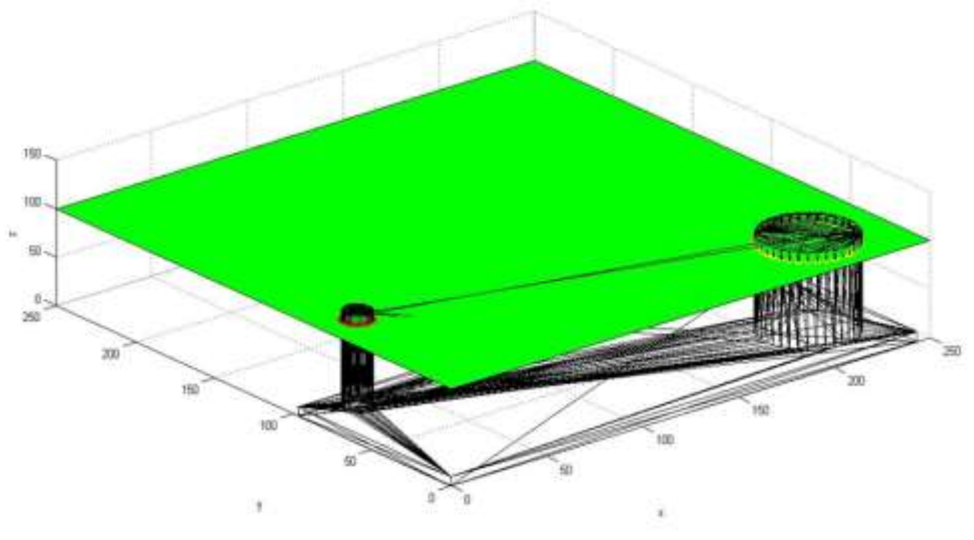


Figure 4-5 Contour generation and separation for the Table with two legs STL file for the horizontal plane

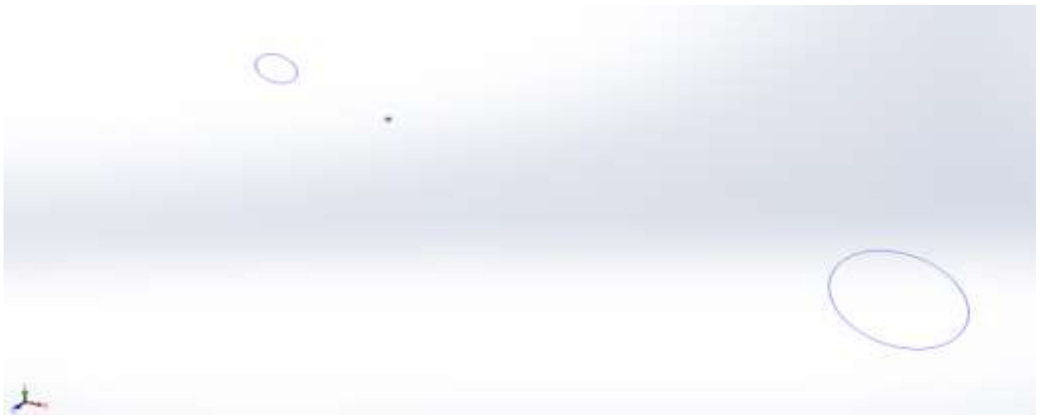


Figure 4-6 Contours exported from MATLAB and imported into SOLIDWORKS for the table with two legs case at the horizontal plane

For better visualization, Contours are imported into SOLIDWORKS from MATLAB that is given in figure 4-6.

4.2.2 Table with two legs case for an inclined plane

Figure 4-7 and 4-8 shows contour generation and separation for an inclined plane case. The obtained red contour is the nearest one from (x, y, z) = (100,150,200) coordinates. Parameters which are used to develop contour generation and separation are given in table 4.

Table 4: Plane coordinates and X Y and Z coordinates which are used to separate and generate contours for inclined plane case

Table with two legs	Inclined plane
Number of facets	300
Plane coordinates p1	[0 0 20]
Plane coordinates p2	[250 20 100]
Plane coordinates p2	[250 150 100]
X coordinate used to separate contours	100
Y coordinate used to separate contours	150
Z coordinate used to separate contours	200

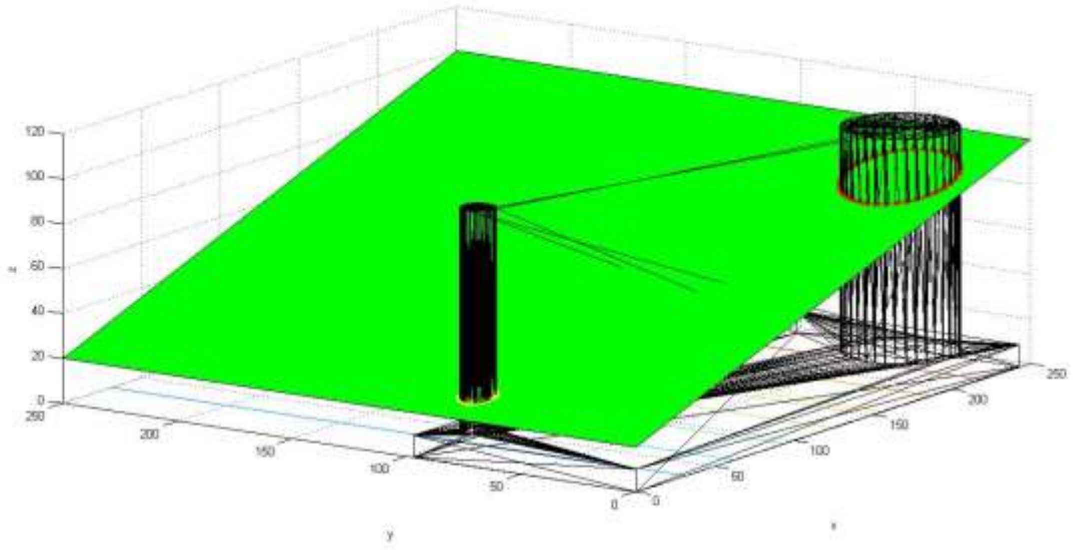


Figure 4-7 Contour generation and separation for the Table with two legs STL file for an inclined plane

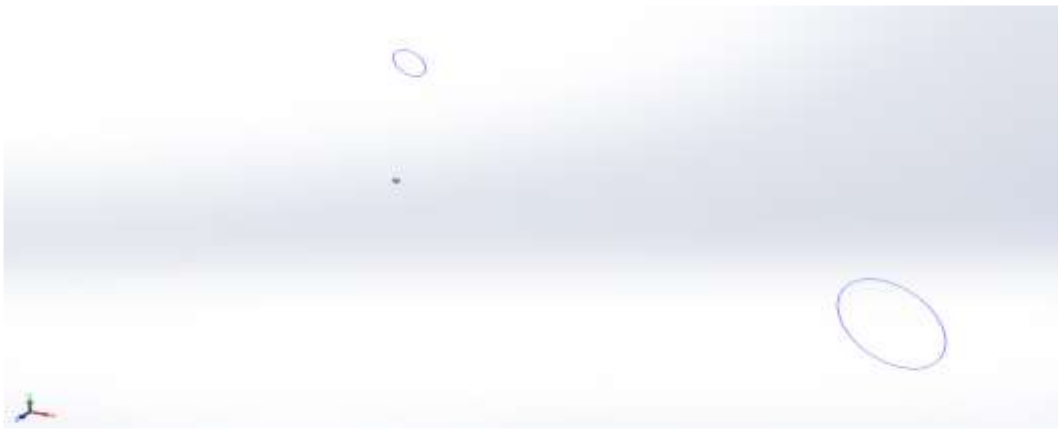


Figure 4-8 Contours exported from MATLAB and imported into SOLIDWORKS for the table with two legs at an inclined plane case

The proposed algorithm to find Separation of Contours is implemented for simple the table with two legs case that has two different dimensions in each leg. Figure 4-5 and 4-7 displays the ability of the Separation of Contours with different dimensions.

4.3 Missile fin case

After consideration of the table with two legs case, this method is applied to more complex STL files. Missile fin case is tested by using the proposed method at the horizontal plane and at an inclined plane. Here, Missile fin geometry is designed using NACA0012 airfoil and CATIA V5. Topology optimization is performed on missile fin using Solid thinking Inspire tool.

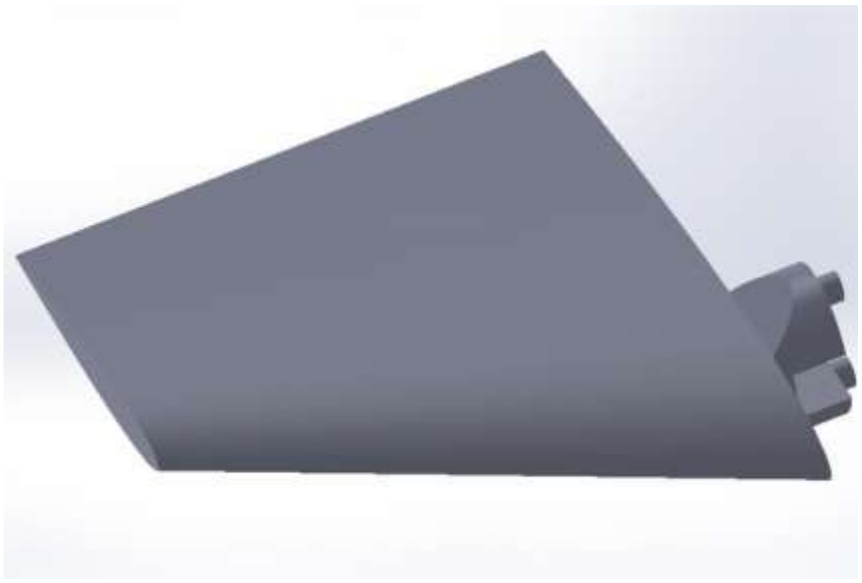


Figure 4-9 Missile Fin Geometry

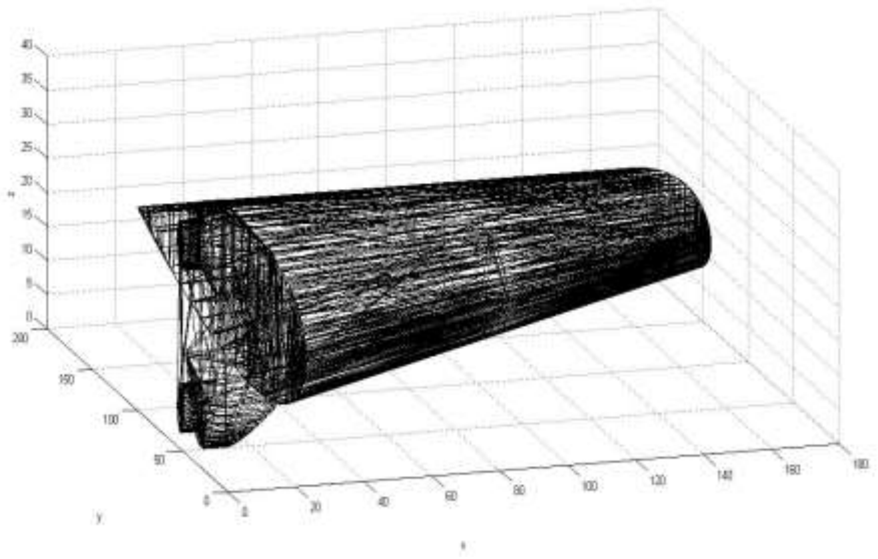


Figure 4-10 STL file of the Missile Fin

4.3.1 Missile fin case for the horizontal plane

Table 5 illustrates the parameters for the missile fin case at the horizontal plane. Figure 4-11 and 4-12 shows errorless closed contours generation.

Table 5: Parameters of the horizontal plane for missile fin and STL file

Missile fin case	Horizontal plane
Number of facets	1428
Plane coordinates p1	[0 0 15]
Plane coordinates p2	[180 0 15]
Plane coordinates p3	[180 200 15]

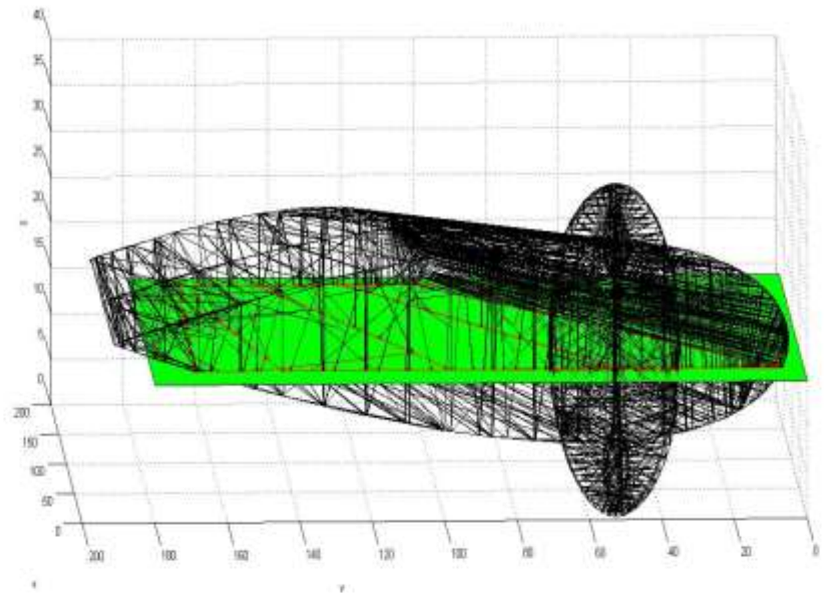


Figure 4-11 Result view 1 for the horizontal plane of the Missile Fin Case

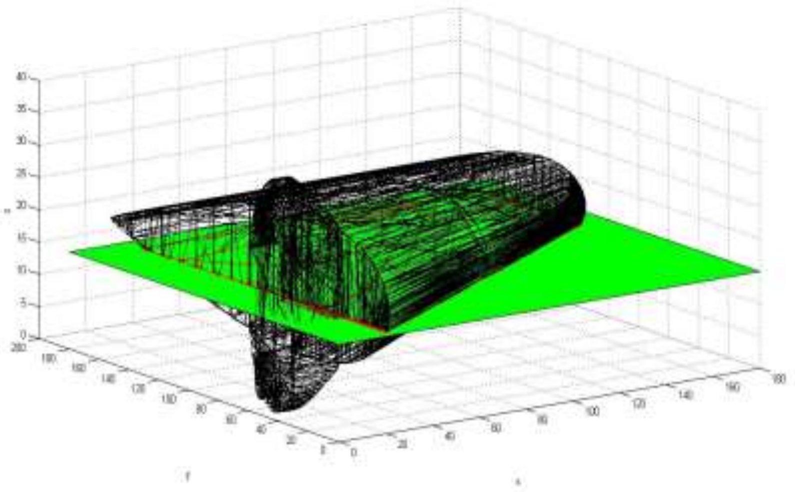


Figure 4-12 Result view 2 for the horizontal plane of the Missile Fin Case

4.3.2 Missile fin case for an inclined plane

In this section, the missile fin case at an inclined plane is analyzed. Table 6 displays two cases of X, Y and Z coordinates in order to understand separation of the contours. In this case, three different contours are generated. When user inputs values (X, Y, Z) = (0, 10, 20) for case A and (X, Y, Z) = (200, 150, 30) for Case B respectively, red contour is generated which is the nearest contour to those points. This is explained by figure 4-13 and 4-14.

Table 6: Plane coordinates and X Y and Z coordinates which are used to separate and generate contours for the missile fin at inclined plane

Missile fin case	Inclined plane
Number of facets	1428
Plane coordinates p1	[0 0 0]
Plane coordinates p2	[200 0 40]
Plane coordinates p2	[200 200 40]
X coordinate used to separate contours	for result a(0) and for result b (200)
Y coordinate used to separate contours	for result a(10) and for result b (150)
Z coordinate used to separate contours	for result a(20) and for result b (30)

The proposed method to find an intersection, Tail to Head Distance and Separation of Contours algorithm are tested extensively with the complex STL file of the missile fin case with 1428 number of facets. Figure 4-11, 4-12, 4-13 and 4-14 indicates successful achievement of the methodology which is discussed in chapter 3.

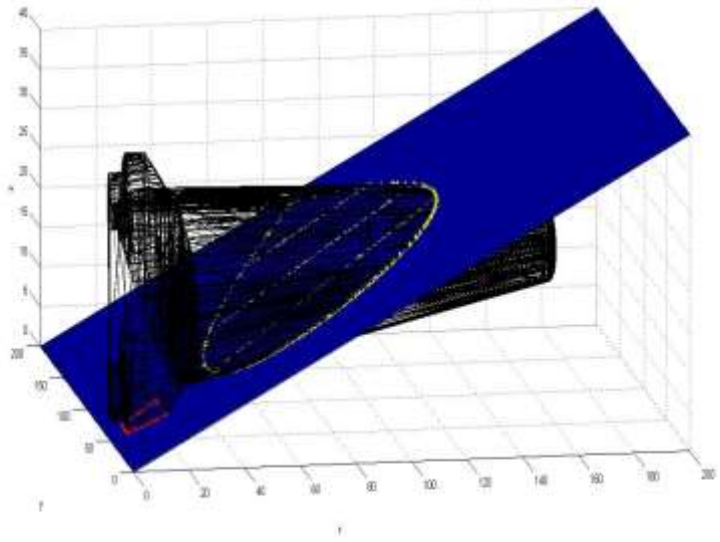


Figure 4-13 Result case A for an inclined plane of the Missile Fin Case

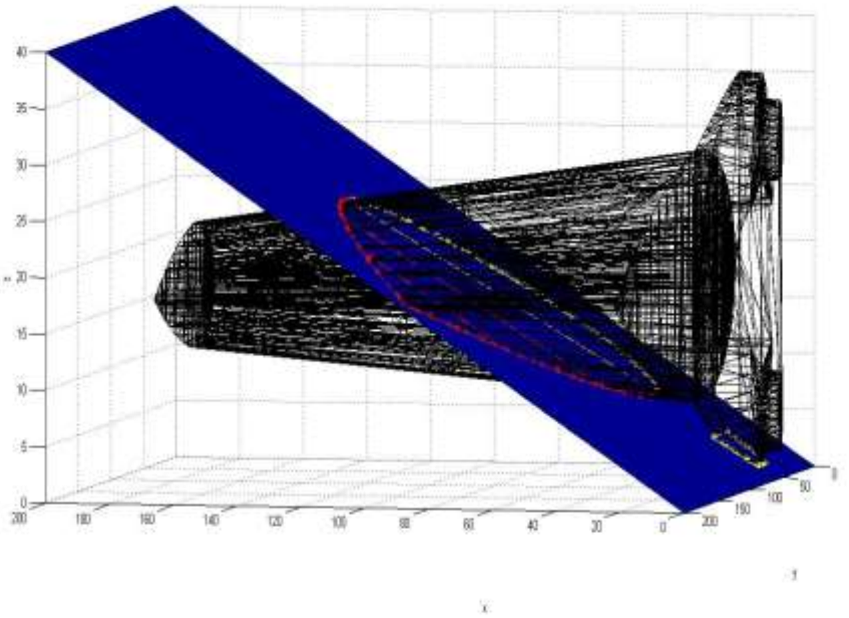


Figure 4-14 Result case B for an inclined plane of the Missile Fin Case

4.3.3 Missile fin topology optimization case for the horizontal plane

Figure 4-15 displays the missile fin surface, where the pressure is decreasing from trailing edge to leading edge in order to derive topology optimization result.

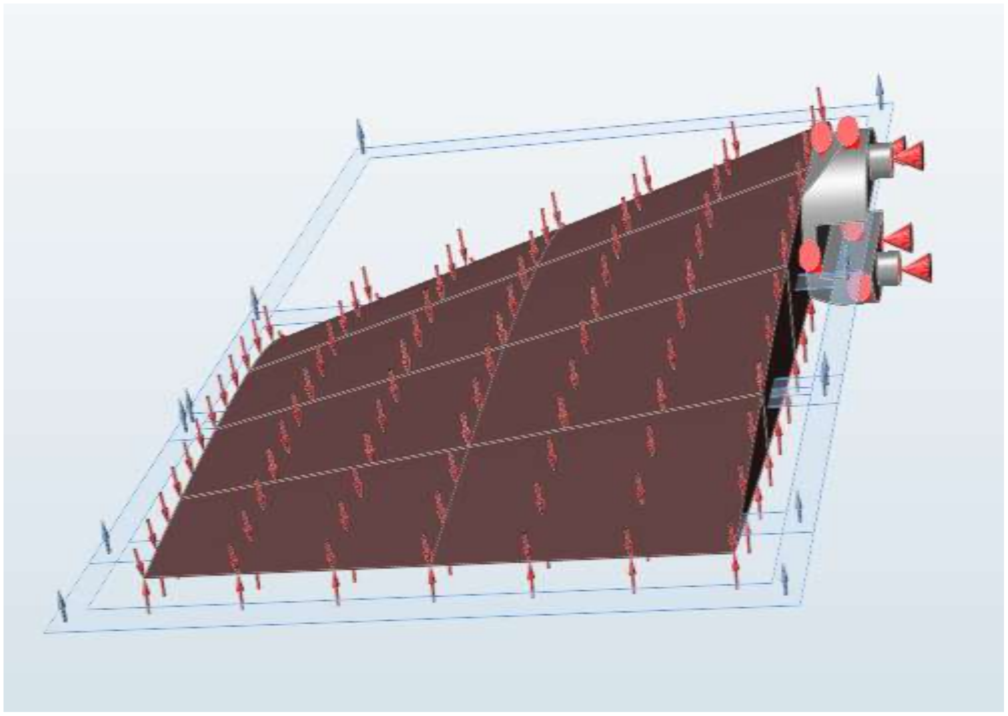


Figure 4-15 Pressure and structural load on the missile fin geometry using Inspire

Topology optimization results achieved by using Solid thinking Inspire tool are given in figure 4-16. In order to find maximum ability of the proposed methodology and algorithm, the missile fin topology optimization result is tested. For the missile fin topology result, the contour points are very close to each other and have very high précised values which is difficult to locate in MATLAB graphics. For example: x1= 0.00464 and x2= 0.00468. In order to understand topology result better, a lump of material is removed.

Figure 4-17 shows the material removal from the missile fin topology optimization result which has 7960 number of facets.

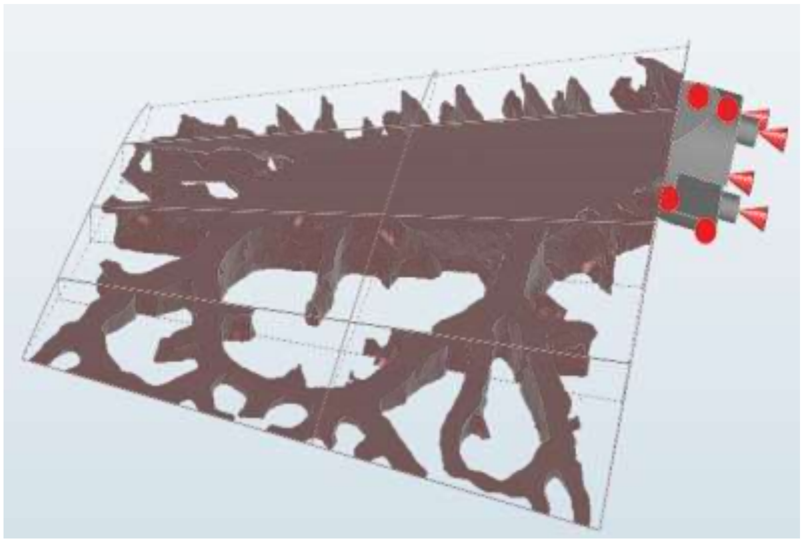


Figure 4-16 Topology optimization result of the Missile Fin

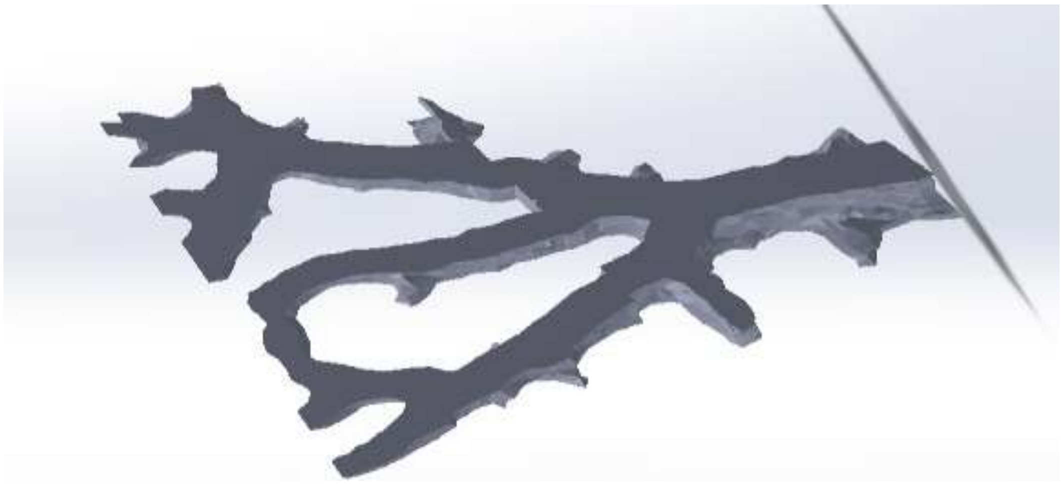


Figure 4-17 Material removal from Topology optimization result of the Missile Fin

Table 7 shows the plane coordinates and X, Y, Z coordinates in use. In this case, there are six total numbers of separated contours. Figure 4-18 and 4-19 shows the different views of achieved results. Tracing of contours is difficult with MATLAB graphics. Thus, Contours are imported in SOLIDWORKS from MATLAB which are shown in the figure 4-18. After careful inspection of figure 4-18, the output layer of the contours shows successful slicing and separation of contours for the missile fin topology optimization result.

Table 7: Plane coordinates and X Y and Z coordinates which are used to separate and generate contours for topology optimization result of the missile fin case

Missile fin case	Horizontal plane
Number of facets	7960
Plane coordinates p1	[-0.04 -0.2 0]
Plane coordinates p2	[0.14 -0.2 0]
Plane coordinates p2	[0.14 0.15 0]
X coordinate used to separate contours	0
y coordinate used to separate contours	0
z coordinate used to separate contours	0

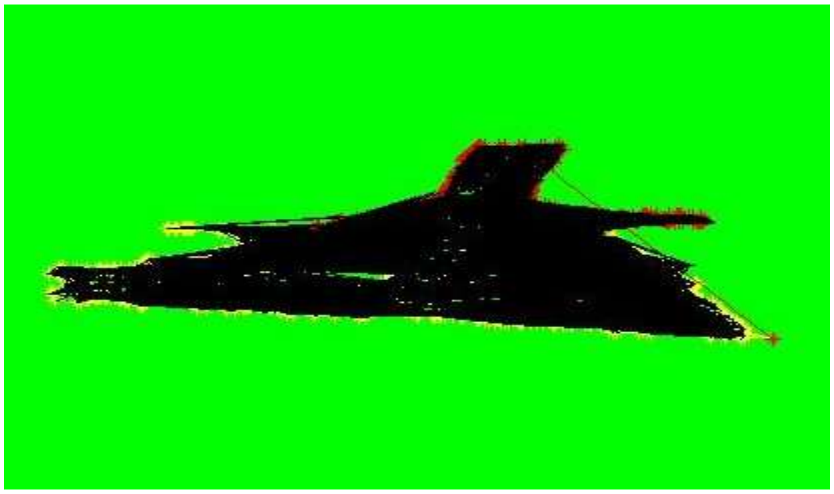


Figure 4-18 Result view 1 for the missile fin topology optimization result case at the horizontal plane

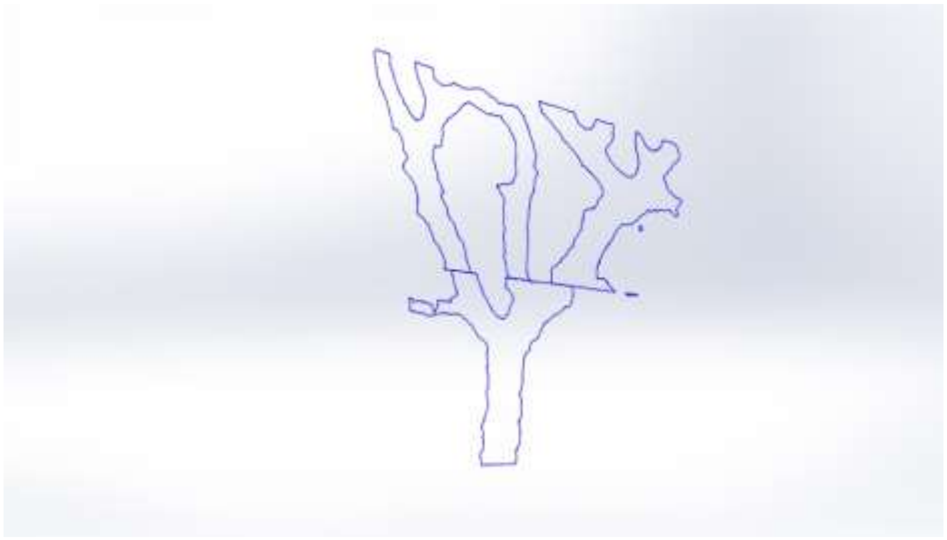


Figure 4-19 Contour importation of the missile fin topology optimization result case in SOLIDWORKS at the horizontal plane

4.4 FSAE RACE CAR Steering Knuckle Case

Figure 4-20 indicates the non-design region for the FSAE RACE CAR steering knuckle case. The non-design region should be constant after topology optimization. Figure 4-21 shows topology optimization results for the FSAE RACE CAR Steering knuckle that is tested in order to check stability of the proposed algorithm.

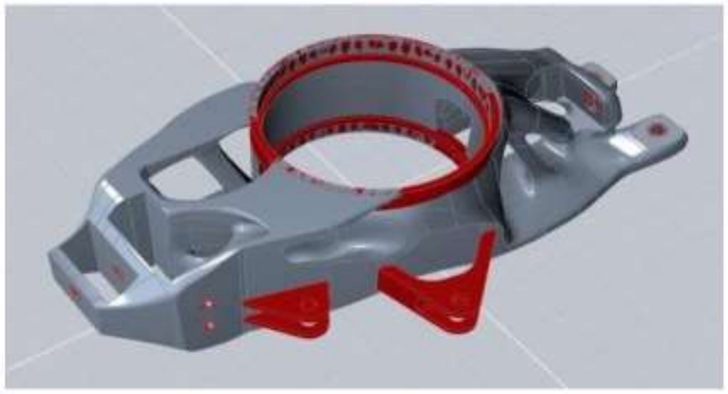


Figure 4-20 FSAE RACE CAR STEERING Knuckle non-design space [14]

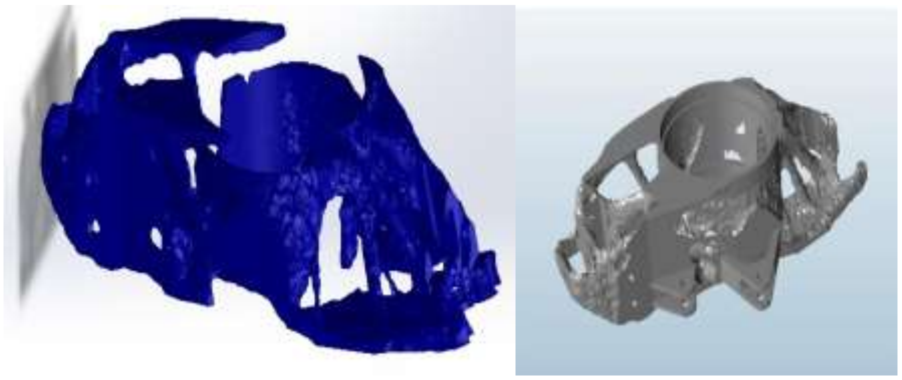


Figure 4-21 FSAE RACE CAR STEERING Knuckle topology optimization result [14]

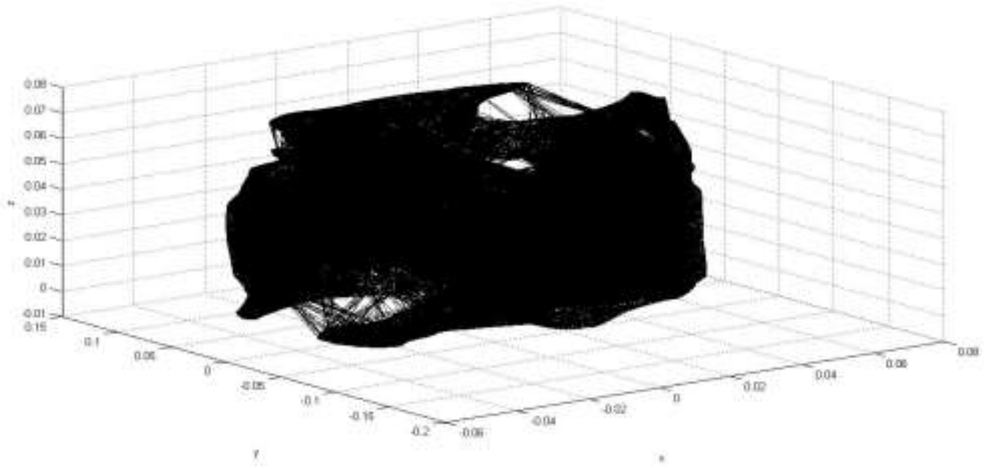


Figure 4-22 FSAE RACE CAR STEERING Knuckle STL file

The above figure is the representation of steering knuckle STL file in MATLAB. There are 93,998 facets in ASCII format of this STL file.

4.4.1 FSAE RACE CAR Steering knuckle case for the horizontal plane

In this case, table 8 shows the parameters of the plane and X, Y, Z coordinates. Figure 4-22 shows the contour generation for the FSAE RACE CAR Steering knuckle at the horizontal plane. It is visually very hard to locate contours in MATLAB. Figure 4-23 indicates successful contours importation in SOLIDWORKS.

Table 8: Plane coordinates and X Y and Z coordinates which are used to separate and generate contours for topology optimization result of the FSAE RACE CAR Steering knuckle case at the horizontal plane

steering knuckle case	horizontal plane
number of facets	93998

plane coordinates p1	[-0.06 -0.2 0.03]
plane coordinates p2	[0.08 -0.2 0.03]
plane coordinates p2	[0.08 0.15 0.03]
x coordinate used to separate contours	-0.02
y coordinate used to separate contours	0.10
z coordinate used to separate contours	0.02

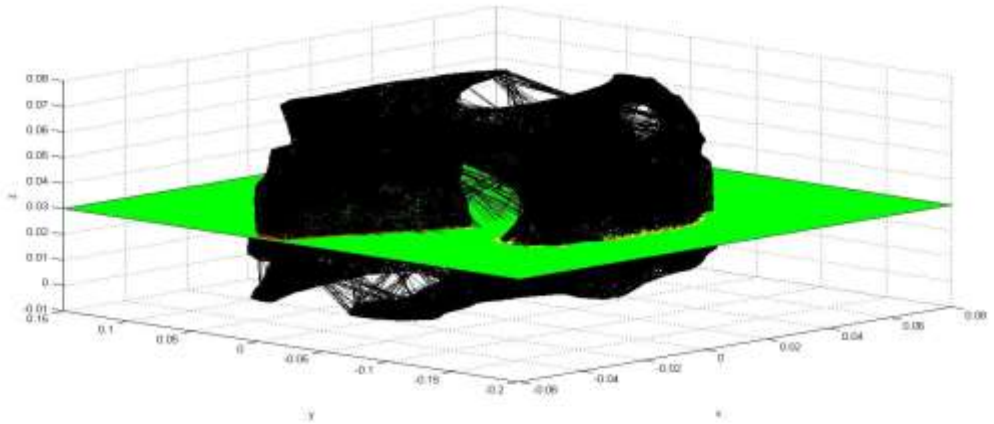


Figure 4-23 Contour generation of topology optimization result for the FSAE RACE CAR STEERING Knuckle Case at the horizontal plane

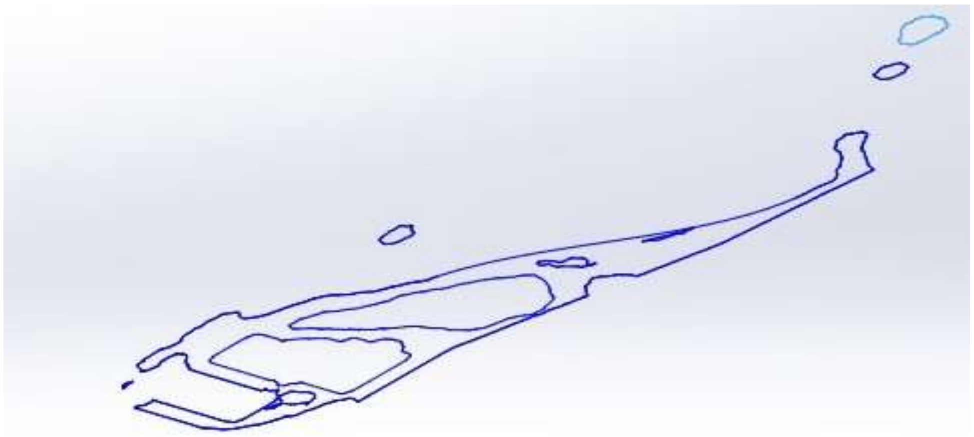


Figure 4-24 Contours exported from MATLAB and imported into SOLIDWORKS for the FSAE RACE CAR STEERING Knuckle Case at the horizontal plane

4.4.2 FSAE Race Car Steering Knuckle case for an inclined plane

For an inclined plane case, table 9 shows parameters of the plane and X, Y, Z coordinates. Figure 4-24 and 4-25 shows different views of the contour generation for the FSAE RACE CAR Steering knuckle at an inclined plane. Figure 4-26 shows importation of contours in SOLIDWORKS.

In spite of 93,998 numbers of facets, Contour generation and Separation for the FSAE RACE CAR Steering knuckle case is achieved successfully. This implies that the proposed method and algorithm is stable and capable enough for large STL files.

Table 9: Plane coordinates and X Y and Z coordinates which are used to separate and generate contours for topology optimization result of the FSAE RACE CAR Steering knuckle case at an inclined plane

Steering knuckle case	Inclined Plane
Number of facets	93998
Plane coordinates p1	[-0.06 -0.2 0.03]
Plane coordinates p2	[0.08 -0.2 0.06]
Plane coordinates p2	[0.08 0.15 0.06]
X coordinate used to separate contours	0.05
Y coordinate used to separate contours	0
Z coordinate used to separate contours	0.02

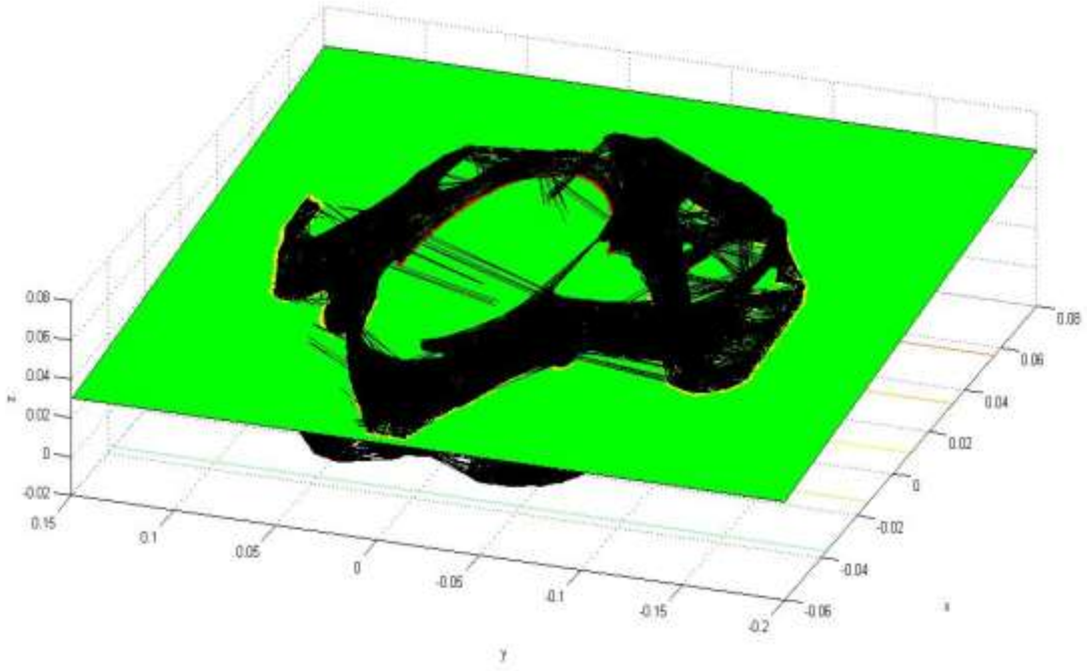


Figure 4-25 Contour generation of the topology optimization result view a for the FSAE RACE CAR STEERING Knuckle Case at an inclined plane

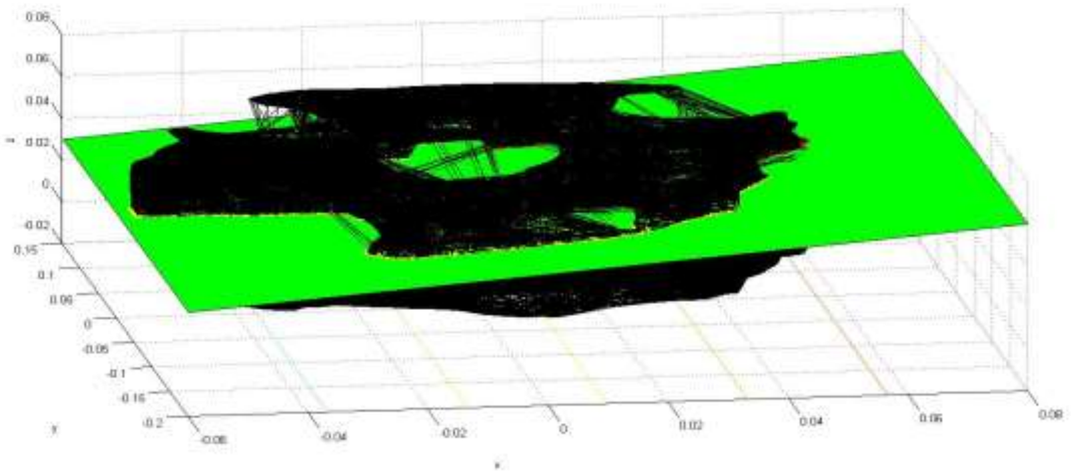


Figure 4-26 Contour generation of the topology optimization result view b for the FSAE RACE CAR STEERING Knuckle Case at an inclined plane

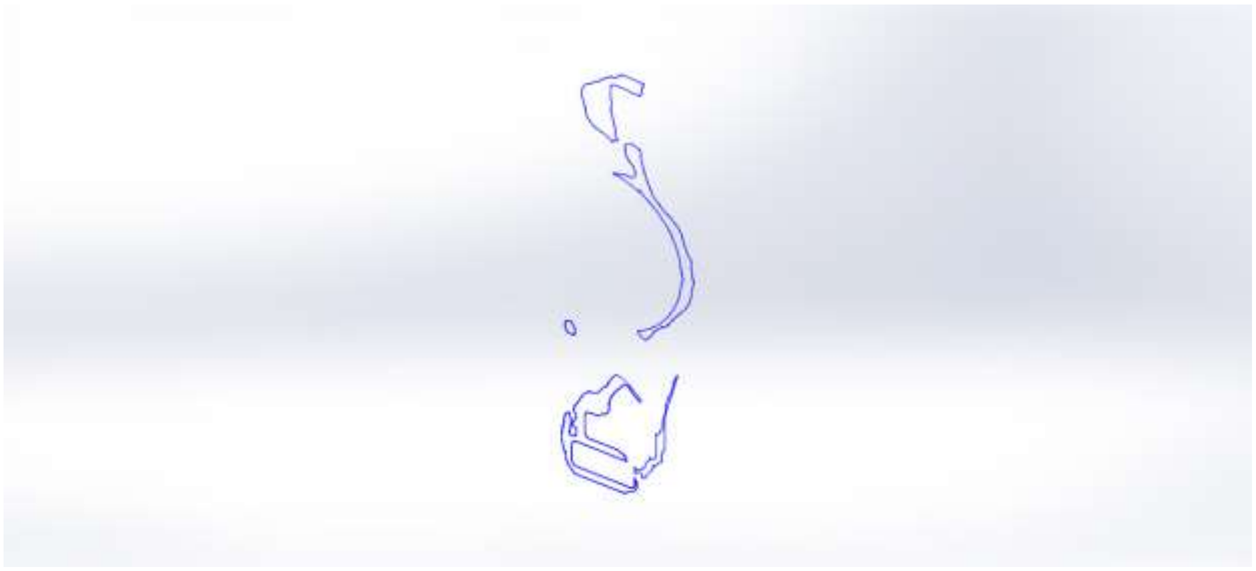


Figure 4-27 Contours exported from MATLAB and imported into SOLIDWORKS of the topology optimization result for the FSAE RACE CAR STEERING Knuckle Case at an inclined plane

Chapter 5

Conclusion

As introduced in chapter 1, Additive Manufacturing Technique builds a 3D geometry using layer by layer fabrication. 3D printing is a significant Additive Manufacturing technique. To convert topology optimization results into printable format, slicing of an STL file is very important. As discussed in chapter 2, typical slicing algorithm is not applicable for 3D complex geometry.

This research work proposes a new method to find intersection points between an arbitrary plane and a line of the facet data. It also includes Tail to Head distance algorithm and Separation of Contours. Application of this research work has been tested using simple cases like a cylinder, table with two legs and more complex cases like missile fin and FSAE RACE CAR Steering knuckle. After reviewing all the above tests, we can conclude that this method and algorithm is applicable for any complex geometry and has a better fit than existing slicing algorithm. For such a complex 3D geometry, Separation of Contours provides the user with a feature to remove extra contours and focus on the required contour to obtain precise and better results.

This process of performing analysis on complex geometry that has a large number of facets takes relatively longer time than usual, specifically during Tail to Head distance calculation. Time taken by the analysis is approximately six minutes for the FSAE Race Car Steering knuckle geometry as it has 93,998 numbers of facets to be scanned.

After generating contours using above methods, it is very difficult to trace them from MATLAB Graphics. Proposed future work includes improvisation for stated limitations.

Chapter 6

Future Work

As we conclude in chapter 5, visualization of complex 3D STL file and contours is very difficult. As facets increase, tracing of contours from MATLAB graphs becomes more complex. Hence, the future work focuses majorly on improvising visualization for complex geometry.

As number of facets increase, computational time increases for contour generation. It might take long time for large STL files. In this research work, time factor is not taken into consideration but we can reform the algorithm based on time complexity and make it more efficient for large STL files. In future, a feature can be added to calculate the centroid of the result point matrix in this algorithm. By calculating centroid, automatic geometry creation can be done.

For separation of contours, when the first row of Tail to Head distance result matrix is repeating again, the values contained between the two are stored in a separate matrix represented as a new page. After scanning all the data, contour matrices are separated into “n” number of pages and then into individual matrix to compute contour. In this case, for each individual matrix the page size is the same but the contour matrices may vary. This variation in size of each matrix contour is addressed by adding zeros to match the page size. This makes it very complex to export contours as it carries null values with it and this can be eliminated in future work.

Appendix A

Read an STL File

TO read an ASCII STL file as follows:

```
function [X, Y, Z, facetTotal] = stl_ASCII_read(stlFile)

    % Read the ASCII STL file

    fileInput = fopen(stlFile); %this is used to open file
    fileContentCell = textscan(fileInput,'%s','delimiter','\n'); %textscan reads data from an
openfile into a cell array
    fileContent = fileContentCell{:}(strncmp(fileContentCell{:},'vertex',6)); %compare
fileContentCell and find vertex till 6 char
    fclose(fileInput); %this is used to close the file

    % Read the vertex coordinates

  facetTotal = sum(strncmp(fileContent,'vertex',6)) / 3; %this is used to find the total
number of facets
   strVertices = char(fileContent(strncmp(fileContent,'vertex',6))); %this method finds
char vector and then convert into array.

coordinateVerticesAll = str2num(strVertices(:,8:end));
%this stores all numerical value from vertices string into coordinateVerticeAll
```

```
    % Save vertex coordinates in seperate X, Y, Z matrices

    COLUMN1 = coordinateVerticesAll(:,1);

    [X] = vec2mat(COLUMN1,3);

    COLUMN2= coordinateVerticesAll(:,2);

    [Y] = vec2mat(COLUMN2,3);

    COLUMN3 = coordinateVerticesAll(:,3);

    [Z] = vec2mat(COLUMN3,3);

End
```

Appendix B

Main.m File

```
            %Input STL file to perform contour generation.
            %stl_ASCII_read function reads the STL file and generates the vertex and total facet
count.
   function [] = main()
   file = 'filename.STL';
 [x, y, z,facetTotal] = stl_ASCII_read(file);

%planeGenerate() returns the plane coodinates to be generated.
 [p1, p2, p3] = planeGenerate();

%findIntersectionPoints() returns intersection points between arbitary plane and facets.
 [intersectPointMatrix] = findIntersectionPoints(x, y, z, p1, p2, p3, facetTotal);
%removeDuplicates() removes the duplicate coordinates generated in same or opposite
direction and generates the row count
[contourPointMatrix, contourPointMatrix_row] = removeDuplicates(intersectPointMatrix,
facetTotal);

%distanceSortAlgorithm() finds the minimum distance from the first row of the
contourPointMatrix  and generates pagination for the contour that are separate
[cntxMatrix1, totalPages, whichPage] = distanceSortAlgorithm(contourPointMatrix,
contourPointMatrix_row);
%Plot the contour, planes and the STL file data
plotContours(x, y, z, p1, p2, p3,cntxMatrix1, totalPages, whichPage);

            end
```

Appendix C

Planegeneration.m File

```
%This function uses simple plane equation to generate plane and it requires user input:


function [p1, p2, p3] = planeGenerate()

P1 = 'Enter the coordinate of plane Point 1 ([a b c])';

P2 = 'Enter the coordinate of plane Point 2 ([a b c])';

P3 = 'Enter the coordinate of plane Point 3 ([a b c])';


p1 = input(P1);

p2 = input(P2);

p3 = input(P3);

end
```

Appendix D

Findanintersectionpoint.m File

```
function [intersectPointMatrix] = findIntersectionPoints(x, y, z, p1, p2,p3, facetTotal)

 % Code to find intersection points between plane and cylinder
  % Till this point, we are reading ASCII file and save the facet
 % coordinates in matrices as:
 %  X = [x1 x2 x3]
%   Y = [y1 y2 y3]
%   Z = [z1 z2 z3]

 %Following code uses X, Y, Z matrices for further operations

        intersectPointMatrix = zeros([facetTotal 6]);

%For each the facet coordinate scan the intersections points, and remove the intersection that
lie beyond the range of the facet

        for i = 1:facetTotal
          columnShift = 0;
          for j = 1:3
            k = mod(j+1,4);
            if k == 0
              k = 1;
            end
```

```
%Defining range of the facet

    x_max = max(x(i,j), x(i,k));
    x_min = min(x(i,j), x(i,k));

    y_max = max(y(i,j), y(i,k));
    y_min = min(y(i,j), y(i,k));

    z_max = max(z(i,j), z(i,k));
    z_min = min(z(i,j), z(i,k));

%Finding the normal vector of the plane

        planeVect1 = p1 - p2;
        planeVect2 = p1 - p3;
        planeNorm = cross(planeVect1, planeVect2);

%Finding the normal vector of the line for a facet

        lineNorm = [(x(i,k) - x(i,j)) (y(i,k) - y(i,j)) (z(i,k) - z(i,j))];
        intersectCheck = cross(planeNorm, lineNorm);
        dotProduct = dot(planeNorm, lineNorm);
        %The intersection coordinate of the plane and line of a facet is found

            if (any(intersectCheck) || dotProduct)

                d = (planeNorm(1)*p1(1)+planeNorm(2)*p1(2)+planeNorm(3)*p1(3));

                t= (d - (planeNorm(1) * x(i,j) + planeNorm(2) * y(i,j) + planeNorm(3) * z(i,j))) /
(planeNorm(1) * (x(i,k) - x(i,j)) + planeNorm(2) * (y(i,k) - y(i,j)) + planeNorm(3) * (z(i,k) - z(i,j)));

                x_coordinate = x(i,j)+(t*(x(i,k) - x(i,j)));

                x_validate = (isinf(x_coordinate) | isnan(x_coordinate));

                y_coordinate = y(i,j)+(t*(y(i,k) - y(i,j)));

                y_validate = (isinf(y_coordinate) | isnan(y_coordinate));

                z_coordinate = z(i,j)+(t*(z(i,k) - z(i,j)));

                z_validate = (isinf(z_coordinate) | isnan(z_coordinate));

                if ((x_coordinate <= x_max && x_coordinate >= x_min) && (y_coordinate <=
y_max && y_coordinate >= y_min))

                    x_range = 1;
                else
```

```
    x_range = 0;
end

if (y_coordinate <= y_max && y_coordinate >= y_min)
    y_range = 1;
else
    y_range = 0;
end

if (z_coordinate <= z_max && z_coordinate >= z_min)
    z_range = 1;
else
    z_range = 0;
end

if(x_range && y_range && z_range && ~x_validate && ~y_validate &&
~z_validate)

    intersectPointMatrix(i, 1 + columnShift) = x_coordinate;

    intersectPointMatrix(i, 2 + columnShift) = y_coordinate;

    intersectPointMatrix(i, 3 + columnShift) = z_coordinate;
    columnShift = 3;

end
```

```
            end
         end
      end
   end
```

Appendix E

Removeduplicates.m File

```
function [contourPointMatrix, contourPointMatrix_row] = removeDuplicates(intersectPointMatrix,
facetTotal)

  % Removing the duplicate points from intersectPointMatrix
contourPointMatrix = zeros([facetTotal 6]);
 zero = [0 0 0 0 0 0];
 contourPointMatrix_row = 1;
duplicateCount = 0;

  for i=1:facetTotal

 if any(intersectPointMatrix(i,:))

contourPointMatrix(contourPointMatrix_row,:) = intersectPointMatrix(i,:); % contourPointMatrix
will have all the coordinates after removing duplicates
                         contourPointMatrix_row = contourPointMatrix_row + 1;
% Duplicte for w/o swap
 straightCompare = ismember(intersectPointMatrix,intersectPointMatrix(i,:),'rows');

% Duplicate for swap cases
  swapMatrix = intersectPointMatrix(:,[4:6 1:3]);
swapCompare = ismember(swapMatrix,intersectPointMatrix(i,:),'rows');
```

```
 % Scan for match cases
                        for j=(i+1):facetTotal


                           if (((straightCompare(j) == straightCompare(i)) || (swapCompare(j) ==
straightCompare(i))) && (i ~= j))
                               intersectPointMatrix(j,:) = zero;
                               duplicateCount = duplicateCount + 1;
                           end
                        end
                    end
                end
            end
```

Appendix F

Distancesortalgorithm.m File

```
            function[cntxMatrix1, totalPages, whichPage] =
distanceSortAlgorithm(contourPointMatrix, contourPointMatrix_row)

             % Finding the nearby contour points and sort the contourPointMatrix
               finalFacets = contourPointMatrix_row - 1;
               distanceMatrix = zeros([finalFacets 3]);

            % Tail to head distance finding algorithm
            % Using the first matrix line, find the head and tail distance for other points.
            % Find the nearest line using its head.
            % Swaps the line coordinates if tail is nearer than the head

            for i=1:finalFacets
                  needRowSwap = 0;
                  minDistance = 65535;

                  for j=(i+1):finalFacets
                     endPoints_head = [contourPointMatrix(i,4), contourPointMatrix(i,5),
contourPointMatrix(i,6); contourPointMatrix(j,1), contourPointMatrix(j,2), contourPointMatrix(j,3)];
                     distance_head = pdist(endPoints_head,'euclidean');
                     endPoints_tail = [contourPointMatrix(i,4), contourPointMatrix(i,5),
contourPointMatrix(i,6); contourPointMatrix(j,4), contourPointMatrix(j,5), contourPointMatrix(j,6)];
                     distance_tail = pdist(endPoints_tail,'euclidean');

                     distanceMatrix(j) = distance_head;
                     distanceMatrix(j,2) = distance_tail;
```

```
    if distance_tail < distance_head
        internalSwap = 1;
        tempDistance = distance_tail;
    else
        internalSwap = 0;
        tempDistance = distance_head;
    end

    distanceMatrix(j,3) = internalSwap;

    if tempDistance < minDistance
        minDistance = tempDistance;
        rowSwapIndex = j;
        needRowSwap = 1;
        internalSwapFinal = internalSwap;
    end
end

if needRowSwap
    tempMatrix = contourPointMatrix(rowSwapIndex,:);

    if internalSwapFinal
        tempMatrix = tempMatrix([4 5 6 1 2 3]);
    end
```

```
            contourPointMatrix(rowSwapIndex,:) = contourPointMatrix(i+1,:);
            contourPointMatrix(i+1,:) = tempMatrix;
        end
    end

    split_1 = contourPointMatrix(1:finalFacets,[1 2 3]);
    split_2 = contourPointMatrix(1:finalFacets,[4 5 6]);
    [nRowsA,nCols] = size(split_1);
    nRowsB = size(split_2,1);

    contourMatrixFinal = zeros(nRowsA+nRowsB,nCols);
    contourMatrixFinal(1:2:end,:) = split_1;
    contourMatrixFinal(2:2:end,:) = split_2;

    contourMatrixFinal = times(contourMatrixFinal,10000);
    contourMatrixFinal = round(contourMatrixFinal);
    contourMatrixFinal = times(contourMatrixFinal,1/10000);

% Generate n matrixes for n contours
% Split the contour matrix from a point that completes a closed contour
% Find the number of such split matrixes and name it as a page

     page = 1;
    totalsize = nRowsA+nRowsB;
    i = 1;
    while (i <= totalsize)
    p = 1;
    firstElement = contourMatrixFinal(i,:);
        cntxMatrix(p,:,page) = firstElement;
        p = p+1;
   i = i + 1;
    while ~(isequal(firstElement, contourMatrixFinal(i,:)))
        cntxMatrix(p,:, page) = contourMatrixFinal(i,:);
        if (i <= (nRowsA+nRowsB))
            i = i + 1;
        else
            disp('Code might be stuck into infinite loop so terminating...');
            return
        end
        p = p + 1;
    end
    cntxMatrix(p,:,page) = contourMatrixFinal(i,:);
    p = p + 1;
    i = i + 1;
    page = page + 1;
end
totalPages = page - 1;

matSize = size(cntxMatrix(:,:,1));
matSize1 = matSize(1) - 1;

  for p = 1:totalPages
```

```
        j = 1;
         for i=1:matSize1
             diff = cntxMatrix(i,:,p) - cntxMatrix(i+1,:,p);
             if (any(diff))
                 cntxMatrix1(j,:,p) = (cntxMatrix(i,:,p));
                 j = j + 1;
             end
             if(i == matSize1 && p == totalPages)
                cntxMatrix1(j,:,p) = (cntxMatrix(i+1,:,p));
             end
         end
      end

% User input to find the desired contours

  X = 'Enter X Co-ordinate:';
     x = input(X);
     Y = 'Enter Y Co-ordinate:';
     y = input(Y);
     Z = 'Enter Z Co-ordinate:';
     z = input(Z);
```

```
          nearbyIntersectFindPoint = [x y z];
          minDist = 65535;

      % Finding the contour nearest to the user input

          for i = 1:totalPages
            length = size(cntxMatrix1(:,:,i));
            for j = 1:length
               twoPoints = [nearbyIntersectFindPoint(1), nearbyIntersectFindPoint(2),
nearbyIntersectFindPoint(3); cntxMatrix1(j,1,i), cntxMatrix1(j,2,i), cntxMatrix1(j,3,i)];
               nearestIntersectPoint = pdist(twoPoints,'euclidean');
               if nearestIntersectPoint < minDist
                  minDist = nearestIntersectPoint;
                  nearestPoint = [cntxMatrix1(j,1,i) cntxMatrix1(j,2,i) cntxMatrix1(j,3,i)];
                  whichPage = i;
               end
            end
          end
      end
```

Appendix G

Plotcontours.m File

```
function[] = plotContours(x, y, z, p1, p2, p3, cntxMatrix1, totalPages, whichPage)
 % Plot for STL file
    surf(x, y, z, 'FaceColor','none')
    xlabel('x'); ylabel('y'); zlabel('z');
    hold on
% Plot for Plane
%Create a mesh using the x coordinate of the three input points.
    x = p1(1):p2(1):p3(1);
    y = p1(2):p2(2):p3(2);
    [X,Y] = meshgrid(x,y);
 % Finding the value of Z using plane formula
    a = (p2(2)-p1(2)) * (p3(3)-p1(3)) - (p3(2)-p1(2)) * (p2(3)-p1(3));
    b = (p2(3)-p1(3)) * (p3(1)-p1(1)) - (p3(3)-p1(3)) * (p2(1)-p1(1));
    c = (p2(1)-p1(1)) * (p3(2)-p1(2)) - (p3(1)-p1(1)) * (p2(2)-p1(2));
    d = -(a * p1(1) + b * p1(2) + c * p1(3));
    Z = -(d + a * X + b * Y)/c;
 surfc(X, Y, Z,'FaceColor',[0 1 0]')
    % Plot of the resulting contour matrix
 for i = 1:totalPages
     plot3(cntxMatrix1(:,1,i), cntxMatrix1(:,2,i), cntxMatrix1(:,3,i), '-Y*');
 end
 plot3(cntxMatrix1(:,1,whichPage), cntxMatrix1(:,2,whichPage),
cntxMatrix1(:,3,whichPage), '-R*');
 end
```

References

[1] Achille Messac, “*Optimization in Practice with MATLAB®: For Engineering Students and Professionals”,* March 18, 2015

[2] Siddhant Brahmbhatt,Harshit Jani,Divya Shukla “*Design Optimization of the ROLL-CAGE FOR MINI BAJA CAR” ,*Project Report, Guide : Dr. Bo P Wang, December 2015.

[3] Karen Wood, “*High performance Composites*,” 2015. [Online]. Available: http://www.compositesworld.com/articles/software-update-simulation-saves [Accessed: 25-October-2016].

[4] “*Topology Optimization*,” *Wikipedia*, 20-May-2016. [Online]. Available: https://en.wikipedia.org/wiki/Topology_optimization. [Accessed: 15-July-2015]

[5] Liu, K. & Tovar, *“An efficient 3D topology optimization code written in Matlab”,* Volume 50, pp 1175–1196, June 25, 2014

[6] Martin Philip Bendsoe, Ole Sigmund, “*Topology Optimization: Theory, Methods, and Applications”,* Apr 17, 2013.

[7] Brent Stucker, David H. Rosen, and Ian Gibson*, “Additive Manufacturing Technologies: 3D Printing, Rapid Prototyping, and Direct Digital Manufacturing”,* December 14, 2009

[8] Zegard, T. & Paulino, G.H, *“Bridging topology optimization and additive manufacturing”,* August 5, 2015.

[9] Frank W. Liou, “*Rapid Prototyping and Engineering Applications: A Toolbox for Prototype”,* September 26, 2007.

[10] Jonas Gomes, Luiz Velho, “*Image Processing for Computer Graphics”*, Springer, ISBN: 0387948546,LC: T385.G65,1997 [Online]. Available:

https://people.sc.fsu.edu/~jburkardt/m_src/image_edge/image_edge.html. [Accessed: 25-October-2015]

[11] "*Stereo lithography*," *Wikipedia*, [Online]. Available: https://en.wikipedia.org/wiki/STL_(file_format) [Accessed: 20-January-2016]

[12] Paul Bourke, *"STL Format",* October 1999 [Online]. Available: http://paulbourke.net/dataformats/stl/ [Accessed: 25-January-2016]

[13] M Vatani, A. R. Rahimi, F. Brazandeh and A. Sanati nezhad, *" An Enhanced Slicing Algorithm using nearest distance analysis for Layer Manufacturing",* World Academy of Science, Engineering and Technology 2009.

[14] Yobani Martinez, *" Design Optimization of Race car steering knuckle for Additive Manufacturing ",* Master's thesis report, The University of Texas, Arlington, December 2016.

Biographical Information

Divya Shukla has completed her Bachelor’s degree in Aeronautical Engineering from Sardar Vallabhbhai Patel Institute of Technology from Vasad, India. After completion of B.E in Aeronautical, She found her strong interest in Finite Element Method and Design Optimization field. To gain advance technical knowledge, she joined The University of Texas, Arlington in August 2014 for Master of Science in Mechanical Engineering.