12-2016

Uniform Micro-Patterning of an Arbitrary Surface

Viraj Rajendra Kulkarni
Clemson University

UNIFORM MICRO-PATTERNING OF AN ARBITRARY SURFACE

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mechanical Engineering

---

by
Viraj Rajendra Kulkarni
December 2016

Accepted by:
Dr. Georges M. Fadel, Committee Chair
Dr. Gregory Mocko
Dr. Rodrigo Martinez-Duarte

ABSTRACT

According to the literature, creating specific micro-level patterns on some surfaces can significantly reduce friction. To this effect, a method is presented to create a regular pattern of micro-level indentations on any irregular surface. Creating a uniform pattern on a regular surface is possible using commercial CAD software, where regular surface is the surface obtained by extrusion or revolution of a 2D sketch along any curve. But, it is complicated and often incorrect for irregular surfaces. The thesis presents the approach followed to create parameterized regular patterns on arbitrary surfaces. Three different algorithms are presented, each achieving a progressively increased quality solution. The last and best method provides a set of points with their corresponding normals to the surface to enable the creation of the patterning feature. The algorithm reads an STL file, a format neutral output of any CAD software and implements the method on the approximated surface. Each facet surface upon which the pattern has to be created is sliced by planes at specific distances from each other. The intersections of the facets and the planes are calculated and chains are formed from the intersections in each plane. Points are interpolated at the required pitch in different chains formed at the intersection of a single plane and the facets. This procedure is repeated for each plane. Thus, a pattern of points of specified pitch distance that can be as low as microns can be generated.

Given specifications of a machine, this method generates the X, Y, and Z translations and the axis rotation angles needed to generate a g-code specific to a micro-milling machine. This code can be used directly for any metal removing process that has

to create micro-level indentations on an arbitrary surface. If instead, the features are protrusions on some irregular surface, then the resultant points obtained with the developed approach can be used to apply the pattern at each of the identified locations.

DEDICATION

I would like to dedicate my work to my family, who always stood by me and supported me in whatever I pursue.

ACKNOWLEDGMENTS

I would like to express my sincere thankfulness to my advisor Dr. Georges M. Fadel for all his continuous guidance and support. He has always encouraged me to think more and come up with different ideas and solutions.

I would like to thank Dr. Gregory Mocko and Dr. Rodrigo Martinez-Duarte for providing valuable inputs through the entire course of my research work.

I would like to thank all my colleagues of the Department of Mechanical Engineering, Clemson University for their assistance throughout my research.

TABLE OF CONTENTS

Page

Table of Contents (Continued)

Page

LIST OF TABLES

LIST OF FIGURES

List of Figures (Continued)

List of Figures (Continued)

NOMENCLATURE

NURBS	Non-uniform rational Basis spline
B-reps	Boundary Representation
tria	Triangular Element
rtria	Right angled triangular element
quad	Quadrilateral Element
CAD	Computer Aided Design
IGES	Initial Graphics Exchange Specification
STEP	Standard for the Exchange of Product model data
VRML	Virtual Reality Modeling Language
ASCII	American Standard Code for Information Interchange
TOL	Tolerance

1 INTRODUCTION

1.1 Motivation

In [5], Ramesh et al. have shown that producing micro-textures on stainless steel surfaces have enhanced their frictional performance. They conducted a study varying the micro-textures pattern geometry. Width, depth, and pitch of the holes were three parameters defined for the study. The width of the hole was considered to be the diameter of the hole. The depth of the hole was defined as the extent up to which the hole is drilled. Pitch was defined as the distance between two holes. These three parameters were varied by microns. By carrying out various experiments, they have reported a friction reduction as high as 80% for textured surfaces when compared with the normal surfaces with no texture. The textured stainless steel surface considered consists of uniform micro-scale holes at uniform distances from each other. They have observed that friction increases significantly when the texture width and density is decreased. They have identified the possibility of applying micro-scale textures to surfaces in fluid power systems like seals, pumps, and valves for a noteworthy reduction in the friction.

In [6], Braun et al. have investigated the effect of varying the size of spherical dimples with diameters ranging from 15 to 800 μm and a depth-to-diameter ratio of 0.1 on friction reduction. At high pressure regimes, there may be a considerable rise in fluid viscosity, causing a separation in surface and possibly generating contact between raised solid features and asperities as a result. This state is called a mixed lubrication. Tribological tests were performed under mixed lubrication against bearing steel 100Cr6.

They have stated that the pressure built up depends on the number of dimple edges and their size. A severe dependence of the dimple diameter was found on the friction reduction. They achieved an 80% reduction in friction with what they consider to be an optimum texture, which they identified for certain operating conditions. The number of dimple edges provide the count for the micro-pattern generated on the surface. The diameter and the depth represent the other characteristic for the micro-pattern on the surface.

In [7] Ventola et al. have produced diamond shaped micro-protruding patterns with the objective of improving convective heat transfer in electronic cooling applications. They have achieved significant improvements in thermal performance per unit production cost of up to 73%. , with respect to the considered commercial heat sink. They have concluded that the methodology can be extended to various and diverse micro-protruding or micro-structured patterns.

The research issue is to find a method to create the micro-level patterns on arbitrary surfaces.

1.2 Literature Review

Creating patterns on an irregular surface requires multiple mathematical techniques based on computational geometry. Much work exists on relevant aspects of the problem. The literature cited below addresses some of these most critical aspects.

In [8], Shih et al. have proposed an algorithm to compute the single-sided offset approximation of a NURBS curve as a tool path for NURBS machining. Their proposed method results in NURBS path curves with no self-intersections and no cusps. The practical test for the algorithm was conducted for a 2.5D mold. It is a 2D sketch with an extrusion thickness. Figure 1.1 gives an example of a 2.5D mold. The mold is obtained by extruding a 2D sketch with a given thickness in the perpendicular direction to the sketch plane.

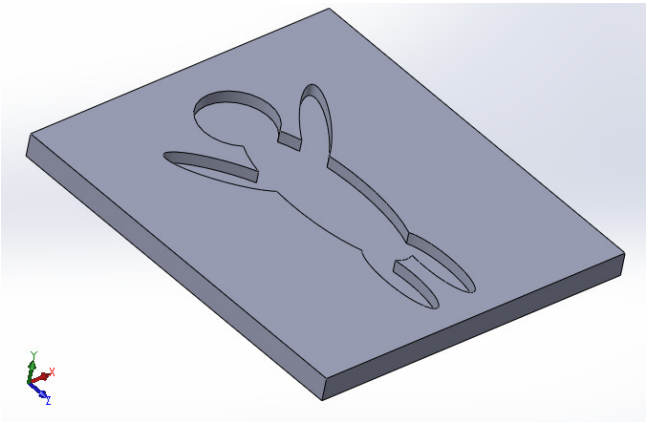


Figure 1.1 Example of 2.5 D Mold

To compute the single-sided offset of the curve, a tolerance is defined on the offset value for obtaining the offset curve. It consists of dividing the tolerance ε defined by the user into two parts ε_1 and ε_2. It is followed by obtaining a single-sided approximation of the original curve, considering an error of ε_1. This is a linear approximation. ε_1 is determined by average error of the corner points of the approximation curve with actual curve.

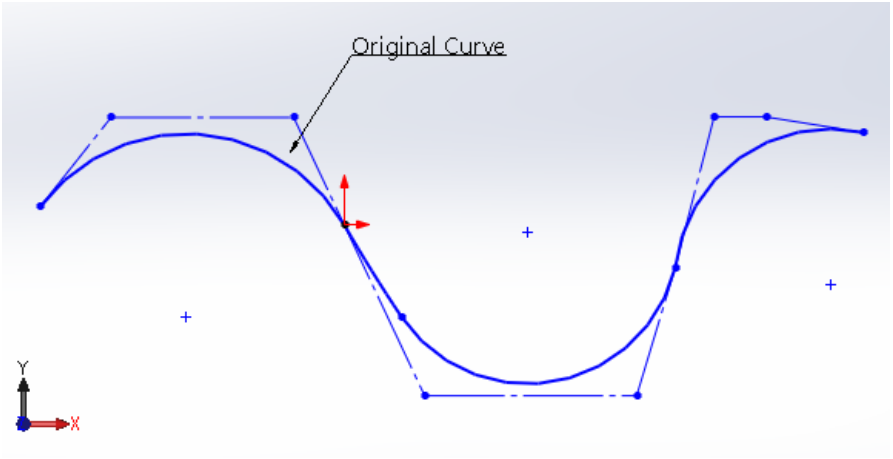


Figure 1.2 Original Curve [8]

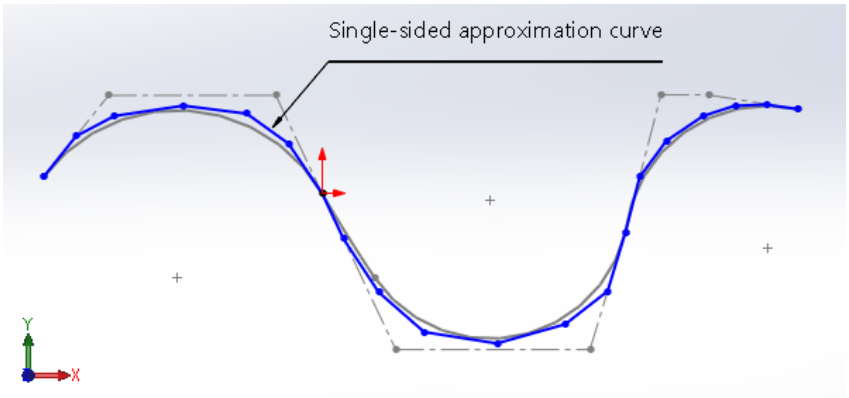


Figure 1.3 Single-Sided Linear Approximation Curve [8]

Figure 1.3 is obtained by approximating the original curve from figure 1.2, considering an error of ε_1. By lowering the error, the linear approximation tends towards the original curve. Further, $\varepsilon_2/2$ is added to the prescribed distance to offset the approximation curve as shown in figure 1.4

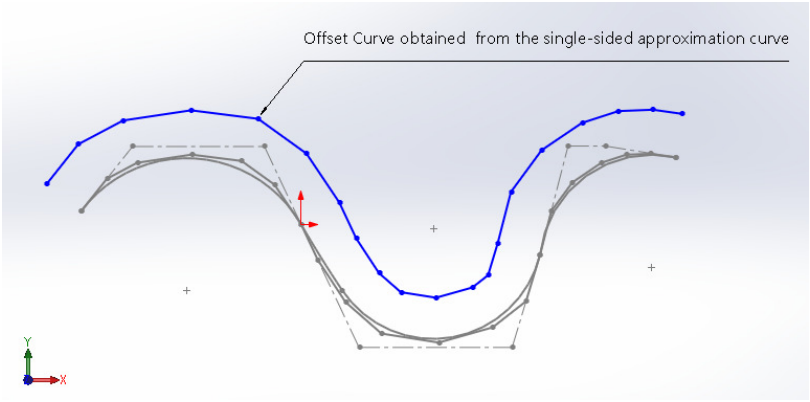


Figure 1.4 Offset of Single-Sided Linear Approximation Curve [8]

It is followed by offsetting the generated offset curve backwards by a distance $\varepsilon_2/2$ and then again offset by distance less than or equal to $\varepsilon_2/2$.

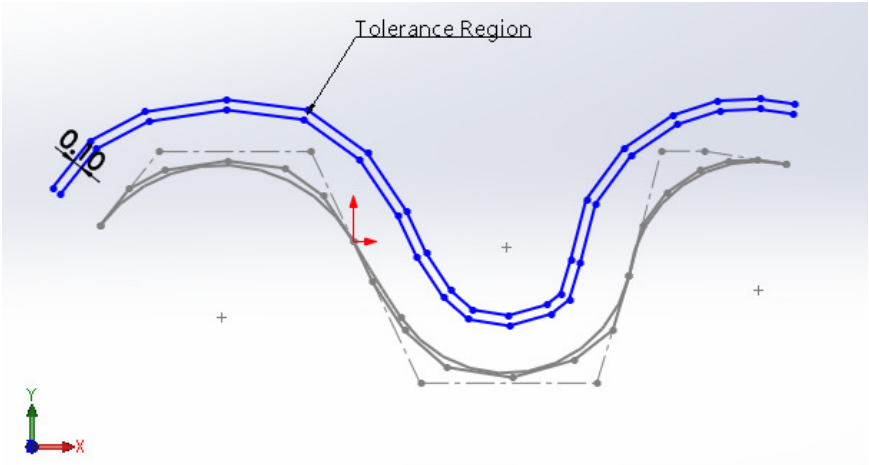


Figure 1.5 Tolerance Region [8]

A tolerance region is achieved by the process as shown in figure 1.5. The two layers of lines are the backward and forward offsets with $\varepsilon_2/2$ error to form a tolerance region of error ε_2. A C^1 piecewise Bezier curve is constructed considering all the control points that lay in the obtained tolerance region. Thus points can be patterned along the

offset curve at uniform distances from each other. To obtain offset curves on a 2d surface, the inner offset is obtained from the outermost boundary of the surface. This process is continued until it is not possible to offset the curve more, where the offset distance is more than the dimensions of the area available. Accordingly, the last offset curve is obtained in the process. For a single 2D surface, there may be many regions where the last inner offset curve is obtained. In addition, there exists many cases where the offset curves obtained display sharp corners. In such cases the pattern of points on the outer offsets may not have a uniform distance from the inner offset, thus a uniform pattern is not guaranteed. Also this study was conducted for a 2.5D object. This method has not been extended to the offsetting of a 3D sculptured surface.

In [9], Ravi Kumar et al. present an approach to offset a NURBS B-Rep surface used for a class of manifold B-Reps. The faces of the B-Rep are offset then followed by a process of removing gaps and intersections. The new offset faces are stitched together. The approach rests on the assumption that faces are at least G_1 continuous and do not have any intersection when a constant offset is applied. When two surfaces share a common edge and the adjoining faces are tangent, they are said to be G_1 continuous.

In [10], Schroeder et al. propose a decimation algorithm for an STL file. Decimation is reducing the number of triangles in a triangle mesh, maintaining the original topology of the surface and also maintaining a good approximation of the original surface. Multiple passes on all the vertices are done, where in each pass every vertex is checked with the decimation criteria. A vertex and the triangles using it are

removed when a vertex meets the decimation criteria. Triangulation is done for the resulting hole created.

In [12], Tata et al. introduced a method for an adaptive slicing algorithm. This method varies the layer thickness according to the complexity of the surface with cusp height, maximum deviation and chord length. This algorithm introduces a simple backtracking technique where there is an increase in surface complexity. The next layer thickness can be calculated by the slicing direction making an angle with the original slice axis. Repeated backtracking is performed for multiple levels of complexities.

Throughout the literature, offsetting of curves has been highlighted. However, the work has not been extended to obtaining a uniform micro-pattern of points. Adding further, for machining operations like drilling at the points of the micro-pattern so obtained, needs additional inputs such as the translations and rotations applied. No work has been done yet to obtain the uniform micro-pattern of the points along with the required machining translations and rotations. Finding a method which does so, can be directly coupled to a g-code and machining operations could be done in succession.

1.3 Using 2D Meshing

The possibility of obtaining a patterning of points on surfaces using 2D meshing was investigated. 2D mesh is forming a finite element model from the given surface. There are different element types for 2D meshing. 'Trias' and 'quads' are the 2D element types which can be used for meshing. A 'tria' element is a triangle with 3 vertices and a 'quad' element is a quadrilateral with 4 vertices. The idea was to obtain a mesh in such a

way that length of the element would represent the patterning distance. 'Trias', 'rtrias', 'quads', 'quads' and 'quads only' are the types of 2D meshing. Trias method consists of all triangles and the rtrias method consists of all right angles triangles. The quads method consists of maximum quads and minimum trias. The mixed method consists of mixture of trias and quads, while not minimizing the number of quads. The quads only type of meshing will produce only quad elements. The quad only was chosen for meshing to obtain a uniform 2D mesh.

A surface was 2D meshed using Hypermesh 13.0. The CAD model shown in figure 1.6 was imported as an IGES format into Hypermesh. The objective of this experiment was to test if a uniform pattern of 0.5 mm can be obtained from 2D meshing.

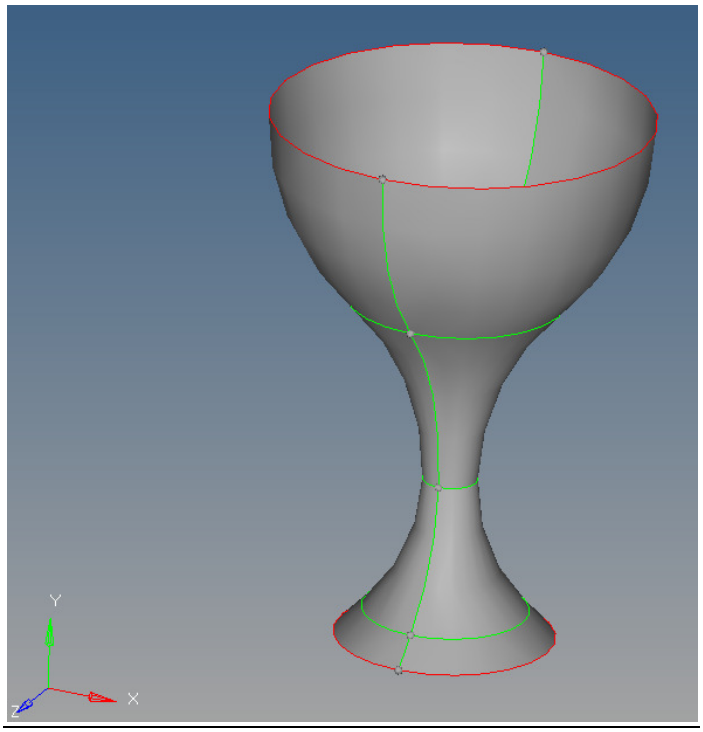


Figure 1.6: Surface Model of Glass

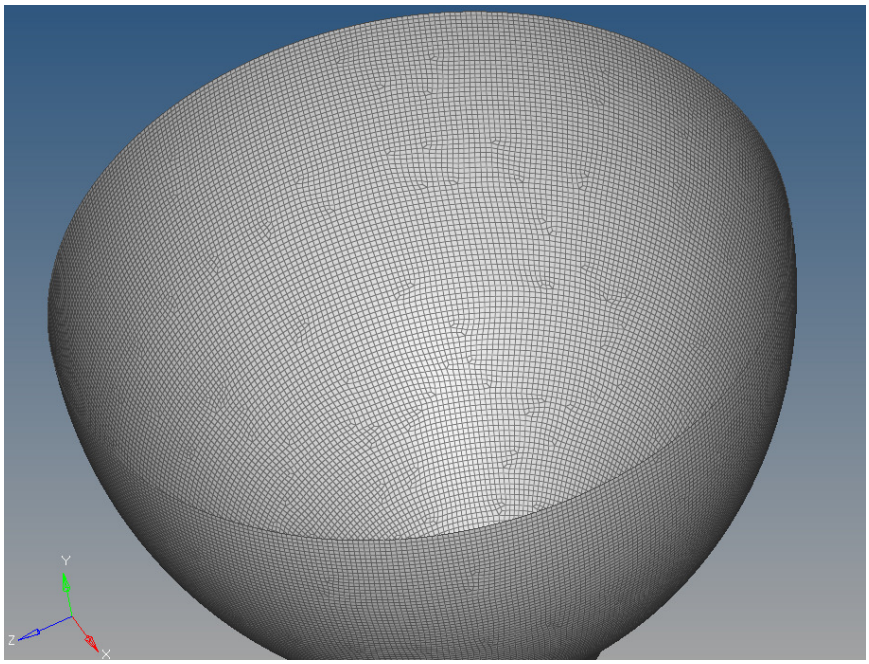


Figure 1.7: 2D Mesh- Surface Model of Glass.

The meshing shown in figure 1.7 was done by quads only. If all the quads have 0.5 mm element size, the pattern would be uniform. But it was observed that 28% of the elements had an element size or the side of the quad less than 0.45 mm and 13% had more than 0.55 mm with element size ranging from 0.21 mm to 0.72 mm. Efforts were made to achieve better mesh using quality index optimization or morphing. Quality index optimization runs an optimization to re-mesh the surface according to a user-defined criteria. The element size was given as a criterion. Morphing is applying an existing 2D mesh on a surface. But application of the quality index optimization meshing and morphing methods for 2D meshing still had the element size had variations. Therefore, to achieve a uniform pattern, more investigation and alternate methods were needed to be identified.

1.4 STL format

STL files are triangular facets representations of surfaces [2]. The STL file format was initially developed under a request by 3D systems company which was about to commercialize the first additive manufacturing process, namely, the stereolithography Apparatus or SLA. Many names have surfaced in the literature for STL, such as "STereoLithography format""Standard Triangle Language" and "Standard Tessellation Language"[3]. An STL file contains geometric information of a 3D object in form of triangular data. It includes the coordinates of the three vertices of each triangle and its normals. The coordinates are in order of rank one to three. Therefore the sides of the triangle formed are by vertices 1 and 2, vertices 2 and 3 and vertices 3 and 1. A cross product can be formed by any two ordered vectors to generate the normal. But as the STL file has normal data, this is not necessary.

The STL file can be encoded in ASCII or Binary formats. The ASCII format is easy to read but takes much more space to be stored in memory. Each data set representing a triangle begins with the term "facet" and ends with "end facet". Normal data is given in the first line inside that data set. Next, the vertices are listed inside a loop which begins with "outer loop" and ends with "endloop". Three vertices data are listed in three separate lines. So in the STL format, point coordinates are repeated when used in different triangles.

For the Binary format, a binary reader is required as a record of 80 characters represent a triangle. That format is much more compact and uses less space than the

ASCII format, but is not readable. For both of the formats, ASCII as well as Binary, the logic of reading an STL file is the same. It can be read as a string and split using delimiters. The character spacing defines the positions of the normal and the vertices for each and every loop.

```
facet normal ni nj nk
outer loop
vertex v1x v1y v1z
vertex v2x v2y v2z
vertex v3x v3y v3z
endloop
endfacet
```

Figure 1.8 ASCII format of STL facet

1.5 VRML format

The Virtual Reality Modeling Language, commonly known as VRML is a standard file format for 3D objects[4]. It also includes options for noting the complex behavior or animation. It consists of additional display parameters like Material ambient color, diffuse color, emissive color, shininess and transparency, etc.

VRML is a more compact and less error prone format than STL. A matrix in the VRML file stores all the coordinates of all vertices. Next, a Coordinate index matrix links the coordinates of the vertices to the triangles they belong to. The coordinate index matrix defines a triangle by specifying the three vertices numbers from the previous coordinates matrix. For instance, a triangle is identified as formed by the coordinates of

points 5,6, and8. The coordinates are in the coordinate matrix. The next triangle can be formed by points 6, 8 and 12. The coordinates of the points are not repeated, they are identified by the indices used.

If the VRML file is imported as a string, then the coordinates of the vertices and the coordinate index denoting the vertices chosen for each triangle can be identified by character spacing. The only problem with the VRML file is that there is no data for normals. Which implies that normals if required, have to be manually calculated. But from the coordinate index matrix we know the order of the vertices. The sides of the triangle formed are by vertex 1 and 2, vertex 2 and 3 and vertex 3 and 1. Thus a cross product can be formed by two vectors identified by their extremities. Once computed, the obtained vector from the cross product has to be converted into unit vector by dividing by its magnitude. This is explained in detail in chapter 2.

1.6 Research Objectives

The thesis has following primary objectives:

1. Create a micro-pattern of points with uniform pitch of 'd' at a micro level on any arbitrary surface.
2. Calculate the X,Y,Z translations and A & B (Machine) angles required by the machine for each point
3. Test the micro-patterning algorithm with different designs having various complexities

1.6 Thesis Organization

The thesis is organized into five chapters. The current chapter includes motivation, literature review and research objective.

Chapter 2 explains the different approaches to obtain the uniform patterning of points on any arbitrary surface. Three different approaches are attempted: The Direct Interpolation approach, the slicing approach and the slicing-chain approach.

Chapter 3 focuses on obtaining X, Y, Z translations and A,B angles for a 5-axis machine.

Chapter 4 focuses on the testing, results and discussion of the algorithm for different complexities of geometry. Planar surface, uniform curved surface and non-uniform curved surface are the three types of surface complexities tested.

Chapter 5 presents the conclusions and the future possible extensions of the work.

2. PATTERNING ALGORITHMS

To obtain a uniform pattern of points on a surface, different approaches are attempted. Both the STL and VRML file formats are considered as an input format since they both can be generated by most CAD software programs. The research objective is to obtain X, Y, Z translations and A & B angle rotations for a 5-axis machine to drill micro-sized holes perpendicular to the surface at regular intervals. Using an STL or VRML file provides either directly or by a cross product of vectors, the normals for each and every triangle. The STL/VRML formats are of lower size for storing than the original surface model. A CAD file can be converted quickly into these tessellated formats. The STL format is a widely accepted format, which is commonly used for 3D printing and rapid proto-typing. 3D printing is a method, in which an object is built in less time from the 3D printer, compared with the traditional machining methods. The 3D printing has gained more attention in recent as producing the models using rapid-prototyping has become much easier. Thus the STL format has its importance increased with the increasing usage of rapid prototyping.

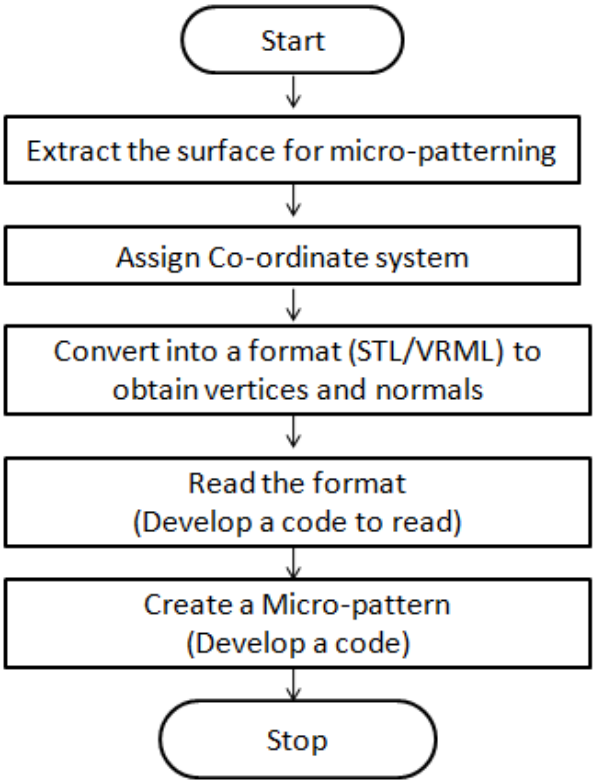


Figure 2.1: Flowchart for creating a Micro-patterning

The flowchart shown in figure 2.1 explains the process for creating a micro-pattern on any arbitrary surface. This is a general flowchart and does not vary with the type of algorithm used. It involves various steps from extracting a surface for micro-patterning, assigning a coordinate system, then converting the geometry into a tessellated surface. The extraction of the surface of interested and its conversion into a tessellated surface is done using Solidworks 2014, a CAD package. A code has been created to read the STL/VRML file from the CAD and to create a micro-pattern of points. Matlab 2014a has been used to write the code and execute it. Different approaches for micro-pattering are discussed in this chapter after a literature review describing the state of the art.

Further obtaining X,Y,Z translations and A & B angles for a 5-axis CNC machine is another part of the algorithm.

Suppose there is a surface of a solid component on which a micro-pattern is to be created. The algorithm for the micro-patterning only needs the surface on which the patterning has to be done. Therefore, the specific component is opened in any CAD application. Solidworks 2014 has been used in this work to import the component.

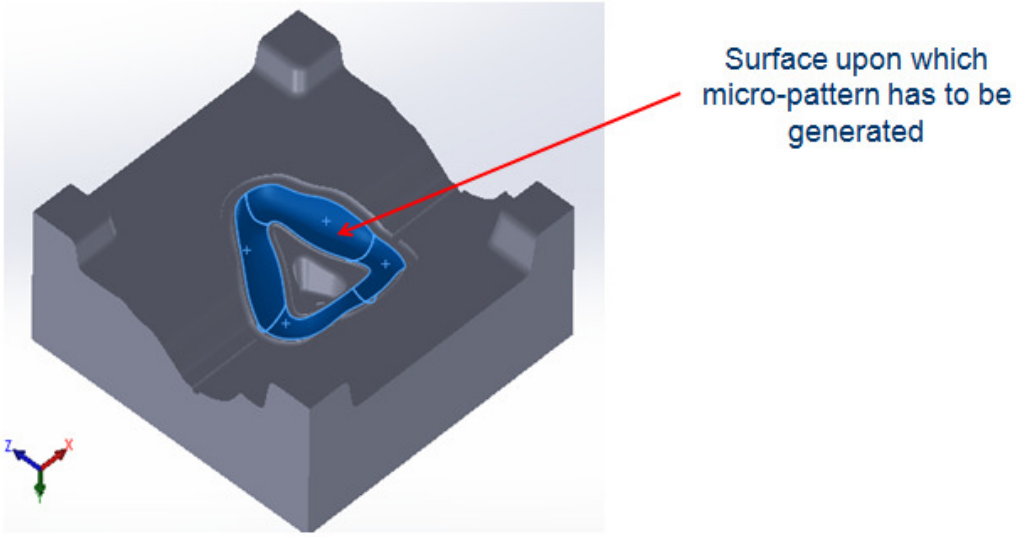


Figure 2.2: Extracting surface for micro-patterning

The highlighted surface in figure 2.2 is extracted for micro-patterning. This figure shows that out of the entire model, only the surface highlighted is the area of interest for applying a micro-pattern. Therefore only the highlighted surface is extracted for tessellation. As a result a tessellated surface would be obtained.

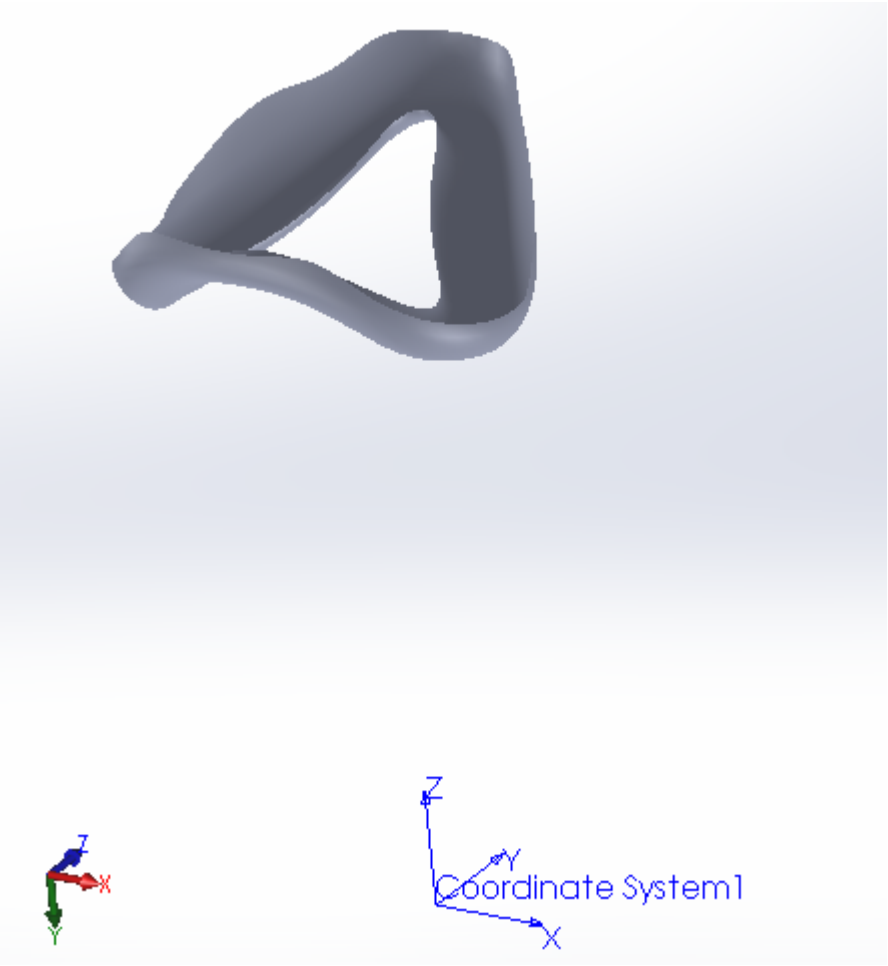


Figure 2.3: Assigning User Co-ordinate System

A user desired coordinate system is set for the part as shown in figure 2.3. Thus only this surface will be converted into the STL or VRML format. The STL file has an option for the user to set export settings for adjusting how fine or how accurate an approximation to the surface the tessellated surface is desired to be.

2.1 Reading STL format

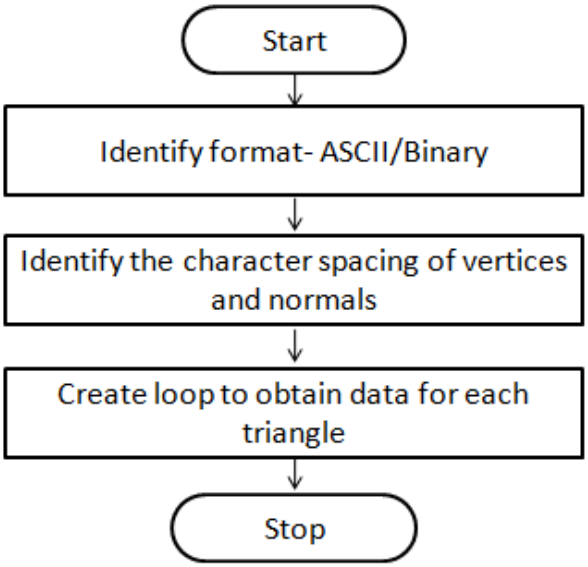


Figure 2.4: Flowchart for reading STL

The flowchart shown in figure 2.4 illustrates the basic steps needed to read the STL file. The first step is to identify whether it is in an ASCII or Binary format. The STL file is imported as a string in MATLAB 2014a. The string is split using delimiters. Every ASCII file has the word 'facet' in it. A program loops through the records to search for the string 'facet'. If it results in a match, then the file is considered to be in the ASCII format, otherwise it is in the Binary format.

The STL file contains data for vertices and normals for every triangle. Thus to read STL, it would be easy to identify where data of a triangle ends and another triangle begins. Then each triangle would have same format of data of vertices and normals in it. Thus it is a repetition of the same logic to read the vertex and normals from each triangles

for all the triangles in the STL file. For designing a code it would be just a loop with varying the triangle number and the same algorithm inside the loop for reading data from each triangle. Variables have to be defined a code to repeat the iterative process. Consider a variable 'loop' as the triangle number in an STL file. Thus, a loop being equal to 10 corresponds to the data for the 10th triangle in the STL file. Consider 'i' as a variable representing the vertex number for a triangle which varies from 1 to 3 in every triangle. Thus values of 1,2 and 3 for 'i' represent the three vertices of each triangle.. Consider 'j' as the variable ranging from 1 to 3 which represents coordinates for x, y and z values for each triangle.

```
solid plane1
   facet normal 0.000000e+000 1.000000e+000 0.000000e+000
      outer loop
         vertex 2.500000e+001 0.000000e+000 -1.000000e+001
         vertex -2.500000e+001 0.000000e+000 -1.000000e+001
         vertex 2.500000e+001 0.000000e+000 1.000000e+001
      endloop
   endfacet
   facet normal 0.000000e+000 1.000000e+000 0.000000e+000
      outer loop
         vertex 2.500000e+001 0.000000e+000 1.000000e+001
         vertex -2.500000e+001 0.000000e+000 -1.000000e+001
         vertex -2.500000e+001 0.000000e+000 1.000000e+001
endloop
   endfacet
endsolid
```

Figure 2.5: Sample STL file

Refer to the above figure 2.5 for the description of an STL file. To read and store the details of the STL file, it is read as a string and split using delimiters. When the STL file is read, it is stored as a complete string in 1 X 1 matrix. In the next step, it is split using delimiters space and tab. The resulting data will be an matrix of size 1 X Total number of separate entities words obtained in STL. Thus each number or word after the delimiter will be stored in a different cell in the matrix. Thus for a matrix 1 X n, each cell will have a rank from 1 to n. Consider 'C' being equal to rank of cell containing the word 'facet' in the matrix created after splitting the string of the STL file. Thus in the above example, the rank of word 'facet', first found in the matrix is three. The words 'solid' and 'plane1' form the first two cells of the matrix. The data of interest from the STL are three numbers representing $\hat{\imath}$, $\hat{\jmath}$ and $\hat{k}$ coordinates for vector representing the normals in 3D and three numbers each for the three vertices representing the x, y, z coordinates.

Observing the figure 2.5 When the STL string is split using delimiters, there is a space delimiter between words 'facet' and 'normal' and another space between the word 'normal' value for i-coordinate for normal. Thus the string after splitting will have 'facet', 'normal' and '0.000000e+000'as consecutive cells in the matrix. Thus, the i-coordinate for normal, being '0.000000e+000'is at two ranks next to facet. There are total 21 entities present in each and every loop of a triangle. The loop of the triangle represents all the entities starting from the 'facet' occurring before normal and the 'endfacet'

$$N(loop, i) = C + 2 + 21 * (loop - 1) + (i - 1)$$

$$V(3 * (loop - 1) + i, j) = C + 8 + 4 * (i - 1) + 21 * (loop - 1) + (j - 1) \qquad (1.1)$$

where,

N = Matrix for storing normals data, with size = number of triangles x 3

V = Matrix for storing x, y, z Coordinates for each vertex with size = 3 * (Number of Triangles - 1) x 3

loop = Triangle Number in STL file,

i = Vertex number for each triangle, varying from 1 to 3

j = Co-ordinate number i.e. 1,2 and 3 representing x, y, z coordinates.

C = rank of 'facet' in the string of STL after splitting.

Therefore, in the string split using delimiters the position of the data for the normals and vertices for identifying which cell in the matrix it belongs, has been parameterized. The rank of the cell representing the normals data and the vertices data in the matrix is a function in terms of variables loop, i, j and C as mentioned in the equation 1.1 above

Similarly, for the Binary format, the position of normals data and vertices data is a function in terms of variables loop, i, j and constant is C'.

$$N(loop, i) = 22 + 25 * (loop - 1) + (i - 1)$$

$$V(3 * (loop - 1) + i, j) = 22 + 25 * (loop - 1) + 3 * (i - 1) + (j - 1) \quad (1.2)$$

where,

N= Matrix for storing normals data, with size = number of triangles * 3

V = Matrix for storing x, y, z Coordinates for each vertex with size

= 3 * (Number of Triangles - 1) x 3

loop = Triangle Number in STL file,

i = Vertex number for each triangle, varying from 1 to 3

j = Co-ordinate number i.e. 1,2 and 3 representing x, y, z coordinates.

2.2 Reading VRML formatted files

```
#VRML V1.0 ascii
Separator {
MaterialBinding {
value OVERALL
}
Material {
ambientColor [
 0.792157 0.819608 0.933333
]
diffuseColor [
 0.792157 0.819608 0.933333
]
emissiveColor [
 0.000000 0.000000 0.000000
]
specularColor [
 0.396078 0.409804 0.466667
]
shininess [
 0.400000
]
transparency [
 0.000000
]
}
Coordinate3 {
point [
 -25.000000 0.000000 -10.000000, -25.000000 0.000000 10.000000,
25.000000 0.000000 -10.000000, 25.000000 0.000000 10.000000
]
}
IndexedFaceSet {
coordIndex [
 2, 0, 3, -1, 3, 0, 1, -1
]
```

Figure 2.6: Sample VRML file

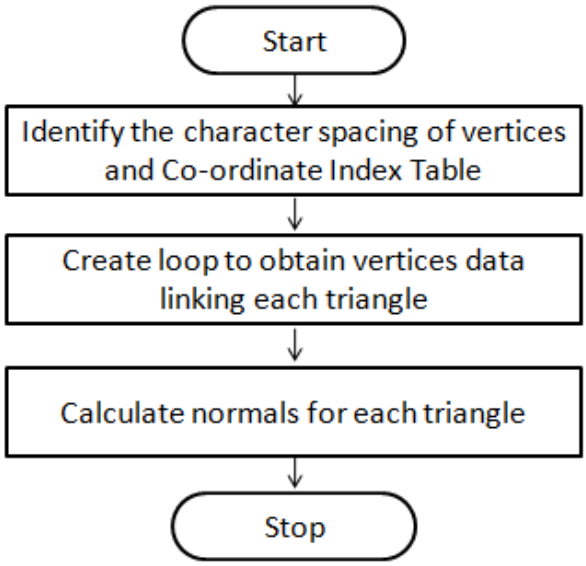


Figure 2.7: Flowchart for reading VRML format

For a VRML format, instead of having repeated data of vertices for each and every triangle, a matrix represents all vertex numbers and their coordinates. Another matrix represents the Coordinate index linking triangles to vertices as explained in section 1.4.

To read the VRML formatted file, a similar logic to that of reading an STL format file is applied. The difference lays in the linking of triangles with the vertex number. Also, as explained in section 1.4, VRML does not contain normals data, but such a direction can be obtained by computing the cross product of two ordered vectors according to the right hand rule.

The first x-coordinate of the first point in the VRML file from the figure 2.6, was found to be at 50^{th} rank of matrix, formed when the string containing VRML file is split

by delimiters. After the first x coordinate, the y and the z coordinates are placed in the VRML file with a space between. The same x, y and z coordinate for the second point is placed with a comma ending at z coordinate of the first point. On observing from the figure 2.6 the data representing the coordinates of points ends with a character ']'. Thus searching the character ']', the end of the set containing the coordinates can be found. Constant 'C1' was defined as ,

$$C1 = rank\ of\ character\ ']'\ after\ first\ 50\ entities\ in\ the\ split\ string\ of\ VRML - 1$$

As the first x-coordinate is placed in the cell of 50th rank in the matrix of split string and the C1 represents the rank of the cell, the number of coordinates is the number of cells between them. Adding further the vertices is one third of the number of cells obtained with coordinates data.

$$Total\ Number\ of\ Vertices = (C1 - 49)/3 \quad (1.3)$$

Similarly from the end of the matrix of the coordinates the character ']' is searched again to obtain the end of the Coordinate Index matrix. The coordinate matrix is the set of coordinates of the points and the Coordinate Index matrix has data for each triangle referring to points from the Coordinate matrix. Thus the searching of ']' here will figure out where the Coordinate Index matrix ends in the matrix for split string data of the VRML.

$$C2 = rank\ of\ character\ ']'\ after\ the\ Coordinates\ Matrix\ in\ the\ split\ string\ of\ VRML - 1 \quad (1.4)$$

From the figure 2.6, the first triangle data in the coordinate index matrix is ' 2, 0, 3, -1,' when the string is split the comma is a delimiter. Thus there are four entities in the set here. '}', 'IndexedFaceSet', '{', 'coordIndex' and '[' are the five cells between the ']' ending for the coodinate matrix and the first point data from the coordinate matrix which is '2'. Thus the number of cells between both the ']' will be difference between the C2 and C1 and then subtracting (5+1). If a cell rank is 1 and another cell rank is 3, then there exists only 1 cell between them. But the difference between 3 and 1 is 2. Thus 1 is subtracted more.

Total number of triangles obtained,

$$Tritot = (C2 - C1 - 6)/4 \quad (1.5)$$

Normals in the VRML can be obtained by following formula

$$Normal = \frac{\overrightarrow{AB}\ X\ \overrightarrow{BC}}{|AB|\ X\ |BC|} \quad (1.6)$$

where,

$\overrightarrow{AB}\ X\ \overrightarrow{BC}$ are the vectors representing the sides AB & BC of Δ ABC,

|AB| is the magnitude of vector $\overrightarrow{AB}$

|BC| is the magnitude of the vector$\overrightarrow{BC}$

.

$$\overrightarrow{BC} = \vec{C} - \vec{B}$$

$$\overrightarrow{AB} = \vec{B} - \vec{A}$$

where,

$\vec{A}, \vec{B}, and\ \vec{C}$ are the position vectors of the vertices A, B and C of Δ ABC

2.3 Direct Interpolation Approach

2.3.1 Methodology

After processing the input files in STL or VRML formats, the data representing the vertices and normals for each triangle is stored in a matrix. Each triangle has a number associated with it. Three vertices and three direction vectors of normals are also associated with each triangle. Thus, the vertices or normals of any triangle can be accessed using the triangle number.

This approach enables the treatment of each and every triangle as a separate entity to obtain the regularly positioned points sought. The idea of this approach is to have an interpolation of points done on the interior of the triangle at equal intervals. A linear interpolation would be done with a uniform distance 'd'. This process would be repeated for each and every triangle from the STL/VRML file

For the interpolation of points for every triangle of the geometry file, two edges of the triangle are selected. From the common vertex of both edges of the triangle, points are placed on the longest edge at a distance 'd' from the previous point. Corresponding points on the other edge are obtained by drawing a line parallel to the third edge. This is followed by the interpolation of points on each of the parallel lines obtained. This interpolation is done with the same increment 'd'. Thus in a triangle, points are plotted at uniform distance in at least one direction.

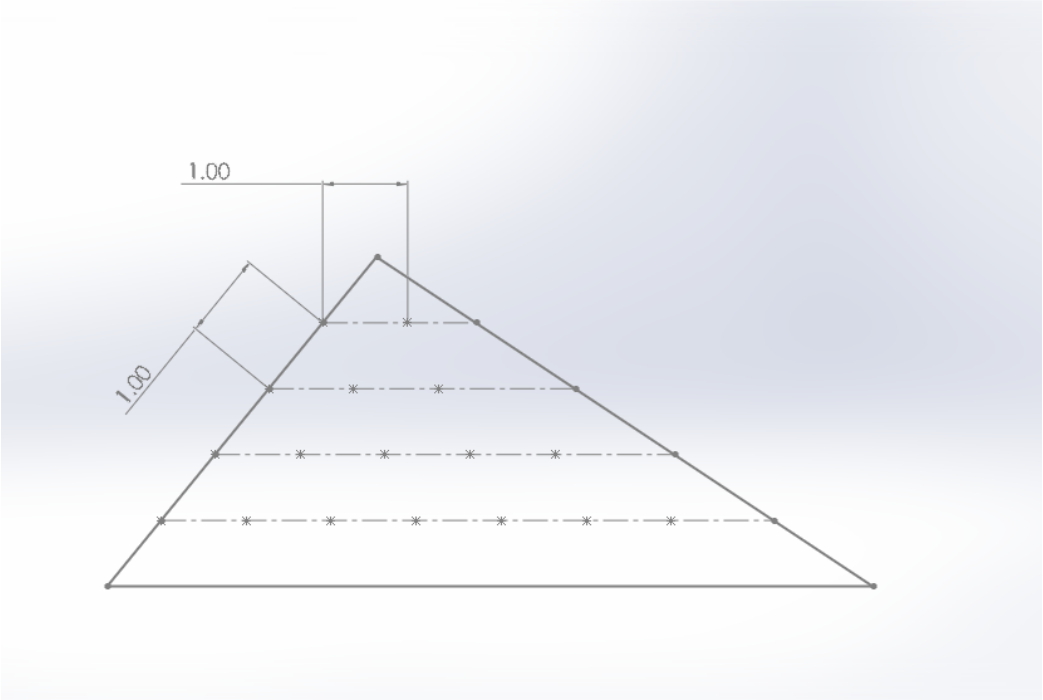


Figure 2.8: Direct Interpolation Approach with starting distance 'd'

While interpolating points on the line, all the points are placed at uniform distances from the previous vertex as shown in figure 2.8. A problem does occur if the first point is also placed at distance 'd' from the edge. The neighboring triangle will also have a point placed at distance 'd' from the edge. Thus the distance between the two points of the neighboring triangles would be '2d' instead of 'd' unless the point on the edge is also considered. Instead, the first point is taken at half of the pitch i.e. 'd/2' from the edge. This maintains an approximate distance of 'd' between two points of neighboring triangles on the same plane. The reason for this is that the normal on each triangle is defined over the whole triangle, and thus, on the edge, there would be two normals, and identifying which is the appropriate one is problematic.

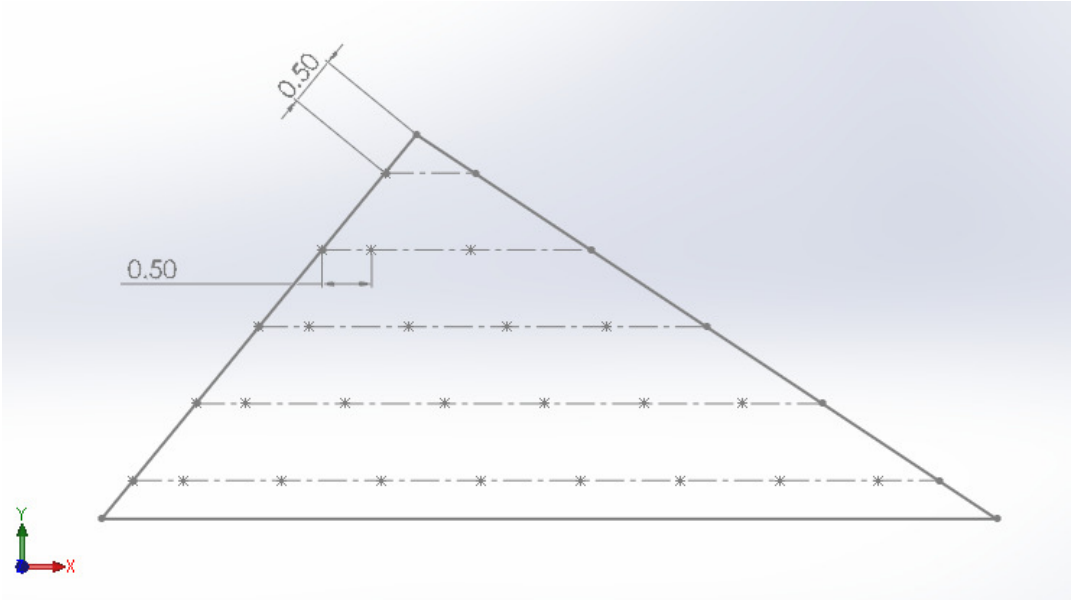


Figure 2.9: Direct Interpolation Approach with starting distance as 'd/2'

Furthermore, for the same triangle, introducing a distance of 'd/2' instead of 'd' at the first point on each plane increases the number of pattern points from 13 to 20 in this specific case when comparing figures 2.9 with 2.8. The distance at the end of the interpolation is also maintained by same distance 'd/2' if possible. This ensures that the first and the last points on any line are at d/2 distance from the edge of the triangle. Therefore, a patterning distance of 'd' can be targeted between points of neighboring triangles. This process is repeated for each and every triangle. The result is a set of points representing the pattern. Each triangle has a specified number and its normal associated with it. As a result, each point in this triangle has the same normal which is identified by its triangle number.

2.3.2 Flowchart for Direct Interpolation Approach

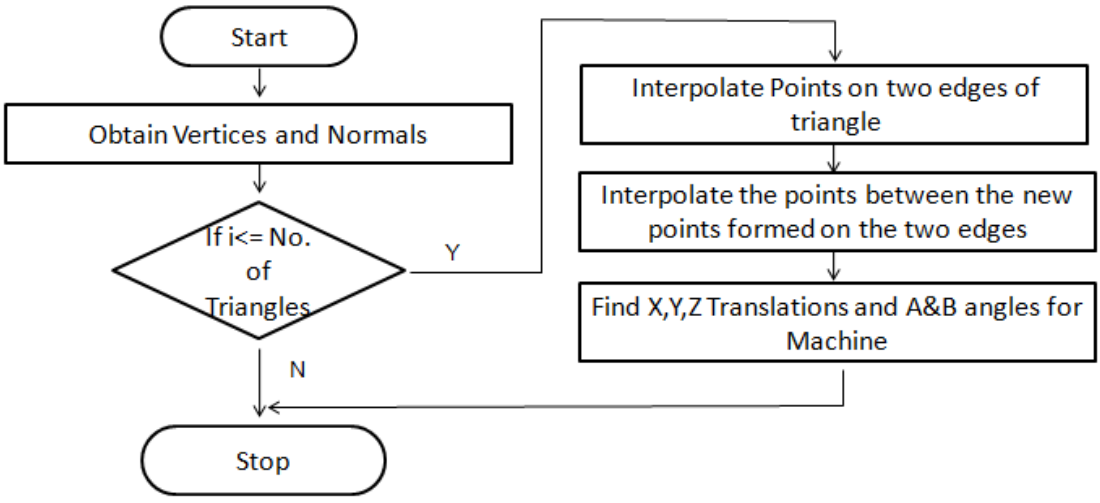


Figure 2.10 Flowchart for Direct Interpolation Approach

The flowchart shown in figure 2.10 explains the algorithm for Direct Interpolation approach for creating a micro-pattern on any arbitrary surface. This is a general flowchart and involves various steps obtaining vertices and normals data from a tessellated surface, followed by interpolation of points on the edge of triangles and within the triangles for micro-patterning. Further obtaining X, Y, Z translations and A & B angles for the 5-axis are also obtained in this algorithm.

2.3.3 Testing

Figures 2.11 and 2.12 show two views of the points generated using this first algorithm and applied to figure 2.3. The figure clearly shows some areas without any points, and some areas with a much higher density of points. This clearly shows that the approach used falls short in producing equally spaced points on an arbitrary surface. Note that the geometry considered was tessellated into a VRML format. This resulted in 4706 triangles.

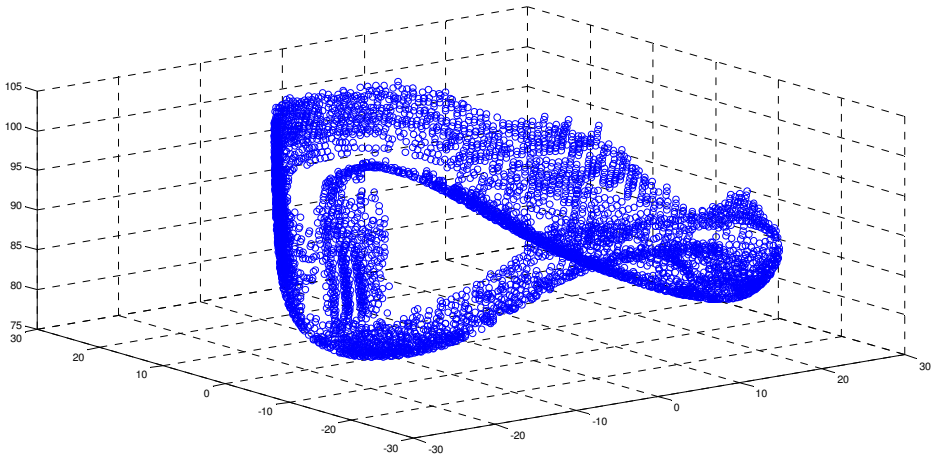


Figure 2.11: 3D plot of obtained points by micro-patterning Direct Interpolation Approach

The Figure 2.11 contains all the patterning points, which are displayed on a 3D plot. Time required for this execution was 79.409 s

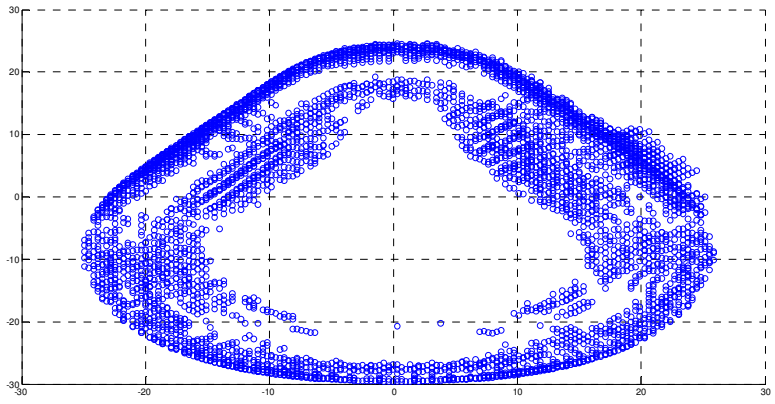


Figure 2.12: 2D -XY plot of obtained points by micro-patterning Direct Interpolation Approach

The total number of micro-patterning points obtained is 5280. From figure 2.11 and figure 2.12 several areas in the pattern are observed to be empty spaces. These are mostly the regions of higher curvature with finer triangles.

2.3.4 Limitation

Upon examining the figure, several limitations are identified. Studying the STL file shows that there are several triangles where the length of the side itself is less than the pitch 'd' selected. If the length is less than 'd', then the interpolation of points or the setting of points at distance 'd' from each other is not possible. This results in a triangle with no points inside it. Typically, triangles are finer in the regions of high curvature, where there is an appreciable change in the normal to the surface. Thus this problem of triangles with no points inside them will occur at critical regions of the geometry. This explains the existence of several blank regions in the figures.

This will therefore result in non-uniform patterning, not a desired result. These limitations lead us to attempt a different approach to generate the patterns. This is discussed next.

2.4 Slicing Approach

2.4.1 Methodology

Instead of handling the patterning one triangle at a time, this approach deals with slicing of the overall geometry with parallel planes. The STL/VRML file consists of triangles forming the complete geometry to pattern. A normal for the plane is chosen. The idea is to offset the plane and obtain intersections of each offset plane with the triangles forming the object to pattern. A starting point is chosen as the lowest point in the normal to the plane direction. Then the first plane is chosen at a distance of 'd/2' from that point. This distance is chosen to have more number of design points. This is a variable and can be changed according to the requirement.

The intersection of any plane and a triangle can have four cases. The intersection can either be a null set, a point, a line or an entire triangle. For few cases when there may be an entire triangle parallel to the plane and therefore an intersection or the side of a triangle can be aligned with a plane and be considered as the intersection. For this approach if an intersection is a point, it will not be considered in the patterning, as the same point will be common with the intersection formed with another triangle. If the point is not common with any triangle then it is just a sharp corner of the part and there cannot be a feature on it. There also exists number of cases where the intersections area complete triangle. The slicing algorithm will only find intersections as a line segment. So in such cases, these triangles are noted and another slicing on these triangles can be done using different slicing direction. this approach will not create any intersections by slicing. Points are interpolated on the intersecting line. Intersections of the plane are checked with

every triangle representing the object under consideration. Then, the plane is shifted in the normal direction by the uniform patterning distance. The intersections for the next plane are checked and new points are formed. This process is carried out until the plane has coordinate in the slicing direction higher than the highest coordinate of the points of intersections formed.

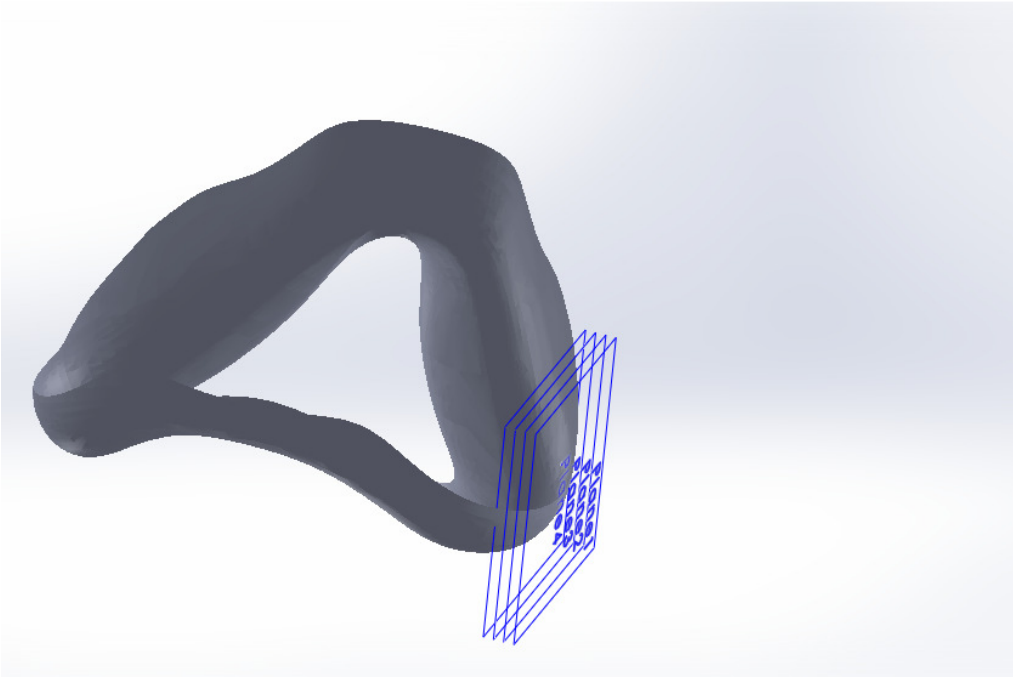


Figure 2.13: Demonstration of slicing the geometry by plane offsets

A demonstration of slicing planes is shown in figure 2.13. The default slicing direction for the geometry is the X direction. The user is prompted in advanced options to switch the slicing direction to another axis of the co-ordinate system defined by the user if so desired.

This approach is different from the earlier direct interpolation approach. In the earlier approach, the points were interpolated within each and every triangle. In this approach the geometry is sliced by a plane. Thus the line intersections formed are taken into consideration.

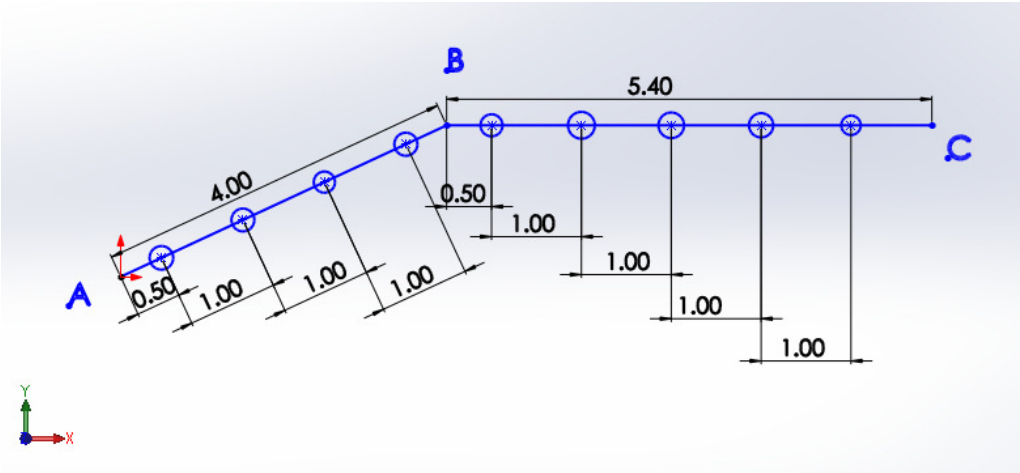


Figure 2.14 Interpolation-Slicing Approach

Figure 2.14 explains how the points are interpolated in the slicing approach. There will exist many line intersections when the plane is intersected with the geometry. This figure explains the slicing algorithm observing two line intersections which are formed from the intersection of the plane with two different triangles. Line segments AB and BC are formed from the two triangles. This model has slicing direction in the Z direction.

When obtaining intersections of the plane with the triangle, each side is checked for intersection.

$$t1 = \frac{z - z1}{z2 - z1}$$

$$t2 = \frac{z - z2}{z3 - z2}$$

$$t3 = \frac{z - z3}{z1 - z3} \tag{2.8}$$

where,

t1: Intersection checking parameter for side 1 of the triangle formed by vertex 1 and vertex 2

t2: Intersection checking parameter for side 2 of the triangle formed by vertex 2 and vertex 3

t3: Intersection checking parameter for side 3 of the triangle formed by vertex 3 and vertex 1

z: z-coordinate for slicing plane

z1: z-coordinate for first vertex of triangle

z2: z-coordinate for second vertex of triangle

z3: z-coordinate for point third vertex of triangle

The order of the vertices are according to the order defined in the tessellated format.

Case 1:

$$If\ 0 < t1 < 1\ \&\ 0 < t2 < 1$$

If t1 is greater than zero it implies that 'point 1 to point z' and 'point 1 to point 2' have the same sign positive or negative. Point z represents the point at the intersection of the plane with the line. This means that both vectors are point to same direction. This can result in two scenarios. The first scenario is point z lies between point 1 and point 2, which means it lies on the line segment from point 1 to 2. Second scenario is point z lies ahead of point 2 in the direction of vector representing point 1 to point 2. Which means the point z lies on the extrapolated line segment of 1 to 2, implying that it does not lie on line segment 1 to 2.

Now it is known that point 1 to point z and point 1 to point 2 are the same directions from 0 < t1. Further adding t1 < 1, it implies that the magnitude of vector from point 1 to point z is less than that of vector from point 1 to point 2. Therefore point z lies between point 1 and point 2 representing the side 1 of the triangle. Similarly if 0 < t2 <1, then the point z lies on side 2 representing point 2 to 3.

Thus, the plane has intersections with side 1 and side 2 of the triangle, where side 1 represents the side of the triangle joining the point 1 and point 2 and side 2 represents the side joining the point 2 and point 3.

Case 2:

$$If\ 0 < t2 < 1\ \&\ 0 < t3 < 1$$

Then, the plane has intersection with side 2 and side 3 of the triangle, where side 2 represents the side of the triangle joining the point 2 and point 3 and side 3 represents the side joining the point 3 and point 1.

Case 2:

$$If\ 0 < t3 < 1\ \&\ 0 < t1 < 1$$

Then, the plane has intersection with side 3 and side 1 of the triangle, where side 3 represents the side of the triangle joining the point 3 and point 1 and side 1 represents the side joining the point 1 and point 2.

Once the two sides of the triangle on which the plane intersects are found out, the x and y coordinates of the point of intersection is found by the following equation which represents the equation of a line in 3D.

$$\frac{x-x1}{x2-x1} = \frac{y-y1}{y2-y1} = \frac{z-z1}{z2-z1} \tag{2.9}$$

2.4.2 Verification of the Intersection Logic

A study is done for different cases to verify the logic for the intersection of the planes. Let us assume the slicing direction is in the Z direction. z1 and z2 will represent the z-coordinates of the point 1 and 2.

Case 1: z1 < z < z2

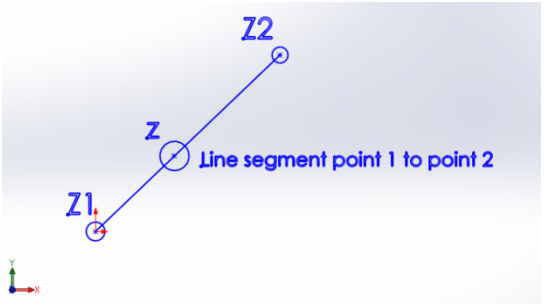


Figure 2.15 Intersection Case 1: z1 < z < z2

Figure 2.15 shows a line segment from point 1 to point 2 with z-coordinates z1 and z2. The intersection point of the slicing plane and the line is the point with 'z' as the z-coordinate. The value z is lies between z1 and z2. Thus the point of intersection lies on segment point 1 to point 2.

$$z > z1$$

$$z2 > z1$$

$$z2 - z1 > z - z1$$

$$Thus,\ 0 < t1 = \frac{z-z1}{z2-z1} < 1 \qquad (2.10)$$

Thus the intersection is found in this case, where the point formed by intersection of the line connecting point 1 to point 2 and the slicing plane lies between the two points of the edge of the triangle.

Case 2: z1 < z2 < z

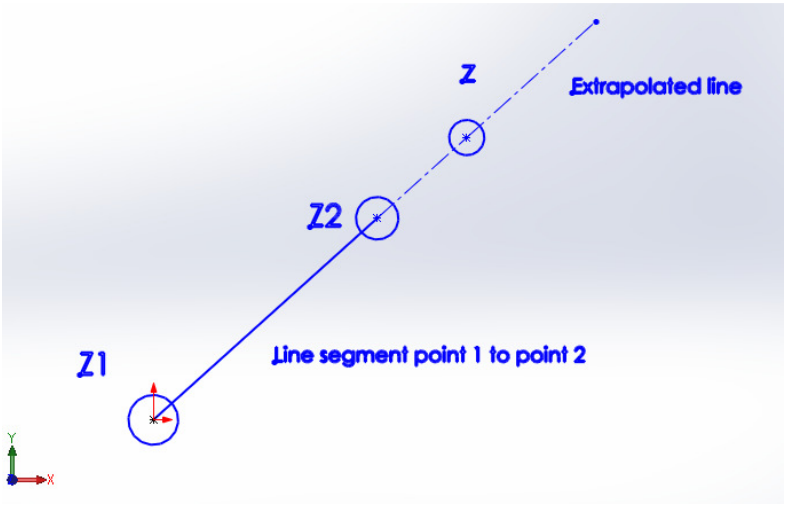


Figure 2.16 Intersection Case 1: z1 < z2 < z

Figure 2.16 shows a line segment from point 1 to point 2 with z-coordinates z1 and z2. The intersection point of the slicing plane and the line is the point with 'z' as the z-coordinate. The value z is greater than z2 and z1. Thus the point of intersection lies on segment point 1 to point 2, extrapolated from the point 2.

$$z > z1$$

$$z2 > z1$$

$$z2 - z1 < z - z1$$

$$\text{Therefore, } t1 = \frac{z-z1}{z2-z1} > 1 \qquad (2.11)$$

Thus no intersection is found. This is correct as where the point formed by intersection of the line connecting point 1 to point 2 and the slicing plane lies is outside the line segment.

Case 3: z < z1< z2

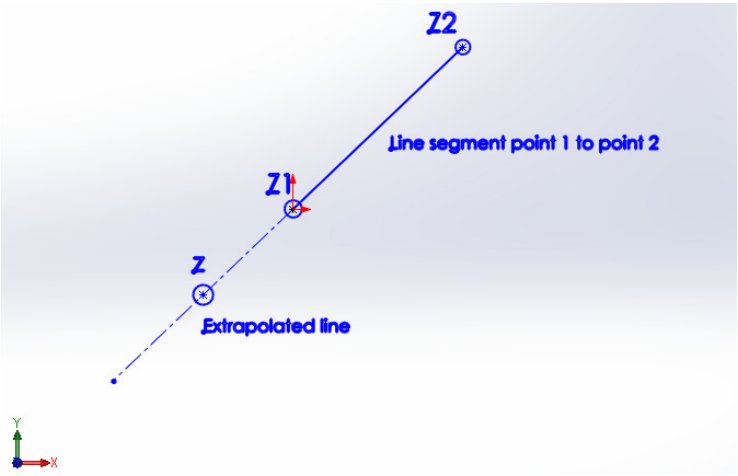


Figure 2.17 Intersection Case 1: z < z1< z2

Figure 2.17 shows a line segment from point 1 to point 2 with z-coordinates z1 and z2. The intersection point of the slicing plane and the line is the point with 'z' as the z-coordinate. The value z is less than z1 and z2. Thus the point of intersection lies on the segment point 1 to point 2, extrapolated from the point 1.

$$z < z1$$

$$z2 > z1$$

$$z2 - z1 > z - z1$$

$$Thus,\ t1 = \frac{z-z1}{z2-z1} < 0 \qquad (2.12)$$

Thus no intersection is found. In this case the slicing plane is below both the points of the edge in Z direction. Thus, it will have no point for intersection.

2.4.3 Interpolating the points

Consider that the intersecting segment is formed between side 1 and side 2 of the triangle. The coordinates of the intersection points are obtained by the previous formulas. Thus the line segment formed by intersection represents a line segment joining the intersection point on side 1 and intersection point on side 2 of the triangle. Let P & Q be the intersection points on side 1 and side 2 respectively.

Therefore, the equation of the line is given by:

$$\frac{x-x_p}{x_q-x_p} = \frac{y-y_p}{y_q-y_p} = \frac{z-z_p}{z_q-z_p} \tag{2.13}$$

The following equation determines the next interpolating point of the line segment starting from point P.

$$\overrightarrow{R1} = \overrightarrow{P} + \left[\frac{(\overrightarrow{Q} - \overrightarrow{P})}{|\overrightarrow{PQ}|}\right] * d/2 \tag{2.14}$$

where,

d: Patterning distance

$\overrightarrow{R}$: Position vector for Point R

$\overrightarrow{P}$: Position vector for Point P

$\overrightarrow{Q}$: Position vector for Point Q

From the first point onwards, the points will be interpolated with a patterning distance of 'd'.

$$\overrightarrow{R2} = \overrightarrow{P} + \left[\frac{(\overrightarrow{Q} - \overrightarrow{P})}{|\overrightarrow{PQ}|}\right] * d \tag{2.13}$$

This process of adding interpolation points is continued until the distance between the last patterning point obtained and the point Q on side 2 is less than 'd/2'. The next connected intersection will have a point on 'd/2' distance from the starting point and an approximate distance of 'd' can be aimed.

2.4.4 Flowchart for Slicing Approach

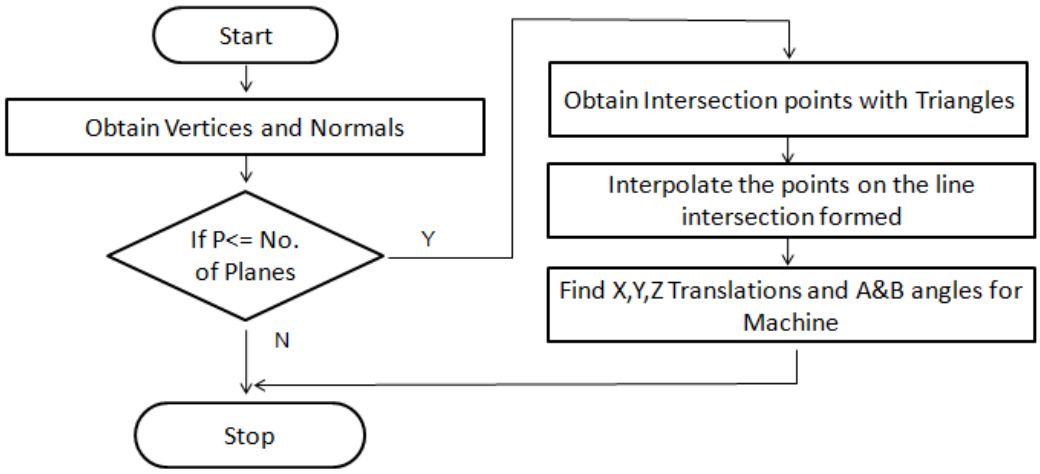


Figure 2.18 Flowchart for Slicing Approach

The flowchart shown in figure 2.18 explains the Slicing algorithm for creating a micro-pattern on any arbitrary surface. This is a general flowchart and involves various steps such as obtaining vertices and normals data from a tessellated surface, slicing the tessellated surface with he a plane and with offsets to the plane at regular intervals. It is followed by interpolation of points on the intersections of the planes and triangles obtained as line segments for micro-patterning. Further, obtaining X, Y, Z translations and A & B angles for the 5-axis are also obtained in this algorithm.

2.4.5 Testing of Slicing Approach

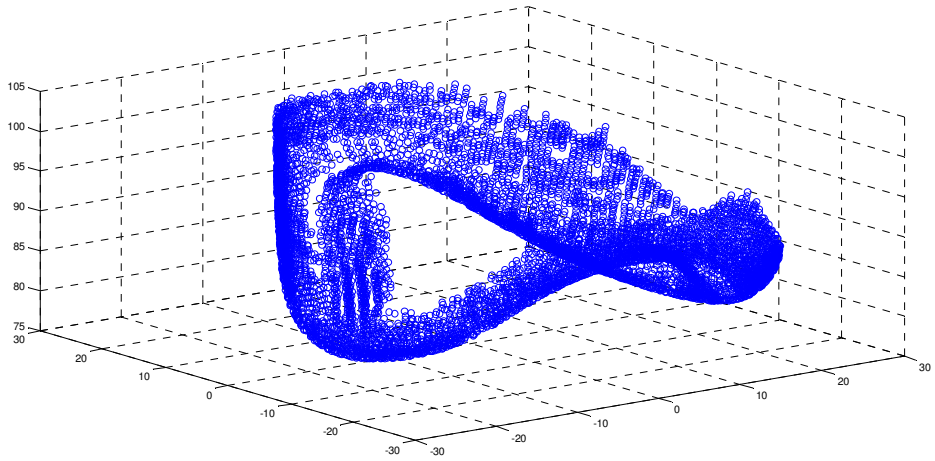


Figure 2.19: 3D plot of obtained points by micro-patterning Slicing Approach

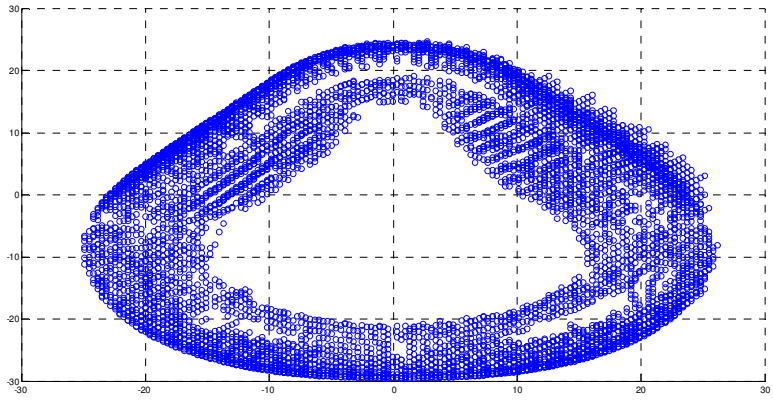


Figure 2.20: 2D-XY plot of obtained points by micro-patterning Slicing Approach

The same geometry from figure 2.3 is considered for testing of this slicing algorithm. This geometry is considered as it is a non-uniform curved surface. It is an

arbitrary surface with no uniform extrusion in a single direction. The total number of micro-patterning points obtained is 7307.

Figures 2.19 and 2.20 indicate that the points distribution is better in the slicing algorithm when compared with the direct interpolation approach presented earlier. But there are still few empty spaces with a need for improvement in the slicing algorithm for a better distribution of the points.

2.4.6 Limitations

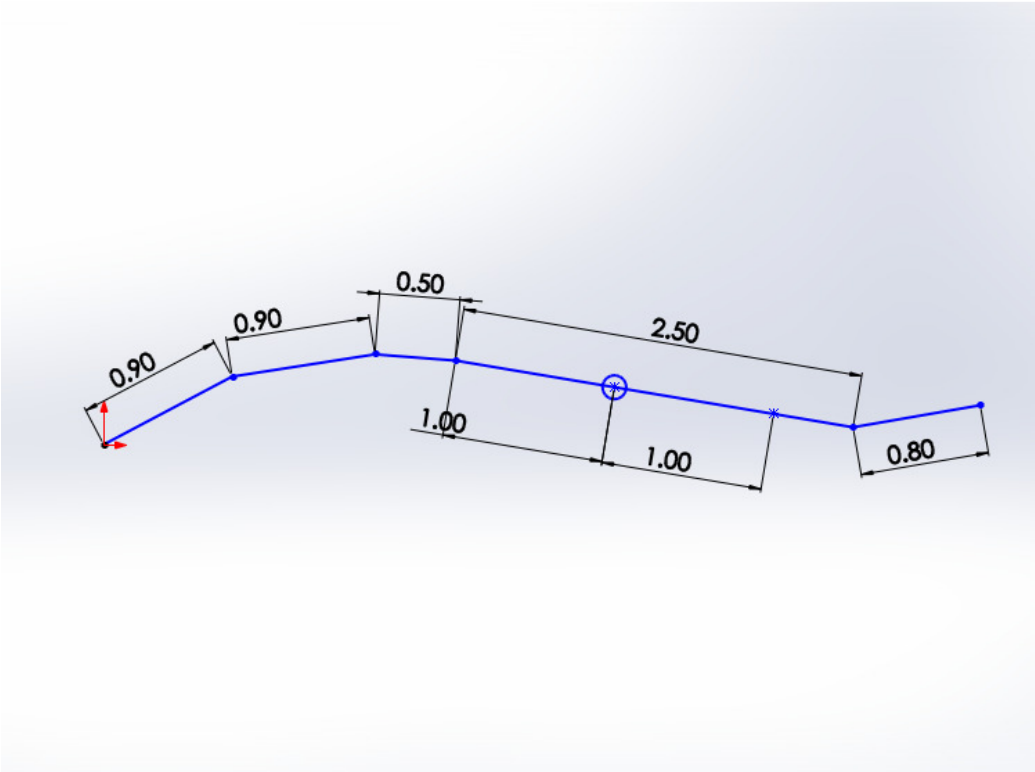


Figure 2.21: Limitation 1 of Slicing Approach

For the five connected line segments shown in figure 2.21 and obtained by finding the intersection of the cutting plane with several adjacent triangles, the total length of the curve is the sum of their individual lengths, which are 0.9, 0.9, 0.5, 2.5 and 0.8. Thus the total length comes out to be 5.6 units. Thus say for a unit distance of patterning over five connected line segments of total length 5.6, only one point is obtained using the algorithm that treats each segment independently. On the other hand, consider the five connected line segments to be a single curve. In a single curve of 5.6 length, 5 points at a unit distance from each other can be plotted over the total interval.

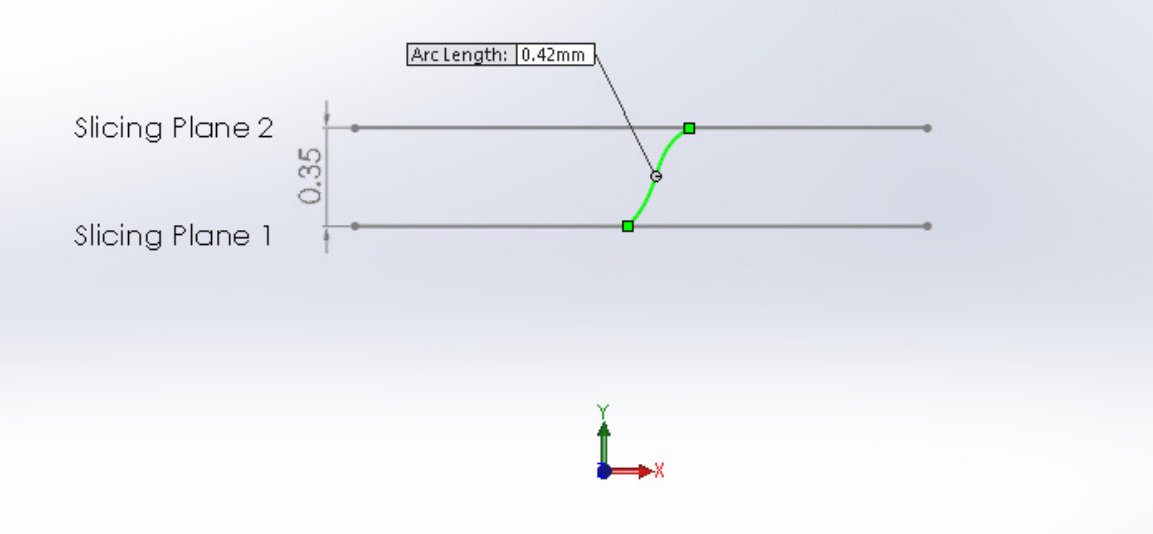


Figure 2.22: Limitation 2 of Slicing Approach

Figure 2.22 shows another limitation in the methodology of the slicing algorithm described. Consider a surface generated by a curve extruded into the direction normal to the sketch. Slicing Planes 1 and 2 are parallel to the ZX plane. Slicing Plane 2 is obtained by offsetting the slicing plane 1 by 0.35 mm. But the arc length within the two planes is

0.42 mm. Thus although the points are patterned with a uniform distance in each slicing plane, there exists an error in the method to offset the slicing plane. This error will be zero for a case where the line is perpendicular to the slicing direction. But it will increase as the angle of the line increases with the slicing direction.

2.5 Slicing-chain Approach

2.5.1 Methodology

In order to remedy the shortcomings of the previous two approaches, a slicing chain approach is implemented. It is an improved algorithm based on the slicing approach. The intersections are obtained similarly for each cutting plane given an offset of patterning distance 'd'. In the prior slicing approach, the points were interpolated on the line intersections for a given plane between the two sides of each intersecting triangle. Several line segments formed from the intersection of the plane with the triangles may have a length which is less than that of the patterning distance 'd'. In such cases, the set of points formed from the interpolation is a null set and regions void of the patterns are created on the surface. Further, there exists several remainders or the line segments formed by intersections after interpolation. Remainders are the distance on the line segment left after patterning of points. It is the distance measured between the last patterning point on the line segment and the endpoint of the line segment. A line segment with length 1 unit and applied a patterning distance of 0.35 on it leaves a remainder of 0.3

units. These occurrences are expected in every intersection and need to be addressed. The next improved algorithm addresses this issue.

The algorithm to obtain the intersections of the plane with the tessellated surface for this approach is the same as in the previous slicing approach. This algorithm differs in the method of interpolating points for patterning on the line segments formed. The idea is to form a chain of line segments. Forming a chain would provide a way to interpolate points on a chain of line segments, similar to interpolating points on any continuous line.

From the slicing algorithm, the intersection points between the plane and the triangles are obtained. Thus, a set of all intersection points for a single plane and all triangles are obtained. To form a chain from the line segments, the common points between the line segments need to be identified. Therefore, a search is carried out to find if any two points in the set of intersection points obtained are equal. As STL is a tessellated surface, though the points are equal may not have exact same value for the coordinate. Thus a tolerance must be assigned to find the points having a difference less than the specified tolerance.

If the tessellated surface is sliced in the z direction, this implies that the z-coordinate must be equal for the obtained intersection points. Thus a check needs to be done on the other two coordinates. The process remains the same even if the slicing direction is different.

$$If\ |y1 - y2| < TOL\ \&|x2 - x1| < TOL,\ Then\ Point\ 1 = Point\ 2 \quad (2.14)$$

where,

Slicing is done in Z-direction,

y1: Y coordinate of Point1

y2: Y coordinate of Point2

x1: X coordinate of Point1

x2: X coordinate of Point2

$TOL\ = 10^{-10} mm$

The patterning is done in microns, which is $10^{-6} mm$. Thus for measuring in microns, the tolerance is $10^{-4} \mu m$. The points can be considered to be same if the condition 2.14 is satisfied.

The intersection points so obtained, represent the line segments of intersection. To apply patterning of the points on these segments, sorting of the of the points with respect to its coordinates is required. The set of intersection points is sorted in one of the two remaining directions, i.e. for slicing in the Z direction, the set of points is sorted in the X direction. For slicing in the Y direction, sorting is done in the Z direction and for slicing in X direction the sorting is done in Y direction. This sorting then determines the direction to go along the line segments for patterning.

The search is started from the point having the lowest coordinate in the sorted direction. The point will also have the reference to which triangle it belongs from the matrix created from the STL file data. Line segments formed as an intersection of the triangles with the plane are considered for patterning of points. Each line segment has two points. The point with lowest coordinate considered will point out to the other point on the same line segment formed as intersection. Thus, this is the first line segment considered for starting the chain. The idea is to have two chains from this segment. Forward chain will start from the lowest point and will search for formation of chains. The chain will break when no point is obtained after searching for points with equal coordinates. Then the chain will again start from the other end of the first line segment. This will be a reverse chain, following the same process.

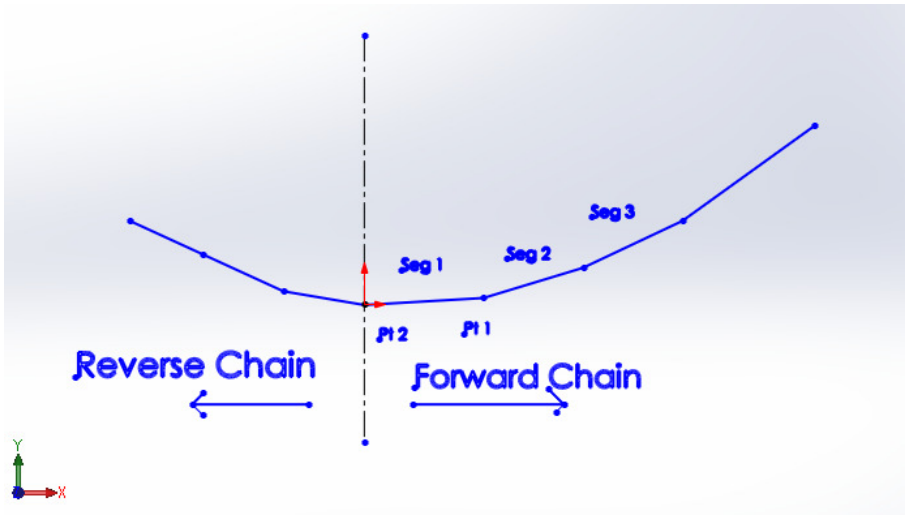


Figure 2.23 Chain Formation

$$If\ Point\ 1\ on\ segment\ 1\ = Point\ 1\ on\ segment\ 2$$

Then the point 2 (point 1 on segment 2) will give a link to the triangle it belongs and the line segment of intersection formed. As the intersection points and line segments so formed are stored in the matrix, obtaining one point, the triangle number is known. For each triangle there exist only two points of intersection. So the second point on the new line segment 2 is determined forming chain between first and second segment adding the segment 2 to the chain with segment 1. Consequently the other point on the second line segment i.e. Point 2 in segment 2 will be the new end point of the chain.

Similarly,

$$If\ Point\ 2\ on\ segment\ 2 = Point\ 1\ on\ segment\ 3,$$

Then segment 3 is added to the chain and the point 2 on segment 3 is the new end point of the chain. The point 2 on segment 3 will be now be the new point for searching the chain formation.

The tessellated surface many have more than one loop of chain formation. The forward chain starting from the 1st point on the first line segment ends when no further points are matched to be the same. It is then followed by the reverse chain starting from the 2nd point on the same first line segment as shown in figure 2.24.

The result is a chain formation done on both ends of the first line segment considered and the chain breaks at both ends. Further, in the set of points of intersection, which is sorted, the next point with lowest coordinate in the slicing direction is considered. The chain formation is again started from this point. This continues until there is no point in the set of intersection points obtained. Also there can be segments with no chain formed, these can be considered as a chain with only one line segment.

Interpolation within a chain is discussed in detail as follows.

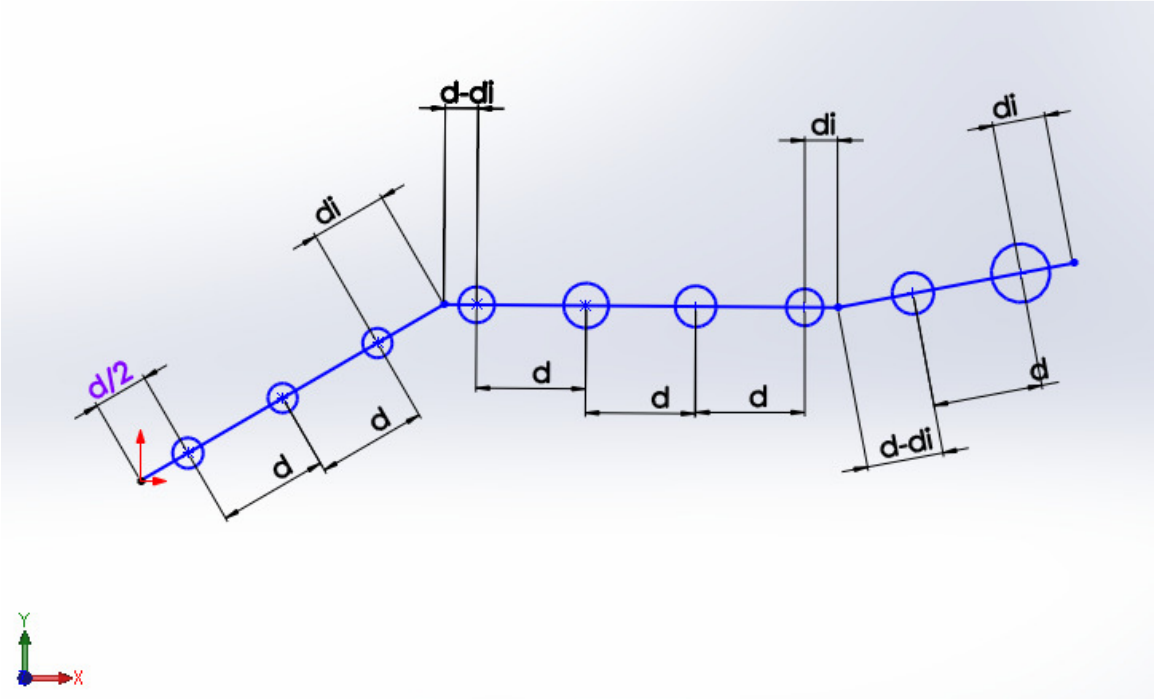


Figure 2.24 Interpolation in Slicing Chain

The figure 2.24 explains the algorithm for the slicing chain approach.

where,

d: Distance of patterning

di: Initial Distance remainder from last segment

The first point on the chain is plotted with a distance of d/2 from the starting point. This is done to have the first point on this intersection plane at a distance of d/2 from the boundary of the geometry. This starting distance can be changed according to requirement.

The general rule to obtain distance for interpolation for the second point on the chain is:

$$Distance\ for\ interpolation\ for\ the\ second\ point = di + d \quad (2.15)$$

where,

di = 0 is the initial value set.

Between a segment PQ, to obtain a point of pattern R

$$\overrightarrow{R} = \overrightarrow{P} + \left[\frac{(\overrightarrow{Q} - \overrightarrow{P})}{|\overrightarrow{PQ}|}\right] * (d - di) \tag{2.16}$$

$$di = 0 \ (set\ to\ zero\ again)$$

where,

d: Patterning distance

di: Initial Distance remainder from last segment (Initial value zero)

$\overrightarrow{R}$: Position vector for Point R

$\overrightarrow{P}$: Position vector for Point P

$\overrightarrow{Q}$: Position vector for Point Q

The interpolation will stop on the segment when the distance between the point of pattern R and Q is less than 'd'.

$$If\ \left|\overrightarrow{Q} - \overrightarrow{R}\right| < d, then\ di = \left|\overrightarrow{Q} - \overrightarrow{R}\right| \tag{2.17}$$

Thus, now the interpolation will carry on to the other connected segment in the chain formed. The same formula for obtaining the point of pattern 'R' will be used. When di is used, it will be set to zero again.

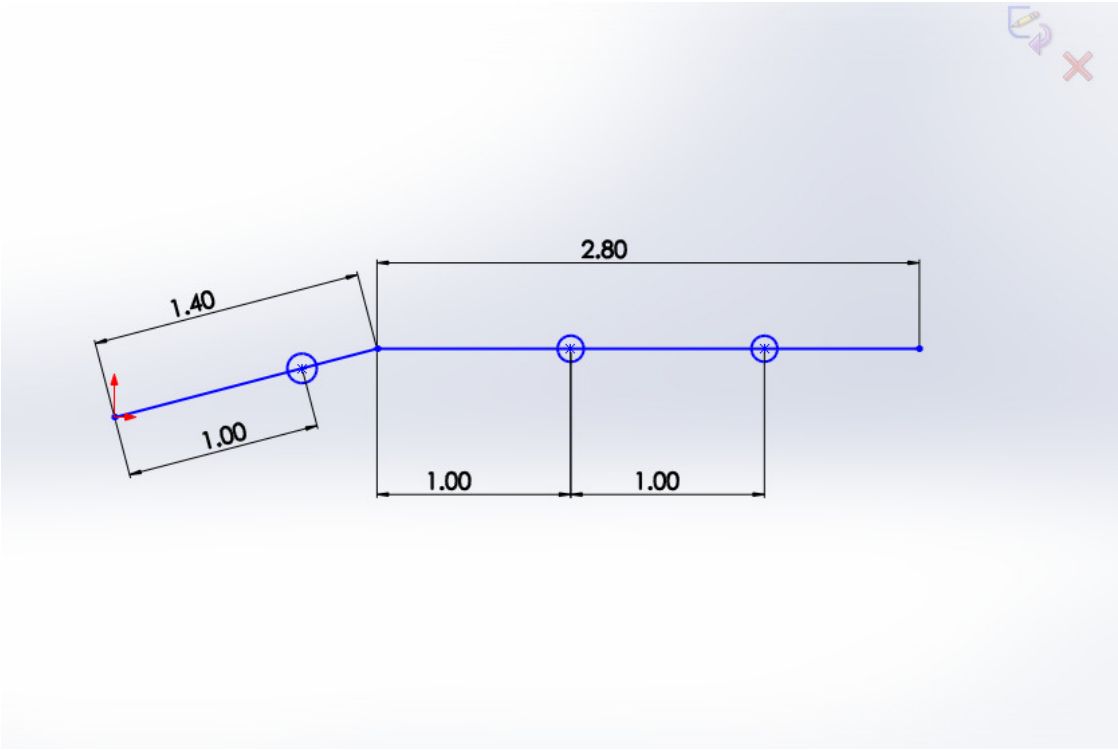


Figure2.25: Before implementation of chain structure for slicing

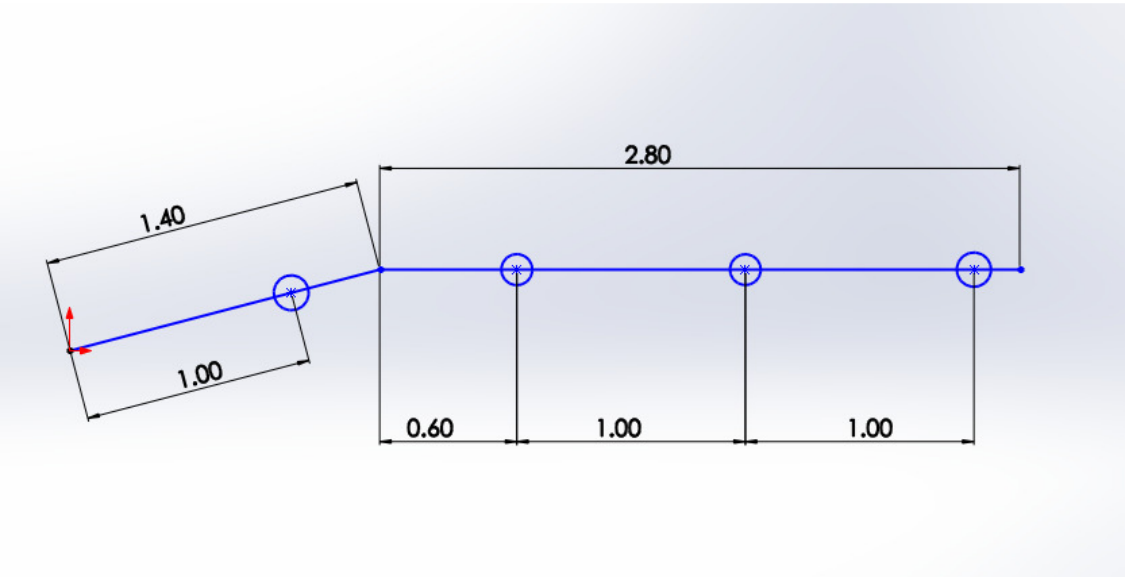


Figure 2.26: After implementation of chain structure for slicing

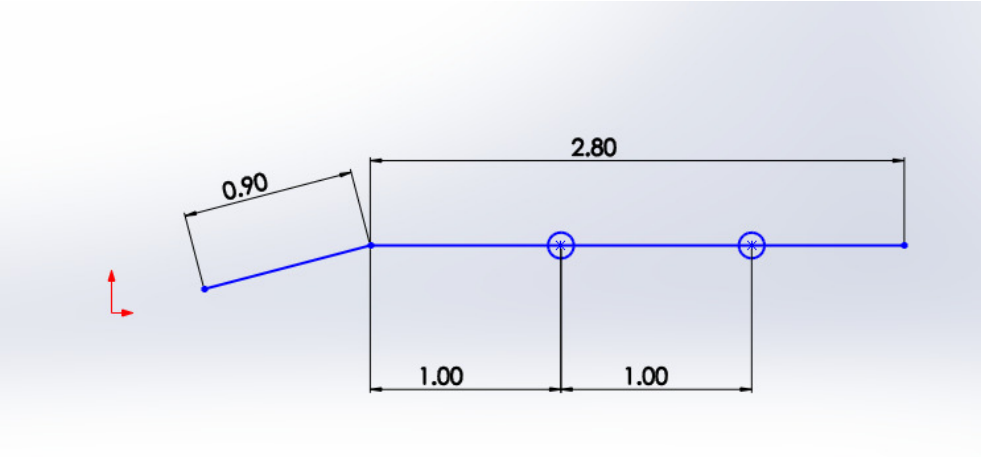


Figure 2.27: Interpolation on shorter segments before implementation of chain structure for slicing

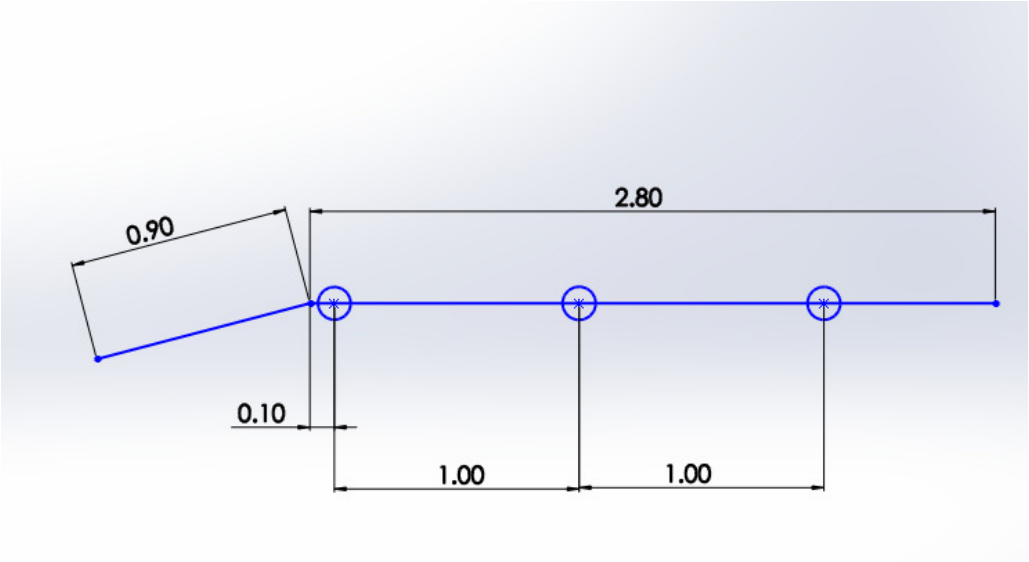


Figure 2.28: Interpolation on shorter segments after implementation of chain structure for slicing

Consider a simple case, refer to figure 2.25. This figure shows the intersection formed by the plane with four adjacent triangles as an example. The following study does

not depend on the number four. Two segments are also sufficient to explain improvement of patterning algorithm in chain slicing approach. This leads to the formation of connected line segments. The figure shows a small part of the total intersections formed by the plane. In this case, for demonstration and ease of understanding, a unit dimension patterning distance is considered. The aim is to study what happens to the when the line segments are longer or shorter than a multiple of the patterning distance 'd'. For instance, in the figure drawn, the first segment formed by the intersection of the cutting plane with the triangulated representation has a length of 1.4 units, but the patterning distance is 1 units. Thus, after the initial point, only one point is obtained on the line segment. If the previous algorithm is used, the next segment is considered, and points are identified at a distance of 1 unit from the beginning. So the distance between the previous point on the last segment and the new generated point is obtained as 1.4. The new slicing chain approach eliminates this problem. Points obtained in figure 2.26 are four, where as in figure 2.27 three points are identified. Figure 2.27 represents the slicing chain approach. Similarly, the improvement in the algorithm can be seen from figure 2.27 and 2.28. This approach eliminates this error as the points are interpolated continuous on a chain of line segments.

2.5.2 Flowchart For Slicing - Chain Approach

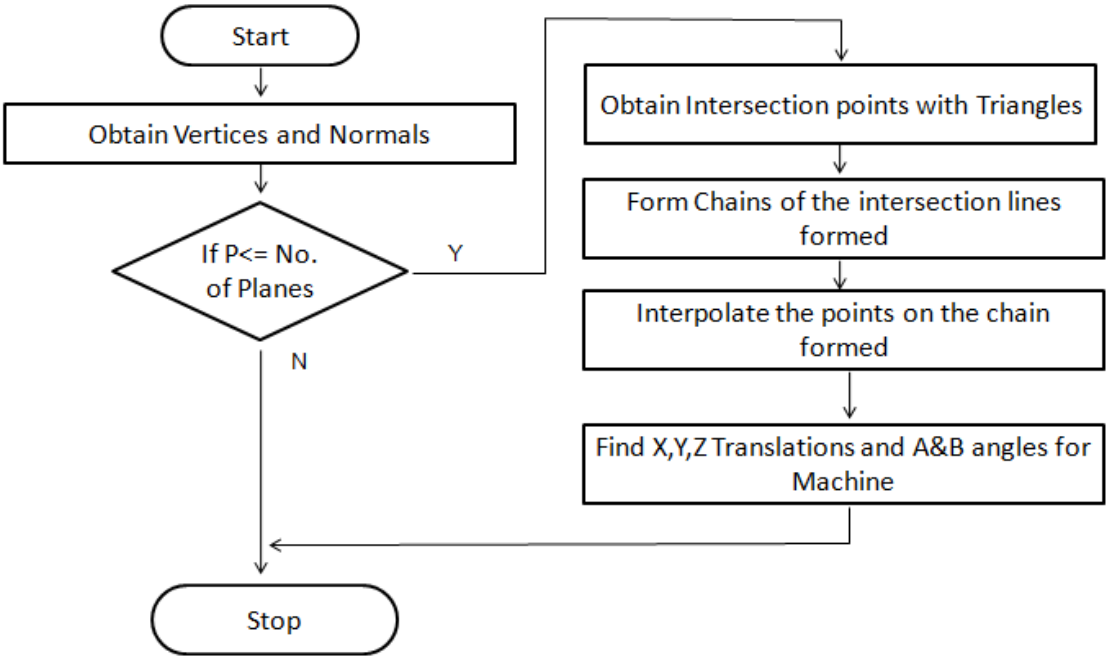


Figure 2.29: Flowchart For Slicing - Chain Approach

The flowchart shown in figure 2.29 explains the algorithm for Slicing-Chain approach for creating a micro-pattern on any arbitrary surface. This is a general flowchart and involves various steps such as obtaining vertices and normals data from a tessellated surface, slicing the tessellated surface with plane and providing the offset to plane. It is then followed by creating intersections by slicing the tessellated surface with a plane. In this approach forming chains of the line segment intersections formed is an important improvement. Further, obtaining X, Y, Z translations and A & B angles for the 5-axis are also obtained in this algorithm.

2.5.3 Testing of Slicing - Chain Approach

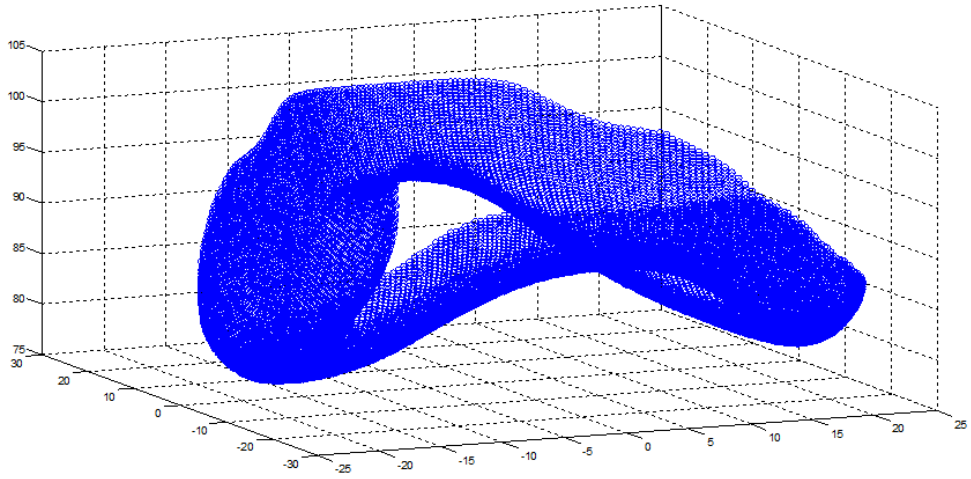


Figure 2.30: 3D plot of obtained points by micro-patterning Slicing-Chain Approach

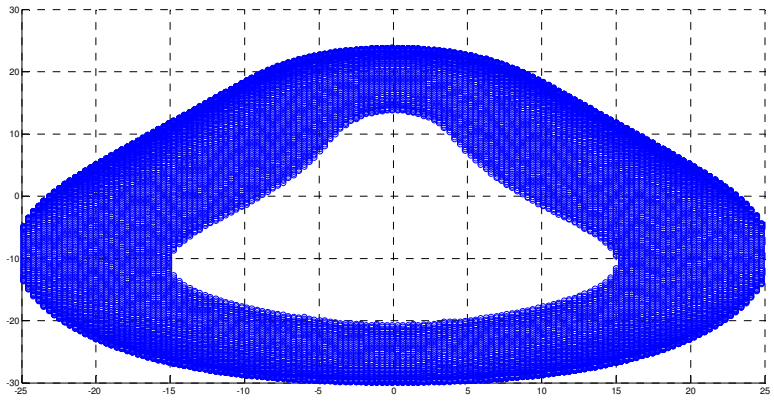


Figure 2.31: 2D-XY plot of obtained points by micro-patterning Slicing-Chain Approach

Note that the geometry considered was tessellated into a STL format with an accuracy of 0.0038881mm. This resulted in 47290 triangles. The capability of the algorithm to handle finer triangles allowed the use of a fine tessellated STL file.

The same geometry from figure 2.3 is considered for testing of this slicing-chain algorithm. This geometry is considered as it is a non-uniform curved surface. It is an arbitrary surface with no uniform extrusion in a single direction.

The number of points generated and shown in figure 2.31 is 17982 micro-patterning points. An approximate number of expected points can be calculated by dividing the surface area of the geometry by the area of a square of side equal to the distance of patterning. Thus the expected number of points come out to be 21628. Thus 83% of the points theoretically obtainable are identified. Time required for the execution was 122.134s.

2.5.4 Limitations

The Slicing-Chain approach is effective for obtaining a micro-pattern of points. The approach has a slight shortcoming in that two segments at some angle from each other will have two points that are not located at a distance equal to the sum of the two line segments on the individual intersecting lines. This will therefore introduce some level of inaccuracy in the distance between the points, although it should be minimal except at very high curvature points. The points are interpolated with desired patterning

distance when interpolation is done within a chain. But giving the plane offset of the patterning distance will introduce a slicing error as mention is section 2.2.4, where the distance along the surface from points interpolated in the consecutive planes may not be accurate. Due to this for a non-uniform surface, the micro-patterning will not be able to capture 100% of the patterning points.

2.5.5 Slicing-chain Approach - Modified

For uniform curved surfaces, the Slicing-chain Approach can be modified to obtain better results. The slicing Direction can be modified along a curve instead of x, y, z directions as done earlier.

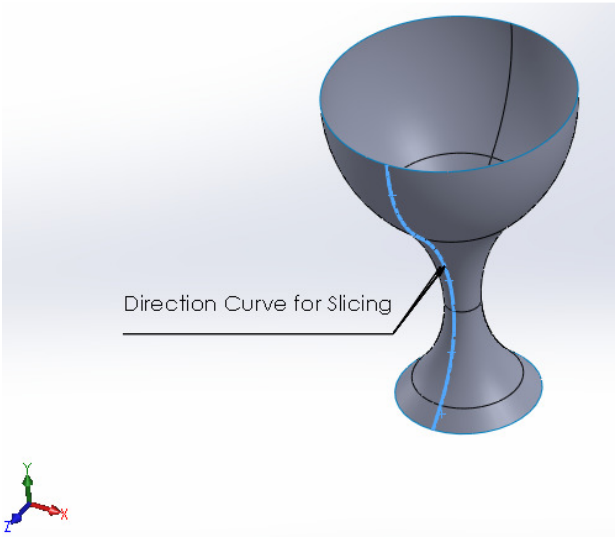


Figure 2.32: Direction Curve for Slicing - Uniform Curved Surface 1

As shown in figure 2.32, the glass surface can be split by the YZ plane to obtain the intersection curve. Thus the code is executed to obtain the intersection points along the YZ plane and the resultant set of points represent the curve for direction of slicing. The points are patterned along the curve highlighted in figure 2.33. Hence, the Slicing will be done by the XZ plane as done in the Slicing chain Approach. But instead of shifting by the patterning distance along Y axis, the slicing plane will pass through each obtained patterning point on the curve, thus obtaining exact uniform pattern.

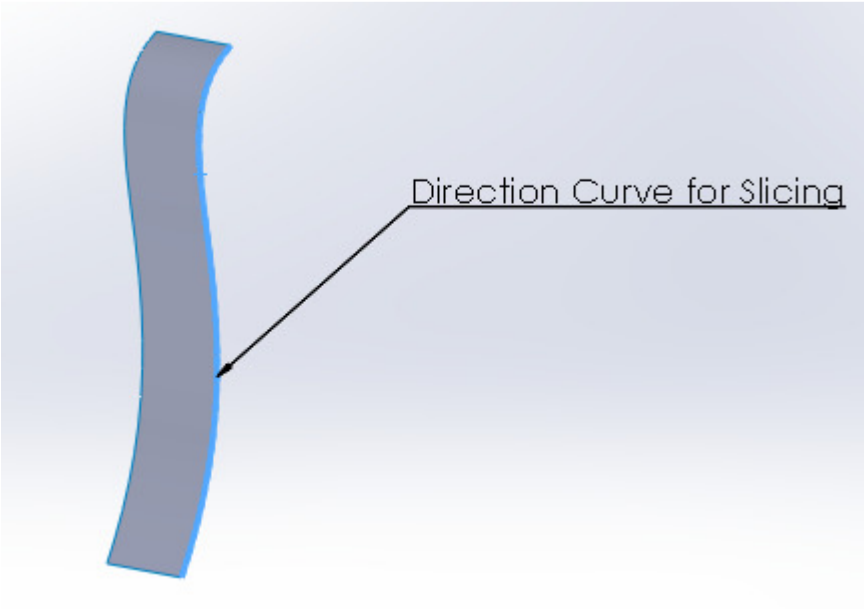


Figure 2.33: Direction Curve for Slicing - Uniform Curved Surface 2

Similarly for the figure 2.33, the points are patterned along the highlighted curve using slicing chain approach and the slicing is done at each patterning point on the curve. The testing and results for this Slicing-Chain approach and modified approach will be discussed in chapter four.

3. OBTAINING X, Y, Z TRANSLATIONS AND A,B ANGLES FOR THE 5-AXIS MACHINE

To obtain the X, Y, Z translations and A,B angles for the 5-axis machine, the normal representing the point comes into play. The normal for any point is represented by following equation. This is obtained as explained in sections 2.1 and 2.2 for STL and VRML formats.

$$n = a\,\vec{i} + b\,\vec{j} + c\,k \tag{3.1}$$

The normal needs to align with the tool of the machine, which is assumed to be vertical. Thus normal first needs to be aligned with the YZ plane first and then with the z axis.

The B angle represents the angle for rotation of the normal in XY plane to align with the YZ plane. Whereas the A angle represents the angle for rotation in YZ plane to align with the z axis.

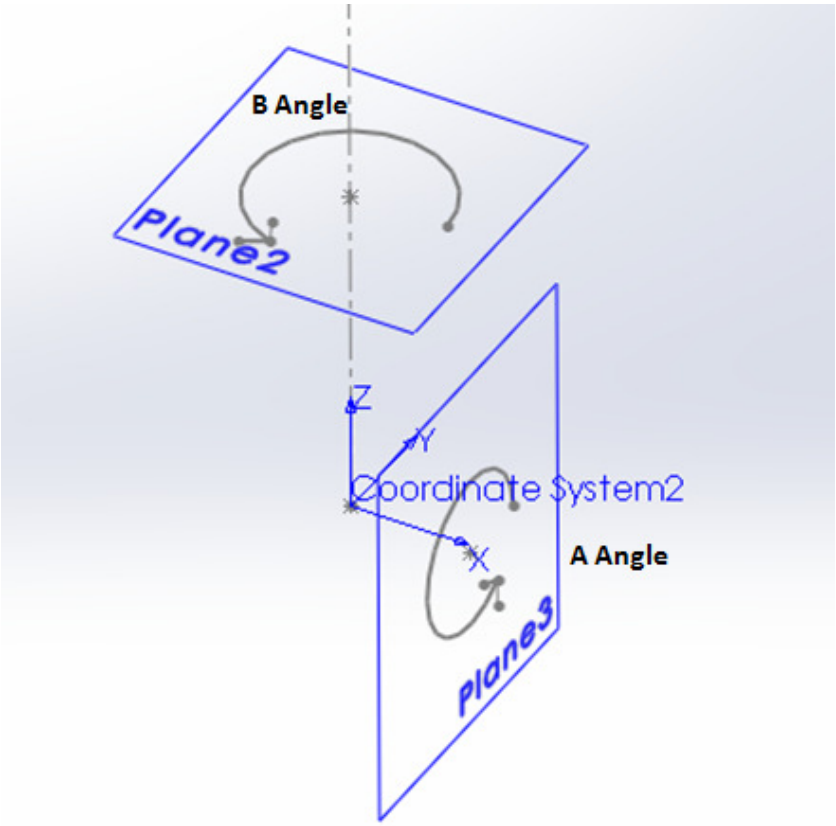


Figure 3.1 A and B angle orientation

To obtain B angle, the angle between the normal and the YZ plane measured along the XY plane is needed. This angle will can be calculated by measuring the angle between the 'normal projected onto the XY plane' and the 'YZ plane'.

Therefore, first the normal is projected onto the XY plane

$$n_{Projection\,XY} = a\,\vec{\imath} + b\,\vec{\jmath} + 0\,\vec{k} \qquad (3.2)$$

The B angle can be calculated by measuring the angle between the normal projected on XY plane and the y axis. The vectors representing the normal projected onto the XY plane and the y axis, both normal lie on XY plane. Thus, just a rotation in XY plane has to be calculated.

$$B\ angle = n_{Projection\ XY} .\ (0\,\vec{\imath} + \vec{\jmath} + 0\,\vec{k}) \tag{3.3}$$

Similarly, to obtain the A angle, the angle between the normal projected onto the YZ plane and the XY plane is needed. It can be calculated by measuring the angle with the x axis as the normal is projected onto the YZ plane and both vectors representing x axis and projected normal lie on the YZ plane.

$$n_{Projection\ YZ} = 0\,\vec{\imath} + b\,\vec{\jmath} + c\,\vec{k} \tag{3.4}$$

The A angle for the machine is defined as rotation angle in the YZ plane. Thus, to obtain the A angle, the angle between the normal projected onto the YZ axis and the z axis is needed.

$$A\ angle = n_{Projection\ YZ} .\ (0\,\vec{\imath} + 0\,\vec{\jmath} + \vec{k}) \tag{3.5}$$

The origin in the CAD file is set to the origin of the machine. But certain axis of machine and the rotation angles A and B may be different from the coordinate system assigned in the CAD. Positive X translation obtained according to CAD coordinate system may be negative with respect to the machine X axis. Therefore, the modifications for obtaining results according to machine has been done in the code. Thus taking care of X, Y, Z axis orientation of the machine, the X, Y, Z translations are determined.

Observing the figure 3.1, angles are considered looking from the top of plane XY in the negative direction of Z axis for angle B and looking from the right in the direction of negative X axis on the YZ plane for A angle.

The position vector in the positive 'i' direction will have an anticlockwise rotation to align with the YZ axis. Similarly, a position vector in the negative 'i' direction will have a clockwise rotation to align with the YZ axis. If the co-efficient of the 'i' coordinate is positive, the B angle is considered positive. The anticlockwise rotation for B angle is considered as positive as shown in figure 3.1. Similarly if 'i' coordinate is negative, the B angle is considered to be negative.

Applying the same logic for calculating the sign of the A angle, the position vector in the positive 'j' direction will have an anticlockwise rotation to align with the YZ axis. Similarly, a position vector in the negative 'j' direction will have a clockwise rotation to align with the YZ axis. If the co-efficient of the 'j' coordinate is positive the A angle is considered positive.

New position vector is obtained after Rotation with B angle.

$$\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = \begin{bmatrix} \cos(B) & -\sin(B) & 0 \\ \sin(B) & \cos(B) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.6}$$

This Position vector is again rotated by A angle, new position vector is obtained after Rotation with A angle.

$$\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ \sin(B) & \cos(B) & 0 \\ \cos(B) & -\sin(B) & 0 \end{bmatrix} * \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} \tag{3.7}$$

The x_A , $y_A z_A$, thus obtained will represent the x, y and z translations. It is assumed that the co-ordinate axis of the machine aligns with the co-ordinate system assumed for the algorithm. If there is any change in the co-ordinate system of the machine, then related transformations or rotations can be obtained easily.

4. TESTING RESULTS AND DISCUSSION

The Slicing-Chain Algorithm was found to be the most effective among the three approaches discussed. Note that other approaches can certainly be identified, however, there were several constraints to consider.

1. We were to be able to consider any object issued from any CAD software. The STL/VRML solution is CAD neutral and will work in all cases. Second, there is the issue of efficiency of computing the points.
2. Identifying points on line segments and identifying intersections of planes and triangles is the most efficient and therefore the fastest to process.
3. It was important to the client to have a pattern of points that looks as uniform as possible. The approach proposed satisfied the client.

The Testing of this algorithm is done with different types of surfaces.

For checking the number of micro-patterning points with expected, an approximate calculation is performed, where surface area of the geometry is calculated. Assuming it as a square with respective surface area, the number of points expected are approximately.

4.1 Planar Surface

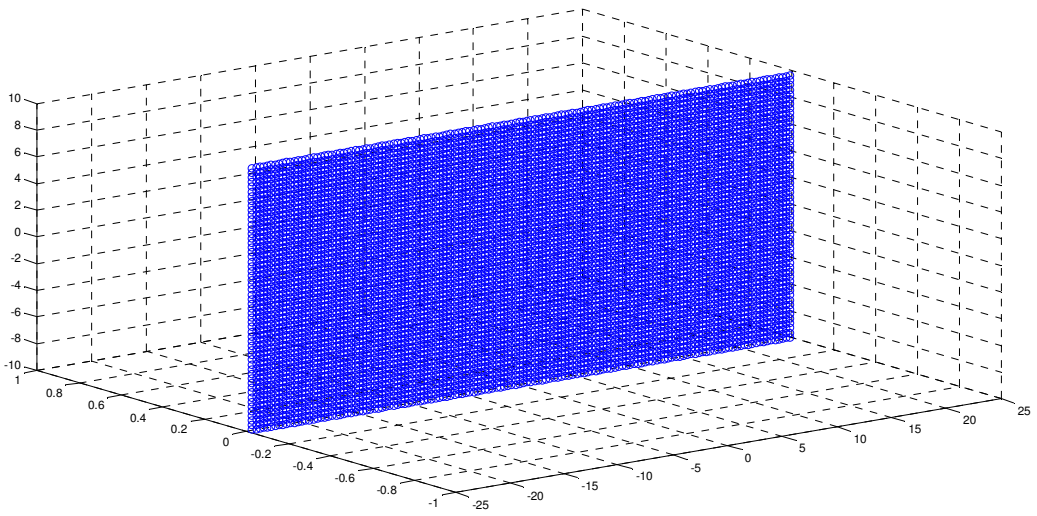


Figure 4.1: Plane Surface with Patterning pitch: 0.35mm

Total number of micro-patterning points obtained are 8094. The rectangle has an area of 1000 sq. mm. Thus approximately expected points is 1000 divided by area of unit square with length equal to patterning distance 0.35 mm which is 0.1225 sq. mm. Thus the number of expected points come out to be approximately 8163.To perform accurate calculation, the rectangle is 20 x 50. Therefore the number of points along both edges are 20/0.35 and 50/0.35, which comes out to be 57 and 142 as integer value for patterning.

57 x 142 is 8094, which implies this method is accurate for planar surfaces when appropriate slicing direction is selected. Time required for the algorithm is 11.256s.

To plot the patterning points using CAD software-Solidworks, following table 4.1 shows time required by Designer. Thus it is observed that it depends upon the expertise of the Designer to plot points. Moreover time taken ranges over from 25-60 for different Designers just to plot the points. Three Designers were chosen for study. The results proves that it does depend on the Designer and also the result obtained is just a partial output of what the algorithm delivers.

Table 4.1 Time Required using CAD-Planar Surface

Designer	Time required (s)
1	30
2	60
3	25

The algorithm designed, reduces human efforts in plotting the points. Adding further it not only plots the points, but also calculates the X, Y, Z translations and the A, B angles required for a 5-axis machine at those points. The algorithm takes only 8-9 s to achieve all together, implying huge benefit of the algorithm.

4.2 Uniform Curved Surface

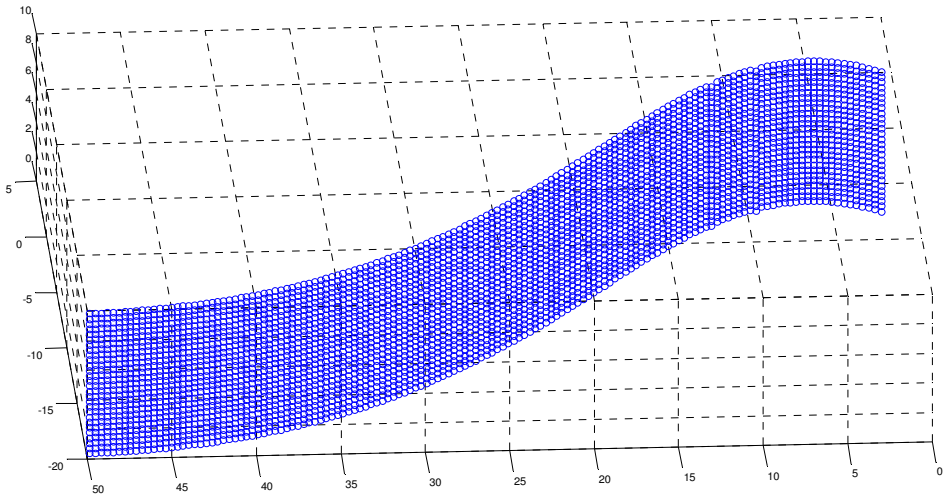


Figure 4.2: Uniform Curved Surface with Patterning pitch: 0.35mm

Total number of micro-patterning points obtained are 3976. This is obtained by slicing direction of x axis, which were 87% of the expected points.

Table 4.2 Analysis for different slicing directions for uniform curved surface

Slicing Direction	No. of Micro - Patterning Points	Surface Area	Patterning Distance	Expected Points	% Points Obtained	Time Required (s)
X	3976	562.12	0.35	4588.734694	87%	8.61
Y	2792	562.12	0.35	4588.734694	61%	8.24
Z	4480	562.12	0.35	4588.734694	98%	8.92

Table 4.3 Time Required using CAD-Uniform Curved Surface

Designer	Time required (s)
1	900
2	755
3	600

Table 4.3 proves that the average time required by the three Designers is much more than the 8-9 s taken by the Algorithm. The CAD software also takes more time as the complexity of the problem increases. Whereas the algorithm works in similar manner for any geometry.

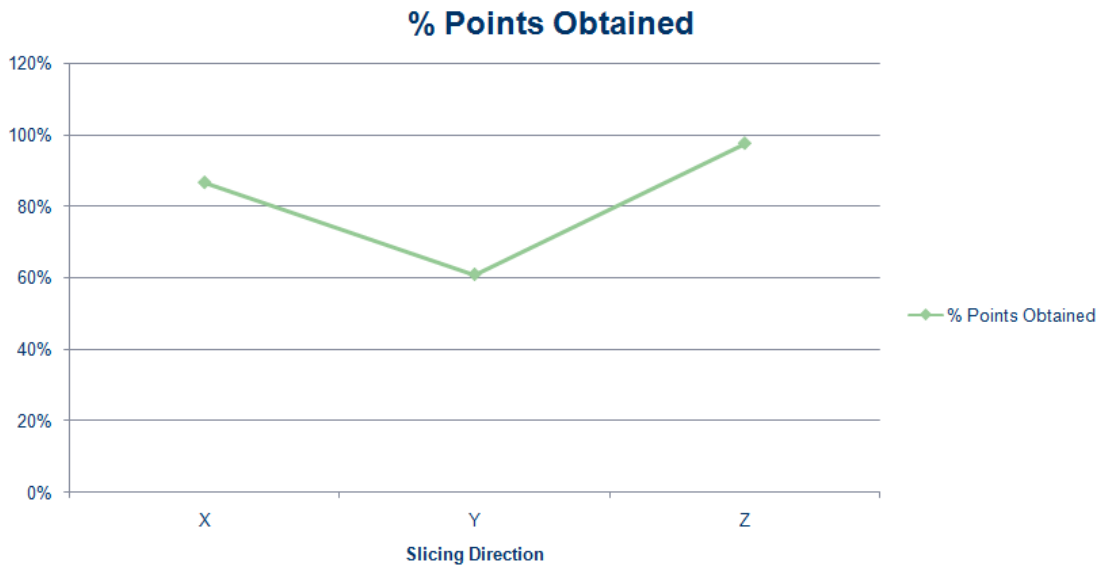


Figure 4.3 Effect of Slicing Direction on micro-patterning for uniform curved surface

Figure 4.3 shows the plot of percentage of micro-pattering points obtained when compared to expected points.

From Table 4.2 it can be observed that the slicing-chain algorithm works efficient if correct slicing direction is selected. Slicing as z direction is the direction of extrusion of the sketch to form a surface. Thus this slicing direction yields maximum number of micro-patterning points.

But for this type of uniform curved surface, which is generated from a curved sketch extruded in a single direction, accurate results can be obtained. Thus this method is readily applicable for these types of surfaces.

4.3 Non- Uniform Curved Surface

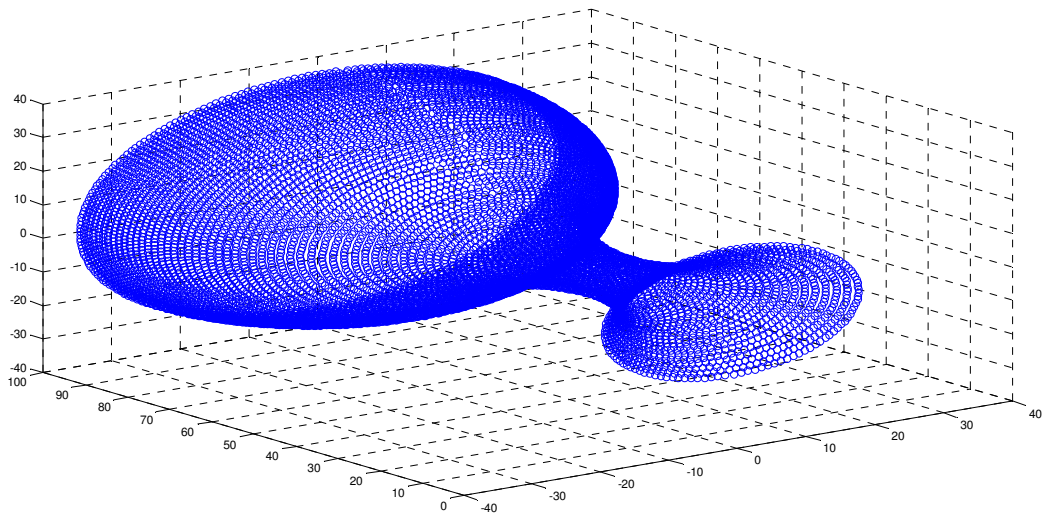


Figure 4.4: Non-Uniform Curved Surface with Patterning pitch: 1mm

Figure 4.4 shows the 3D plot of the patterning points for the glass. Total number of micro-patterning points obtained in the figure 3.4 are 10455, which are 87% of the total approximate expected points.

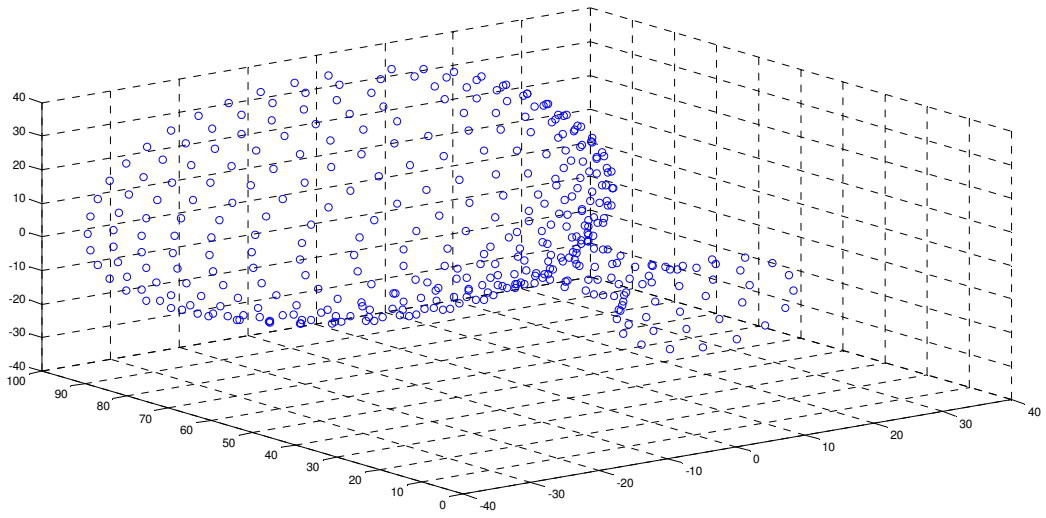


Figure 4.5: Non-Uniform Curved Surface with Patterning pitch: 5mm

Total number of micro-patterning points obtained in the figure 4.5 are 396, which are 82% of the total approximate expected points.

Table 4.4 Analysis for different slicing directions for non-uniform curved surface

Slicing Direction	No. of Micro -Patterning Points	Surface Area	Patterning Distance	Expected Points	% Points Obtained	Time Required (s)
X	35871	12075	0.5	48300	74%	141.227
Y	42168	12075	0.5	48300	87%	153.684
Z	35881	12075	0.5	48300	74%	141.573

Varying the slicing direction affects the points obtained by micro-patterning. For this non-uniform curved surface, the slicing in Y direction gives more accurate. It is because it is perpendicular to the shaft axis for the glass. For given any arbitrary surface, the results can be checked for different slicing directions and the most accurate of them can be selected.

Table 4.5 Time Required using CAD-Non- uniform Curved Surface

Designer	Time required (s)
1	865
2	720
3	480

Table 4.5 proves that the average time required by the three Designers is much more than the 8-9 s taken by the Algorithm. The CAD software also takes more time as the complexity of the problem increases. Whereas the algorithm works in similar manner for any geometry.

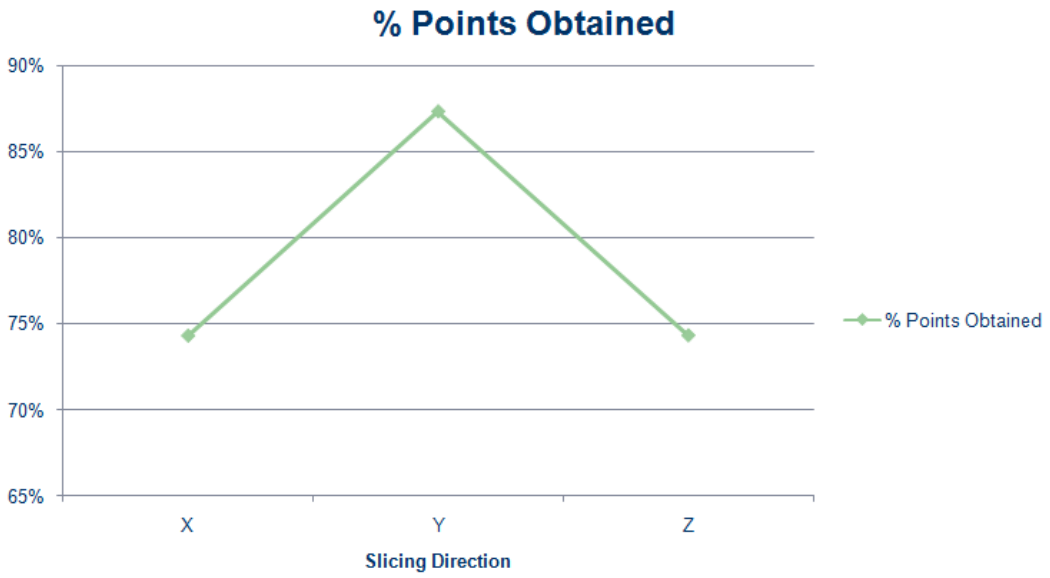


Figure 4.6 Effect of Slicing Direction on micro-patterning for non-uniform curved surface

Table 4.6 Analysis for varying patterning distance for non-uniform curved surface

Slicing Direction	No. of Micro - Patterning Points	Surface Area	Patterning Distance	Expected Points	% Points Obtained	Time Required (s)
Y	42168	12075	0.5	48300	87%	153.684
Y	10455	12075	1	12075	87%	68.085
Y	396	12075	5	483	82%	19.508

It can be observed that this method is more accurate for micro-patterning, and as the patterning distance is increased after 1mm, lesser percentage of expected points would be obtained.

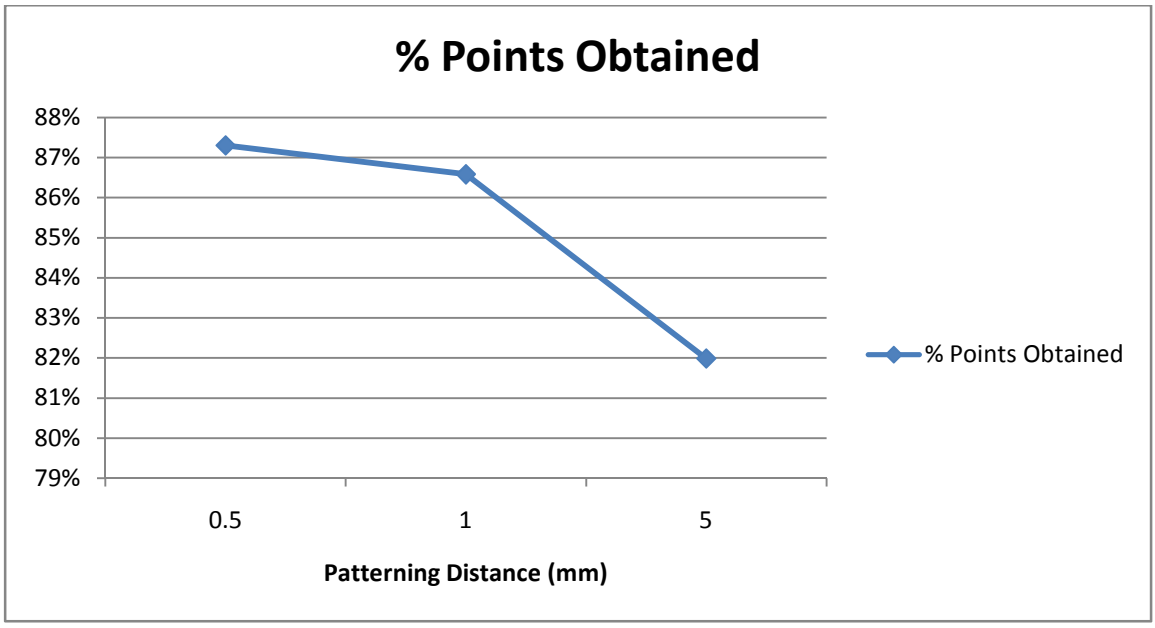


Figure 4.7: Effect Patterning distance on micro-patterning for non-uniform curved surface

Table 4.7 Analysis for varying patterning distance for non-uniform curved surface modifying slicing direction

Slicing Direction	No. of Micro - Patterning Points	Surface Area	Patterning Distance	Expected Points	% Points Obtained	Time Required (s)
Y	47368	12075	0.5	48300	98%	221.89
Y	23902	12075	0.7	24642	96%	212.64
Y	11652	12075	1	12075	96%	201.56

Table 4.7 shows results modifying the slicing direction along the curve as shown in the figure 2.32 for the slicing chain approach. The points are patterned with a uniform distance 'd' along the curve line shown in figure 2.32. It is followed by offsetting the ZX planes in y direction at the points of the pattern obtained along the curve. As a result this has eliminated the slicing error mentioned in figure 2.22. Hence, the accuracy obtained in table 4.7 is higher than that of table 4.6, where slicing error existed.

For the direct interpolation approach, it fails mostly when the patterning distance is more than the length of the sides of the triangles obtained. This can be rectified by decimating the STL mesh specified in [10]. But this will increase more approximation of the geometry. Thus, even if the pattern can be plotted on the decimated STL file, it will be less accurate when compared to the actual surface data.

There has been much research done in field of NURBS [8,9]. Offsetting curves by uniform distance, then interpolating points on the curves can be an approach to obtain patterning on any surface. The offset of NURBS have intersections with each other. Method has been proposed in [8] for rectifying the intersections. AS the profile changes or at the extreme inner curves, the distance of one curve with the nearby does not

maintain itself at the offset distance. This can also be an area of interest for applying micro-patterns.

A triangular meshed model from an FEA software will also function in same way as an STL file for inputting the coordinates and normals data to the algorithm.

The Slicing approach fails when the intersections have length less than the patterning distance. Also error occurs when there is remainder of the intersection after interpolation is done.

The Slicing-Chain approach has so far proven to be applicable method for use of micro-patterning. For planar surfaces, it is accurate when correct slicing direction is chosen. For uniform curved surfaces which have a sketch extruded in a single direction, the Slicing-Chain approach will be provide more accurate results if the slicing direction is chosen as the extrusion direction. For any arbitrary surface, varying the slicing direction affects the number of points generated in the micro-pattern.

5. CONCLUSION AND FUTURE WORK

5.1 Conclusions

It can be concluded that the method proposed by the Slicing-chain algorithm is a good method for micro-patterning. In addition, the slicing direction can be modified for better results. The previous approaches of direct interpolation and direct slicing fail when the patterning distance is more than the geometric parameters of a triangle.

If a material removal process (micro-drilling) has to be carried out at the pattern locations, then the X, Y, Z translations and the A, B angles for a 5-axis machine are obtained by the algorithm. Also if there are any CAD operations on those points generated by the micro pattern, then these points can be imported into any CAD package.

The limitation for this approach is the slicing error caused, when the distance of the pattern points have an error when measured along geometry as discussed earlier for figure 2.22. But the slicing chain approach can be modified as shown in section 2.6 to improve the patterning of the points.

5.2 Future Work

Investigation can be done on how to reduce the slicing error and then interpolate points for patterning. Effective ordering of the points for machining can be studied. Research can be done on how to incorporate the use of offset of NURBS for the micro-patterning algorithm. New approaches can be attempted such as whereby a pattern is created by a point-to-point approach starting from a base point on the geometry. This can

be done using a 2D mesh starting from a point and obtaining each point at a patterning distance from the neighboring points.

REFERENCES

1. J.L. Huertas-Talon, C. Garcia-Hernandez, L. Berges-Muro, and R. Gella-Marin, “Obtaining a spiral path for machining STL surface using non-deterministic techniques and spherical tool”, Computer Aided Design, vol. 50, 2014, pp. 41-50

2. M. Szilvasi-Nagy, G. Matyasi, Analysis of STL filesMath. Comput. Modelling, 38 (2003), pp. 945–960

3. StereoLithography Interface Specification, 3D Systems, Inc., July 1988

4. H. Baerten, F. Van Reeth, Using VRML and Java to visualize 3D algorithms in computer graphics education, in: Proc. Edugraphics 97, Grasp Queluz, 1997, pp. 269–274.

5. A. Ramesh, W. Akram, S.P. Mishra, A.H. Cannon, A.A. Polycarpou, W.P. King, Friction characteristics of microtextured surfaces under mixed and hydrodynamic lubrication, Tribol Int, 57 (2013), pp. 170–176

6. D. Braun, C. Greiner, J. Schneider, P. Gumbsch, Efficiency of laser surface texturing in the reduction of friction under mixed lubrication, Tribol Int, 77 (2014), pp. 142–147

7. L. Ventola, M. Dialameh, M. Fasano, E. Chiavazzo, P. Asinari, Convective heat transfer enhancement by diamond shaped micro-protruded patterns for heat sinks: thermal fluid dynamic investigation and novel optimization methodology, Appl Therm Eng, 93 (2016), pp. 1254–1263

8. J.-L. Shih, S.-H. Frank Chuang, One-sided offset approximation of freeform curves for interference-free NURBS machining, Computer Aided Design, 40 (9) (2008), pp. 931–937

9. G.V.V. Ravi Kumar, K.G. Shastry, B.G. Prakash, Computing constant offsets of a NURBS B-Rep, Computer-Aided Design, 35 (2003), pp. 935–944

10. William J. Schroeder, Jonathan A. Zarge, and William E.Lorensen. Decimation of triangle meshes. Computer Graphics, (SIGGRAPH ’92 Proc.), 26(2):65–70, July 1992.

11. H. Pottmann, C. Jiang, M. Höbinger, J. Wang, P. Bompas, J. Wallner, Cell packing structures, Comput-Aided Des, 60 (2015), pp. 70–83

12. Efficient slicing for layered manufacturing. / Tata, Kamesh; Fadel, Georges; Bagchi, Amit; Aziz, Nadim. In: Rapid Prototyping Journal, Vol. 4, No. 2-4, 01.12.2098, p. 151-167.

13. Xu Xiduo, Li Jijun and Zheng Hong, “A new NURBS Offset curves and surfaces Algorithm based on different Geometry Shape”, IEEE 10th

International Conference on Computer-Aided Industrial Design &Conceptual Design, 2009, pp 2384-2390

14. Les A. Piegl and Wayne Tiller, “Computing Offsets of NURBS curves and surfaces”, Computer-Aided Design, 1999, Vol.31, pp 147-156
15. MATLAB R2014, The MathWorks Inc.
16. Solidworks 2014, Dassault Systems
17. Hyperworks 2013, Altair

APPENDICES

APPENDIX A. STL FILE OF PLANAR SURFACE

```
solid plane1
  facet normal 0.000000e+000 1.000000e+000 0.000000e+000
    outer loop
      vertex 2.500000e+001 0.000000e+000 -1.000000e+001
      vertex -2.500000e+001 0.000000e+000 -1.000000e+001
      vertex 2.500000e+001 0.000000e+000 1.000000e+001
    endloop
  endfacet
  facet normal 0.000000e+000 1.000000e+000 0.000000e+000
    outer loop
      vertex 2.500000e+001 0.000000e+000 1.000000e+001
      vertex -2.500000e+001 0.000000e+000 -1.000000e+001
      vertex -2.500000e+001 0.000000e+000 1.000000e+001
endloop
  endfacet
endsolid
```

APPENDIX B. STL FILE OF UNIFORM CURVED SURFACE

```
solid curved surf1
  facet normal -3.700537e-002 -9.993151e-001 0.000000e+000
    outer loop
      vertex 5.000000e+001 -2.000000e+001 0.000000e+000
      vertex 5.000000e+001 -2.000000e+001 1.000000e+001
      vertex 4.969327e+001 -1.998837e+001 1.000000e+001
    endloop
  endfacet
  facet normal -3.875547e-002 -9.992487e-001 0.000000e+000
    outer loop
      vertex 5.000000e+001 -2.000000e+001 0.000000e+000
      vertex 4.969327e+001 -1.998837e+001 1.000000e+001
      vertex 4.969327e+001 -1.998837e+001 0.000000e+000
    endloop
  endfacet
  facet normal -4.591546e-002 -9.989453e-001 0.000000e+000
    outer loop
      vertex 4.969327e+001 -1.998837e+001 0.000000e+000
      vertex 4.969327e+001 -1.998837e+001 1.000000e+001
      vertex 4.877059e+001 -1.994351e+001 1.000000e+001
    endloop
  endfacet
.
.
.
.
.
  facet normal 5.087036e-001 -8.609417e-001 0.000000e+000
    outer loop
      vertex 3.520512e-001 2.051629e-001 0.000000e+000
      vertex 0.000000e+000 0.000000e+000 1.000000e+001
      vertex 0.000000e+000 0.000000e+000 0.000000e+000
endloop
  endfacet
endsolid
```

APPENDIX C. VRML FILE FOR PLANAR SURFACE

```
#VRML V1.0 ascii
Separator {
MaterialBinding {
value OVERALL
}
Material {
ambientColor [
 0.792157 0.819608 0.933333
]
diffuseColor [
 0.792157 0.819608 0.933333
]
emissiveColor [
 0.000000 0.000000 0.000000
]
specularColor [
 0.396078 0.409804 0.466667
]
shininess [
 0.400000
]
transparency [
 0.000000
]
}
Coordinate3 {
point [
 -25.000000 0.000000 -10.000000, -25.000000 0.000000 10.000000, 25.000000
0.000000 -10.000000, 25.000000 0.000000 10.000000
]
}
IndexedFaceSet {
coordIndex [
 2, 0, 3, -1, 3, 0, 1, -1
]
}
}
```

APPENDIX D. PROGRAM MANUAL

Program To Read STL File And Create Micro-Pattern of Points
Obtains Points, Normals, X,Y,Z Translations, B & A Angles

Developers:

Viraj Kulkarni
Graduate Student
Department of Mechanical Engineering
Clemson University
Clemson, SC 29634-0921 USA

Dr. Georges Fadel
Professor and ExxonMobil Employees Chair in Engineering
Department of Mechanical Engineering
Clemson University
Clemson, SC 29634-0921 USA

Introduction:

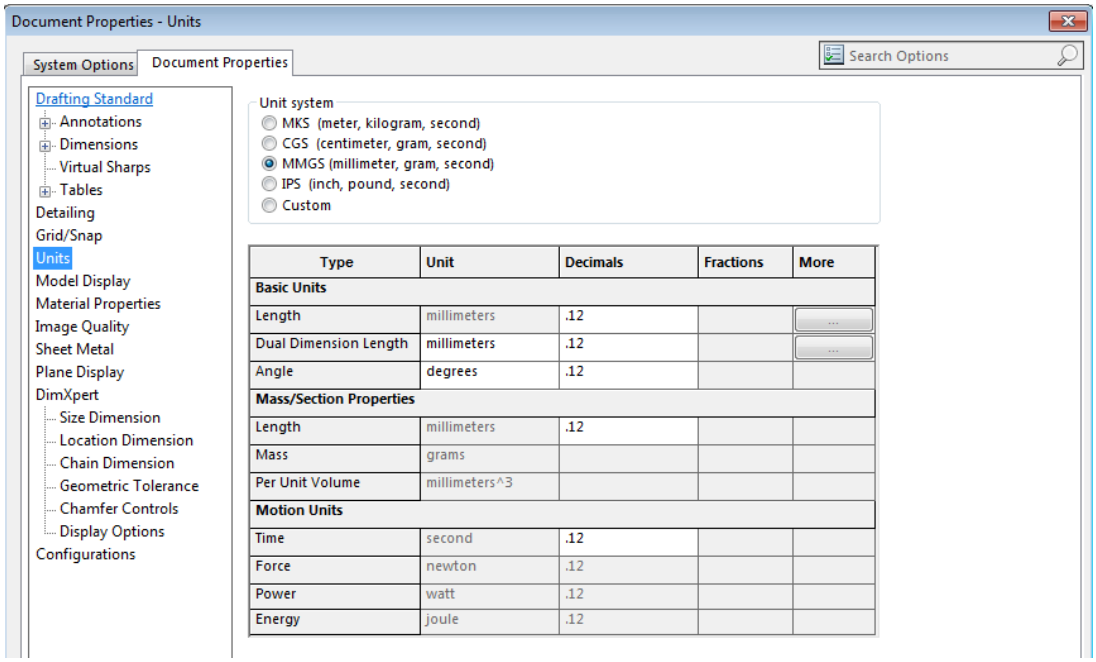
The aim of the program is to apply a pattern on a surface on the micro-scale. Matlab and Solidworks are used in the process. CAD file is imported into Solidworks. After assigning the new Co-ordinate system, the CAD file is converted into a STL format. A Program has been coded using Matlab to read the STL file, store the data and obtain the micro-pattern of the points. The result is a spreadsheet which contains the X,Y,Z translational values for the machine, the B and A angles for the machine, the actual X,Y,Z Coordinates and the normals of the obtained points for CAD file.

Following are the steps to run the program

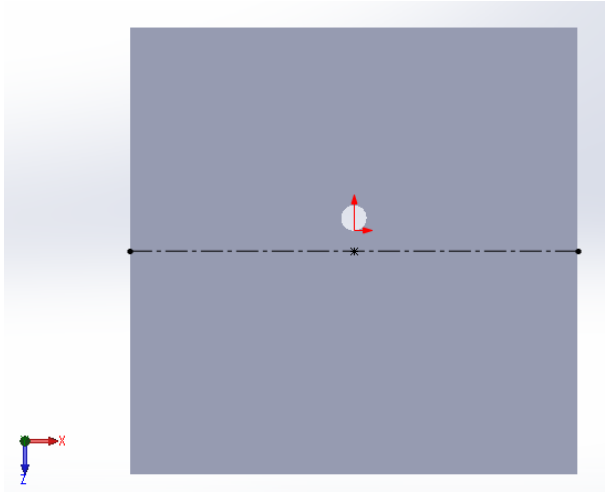
I. Create a STL(.stl) file using Solidworks
II. Run the Program using Matlab

(I) Steps for creating a STL file:

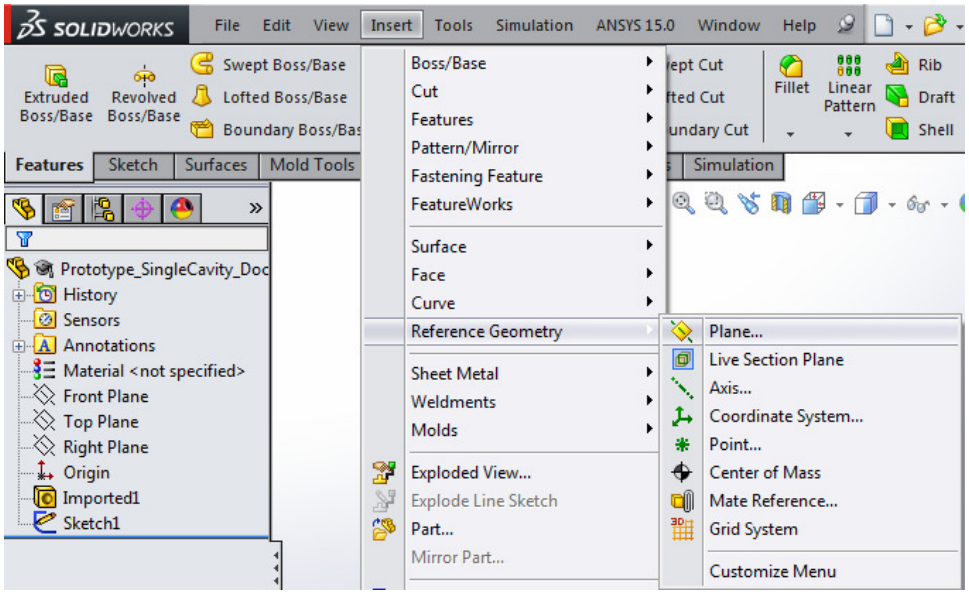
1. Open the CAD file in Solidworks.
2. Verify that the units used for the CAD file are in mm
   Go to Tools - Options
   Click Document Properties and select MMGS.



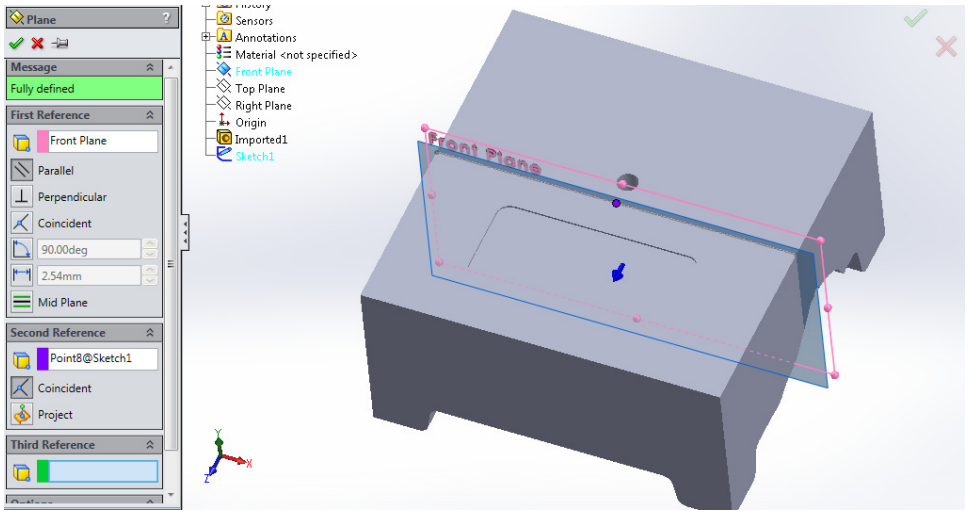
3. Draw a sketch (In proposed XY plane or Top view) to mark the projection of the new origin.
   Draw a centre line and plot the midpoint on it.



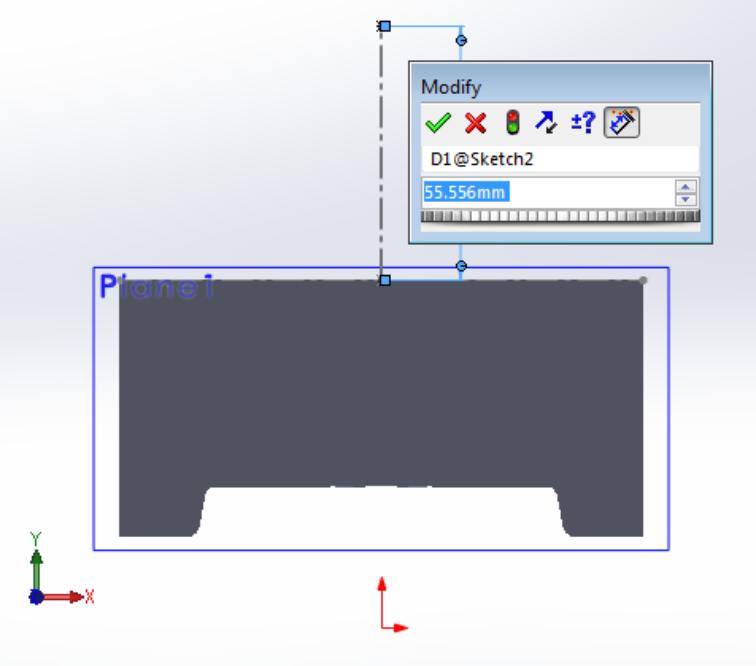
4. Insert Vertical Plane (proposed ZX plane) passing through that point
   Go to Insert - Reference Geometry - Plane



Select the parallel plane to proposed ZX plane and select the point created.
(In absence of a parallel plane, a plane has to be created before)



5. Draw a sketch to mark the point for proposed Origin at the appropriate position.

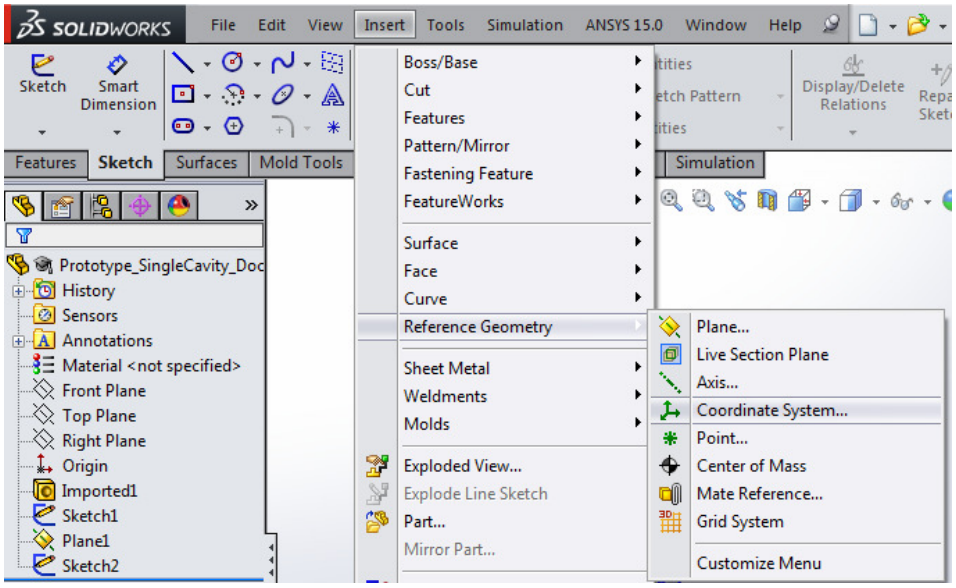


6. Insert New Co-Ordinate System

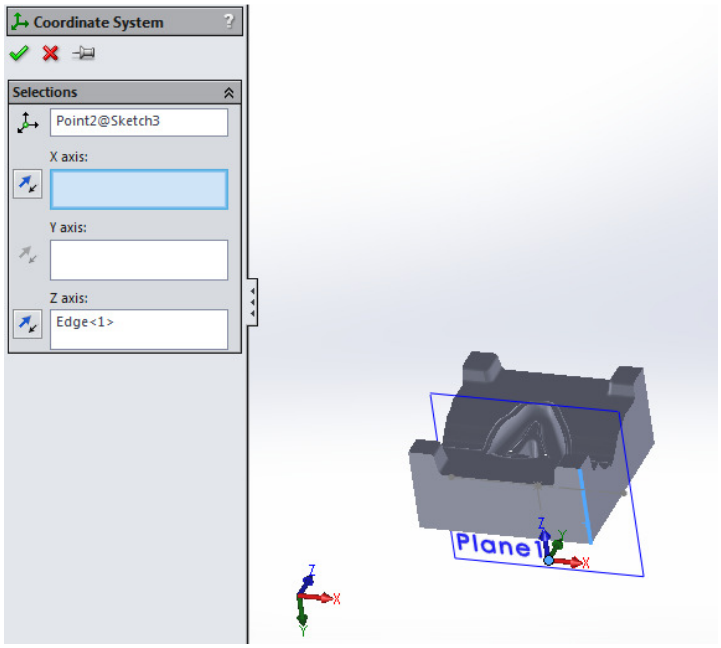
   Always assign the Co-ordinate system as assigned in the following figure.

   (X positive towards right and Y positive towards top in the top view and Z upwards positive)

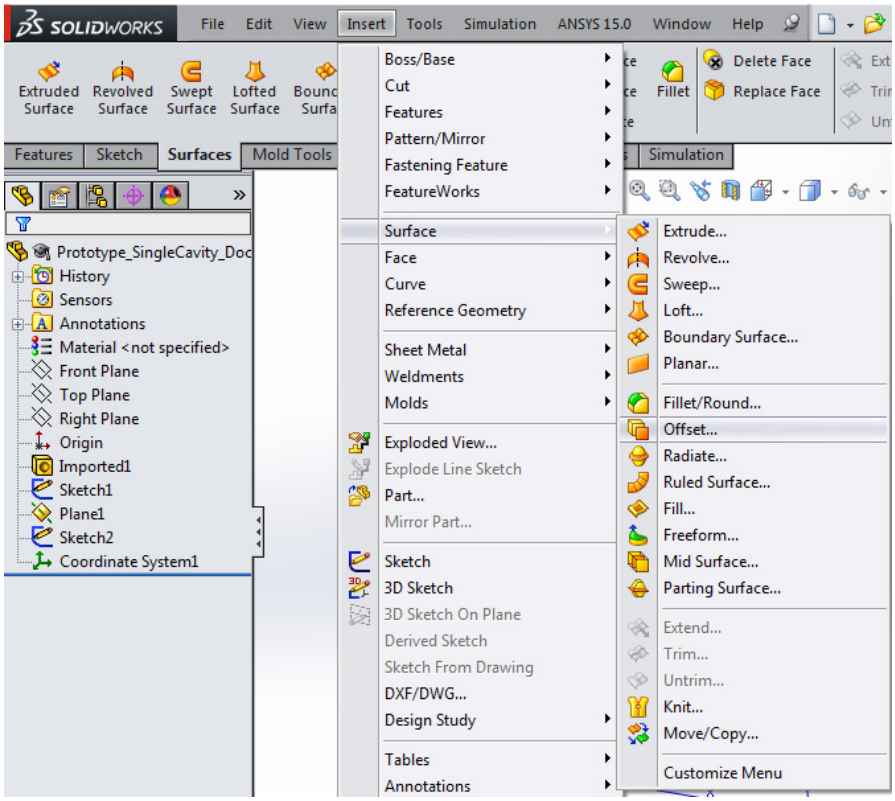
   Note: The X,Y and Z axis of the machine are reverse of the CAD system to be followed in Solidworks, The co-ordinate system of the machine cannot be followed in Solidworks due to its axis orientations. The program interprets the values in co-ordinate system of the machine)



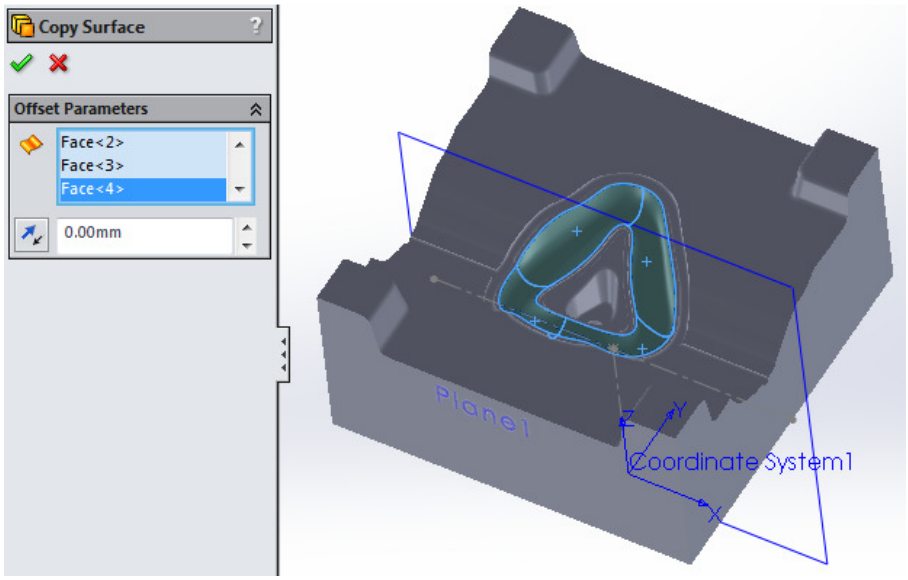
Select the New point for Origin and Select X,Y,Z axes accordingly.



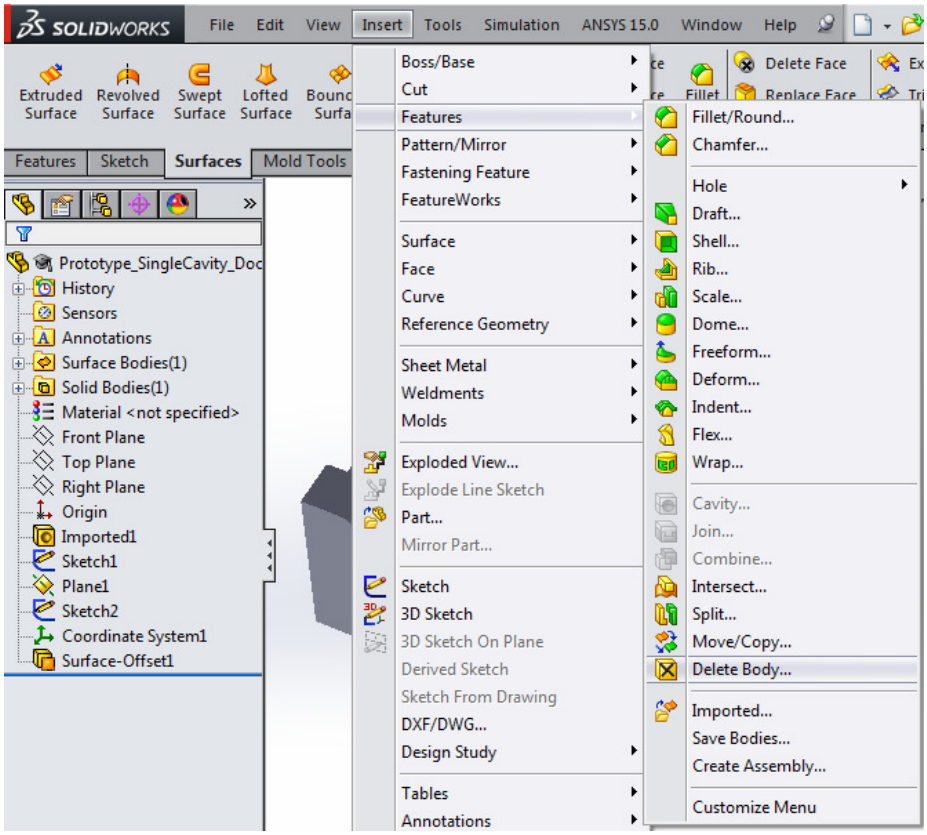
7. Extract the Surfaces on which pattern has to be created.
   Go to Insert - Surface - Offset



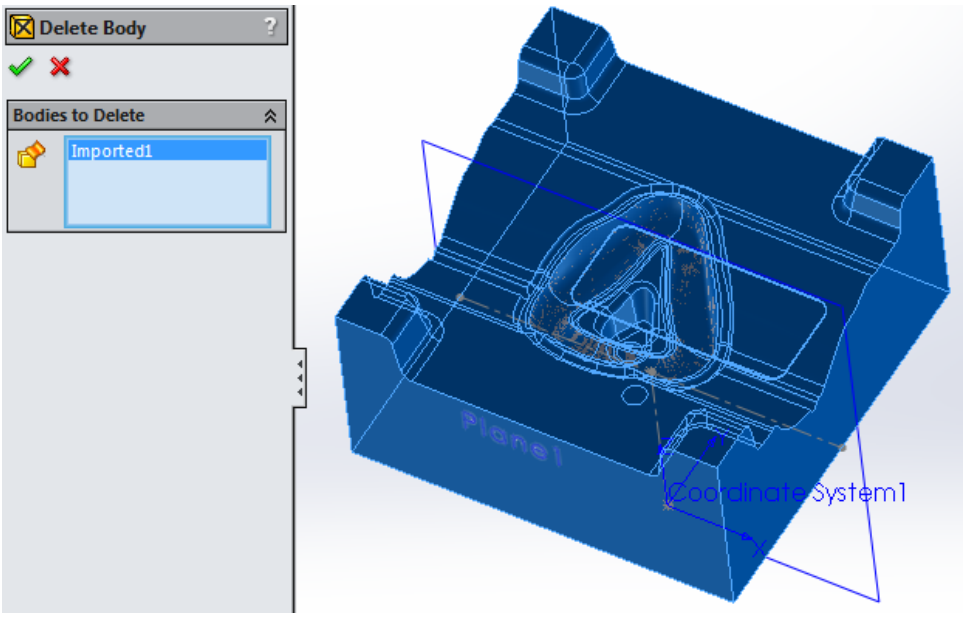
Enter Offset distance as zero



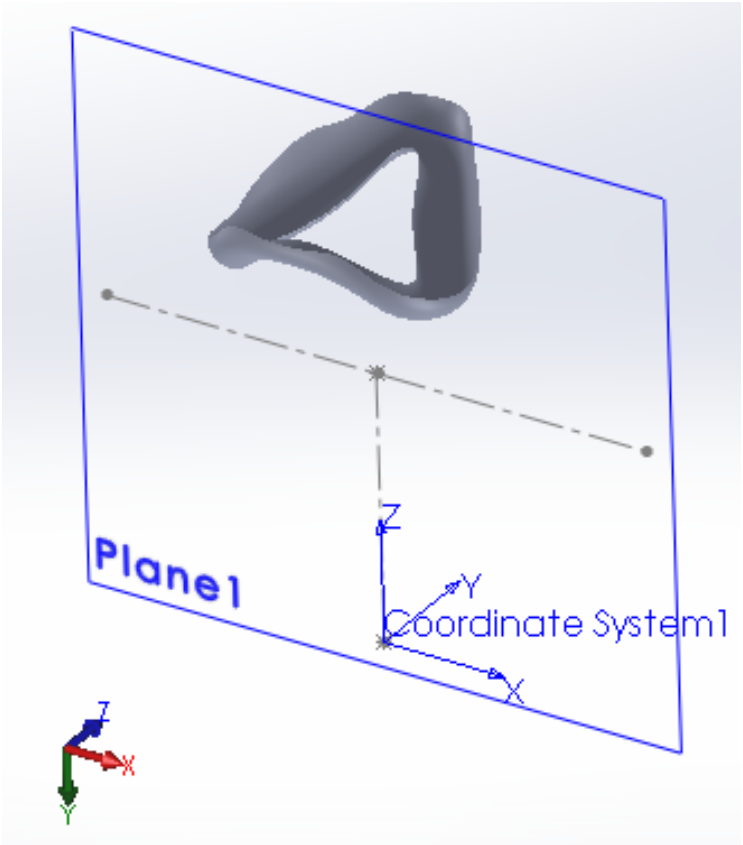
8. Delete the Body to obtain just the surface in the CAD file.
   Go to Insert - Features - Delete Body



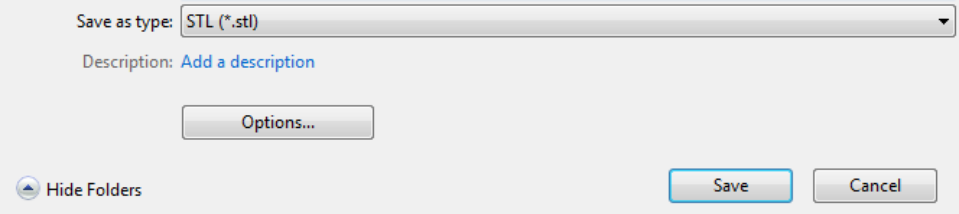
Select the Body to delete



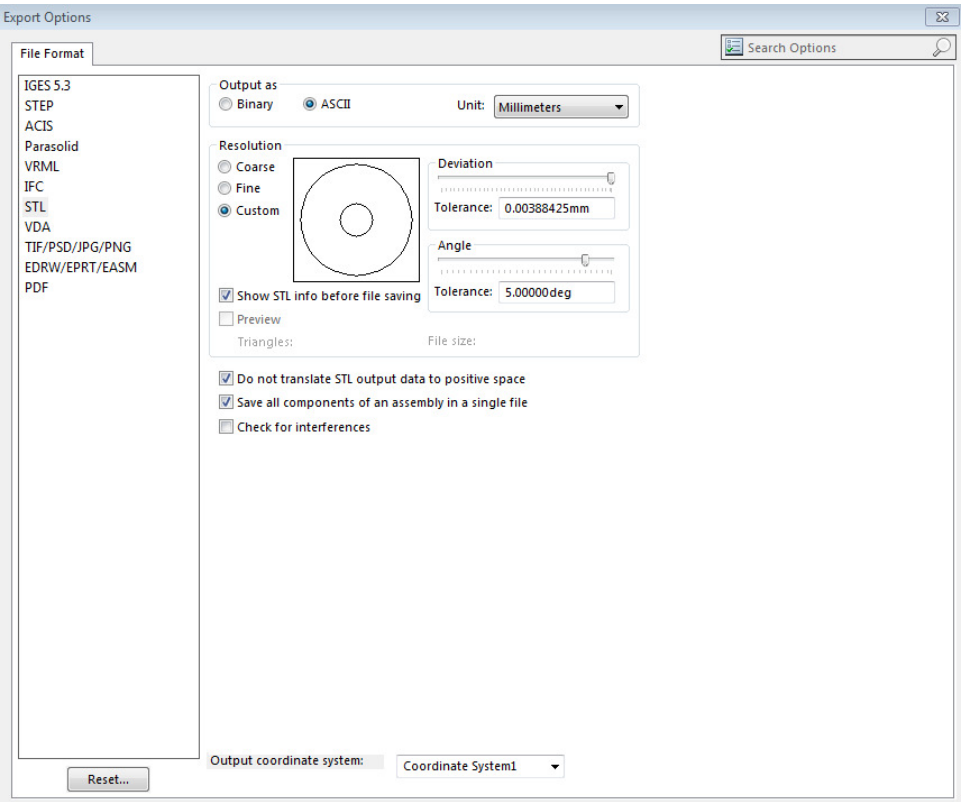
Following will be displayed as result:



9. Save as STL file (.stl)
   Go to File - Save as - Select STL format and click options



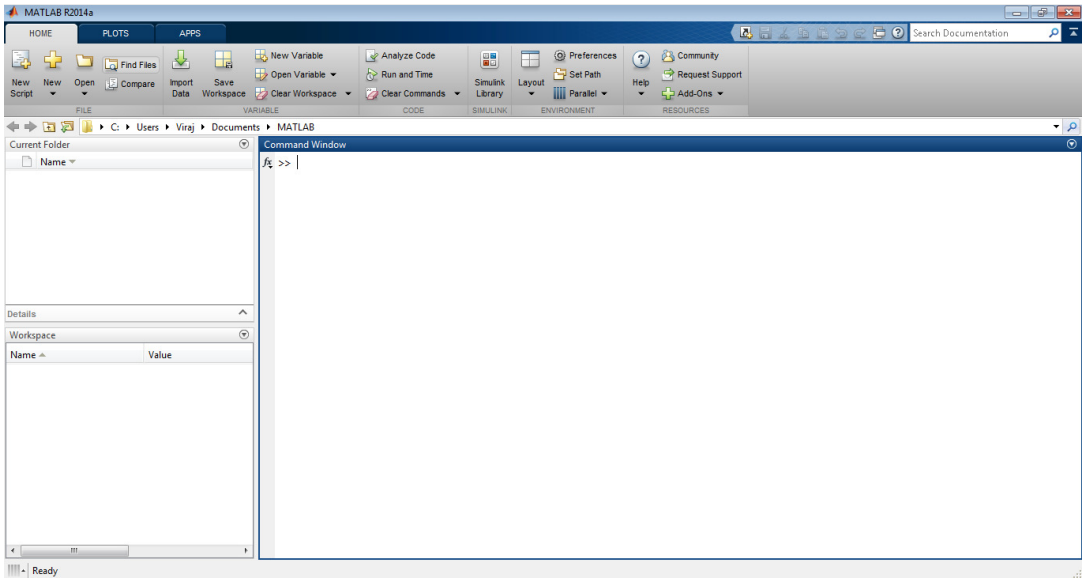
Select Version: STL and Units: millimeters. Select new 'Co-ordinate System 1' in Output Co-ordinate system.



10. Again Click save as Solidworks part file also to have main file back up.

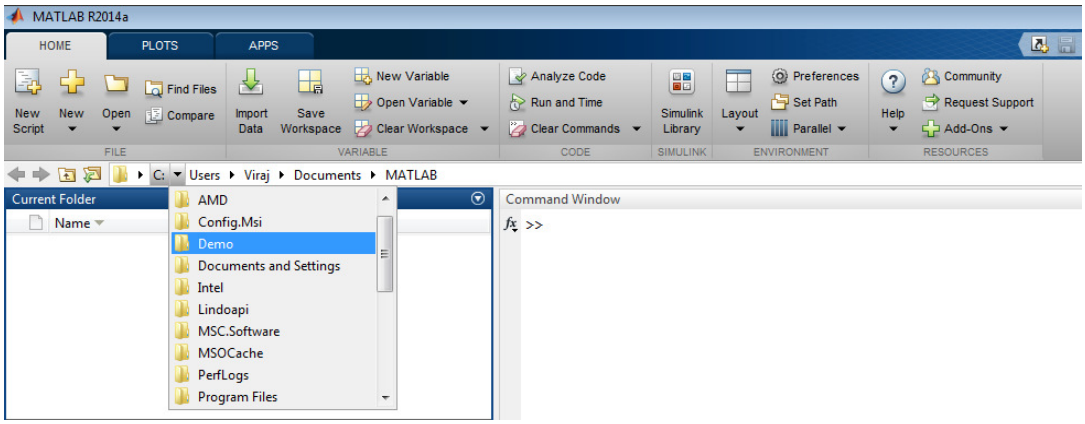
(II) Run the Program using Matlab

1. Make sure that the STL file and the program are saved into a same folder or location.
2. Open Matlab

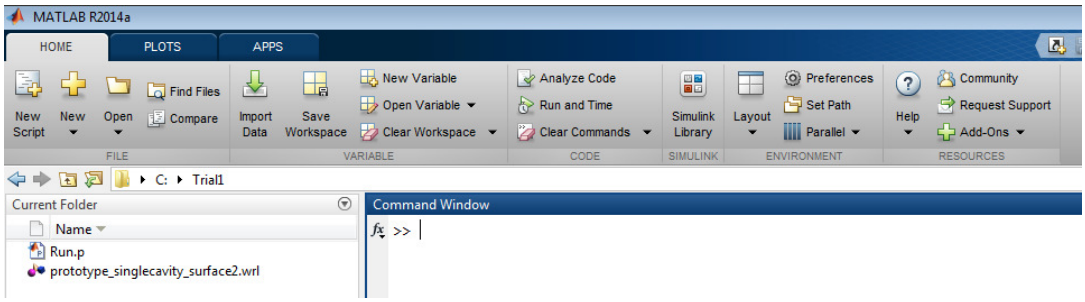


The default location path for running a program in Matlab is:
C:\Users\'Username'\Documents\Matlab

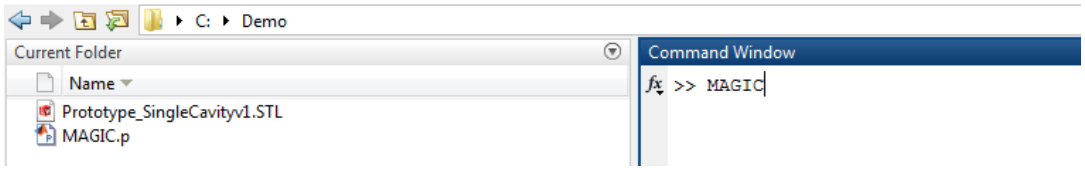
3. Change the location of the current folder to the location where the program and the STL file is saved.



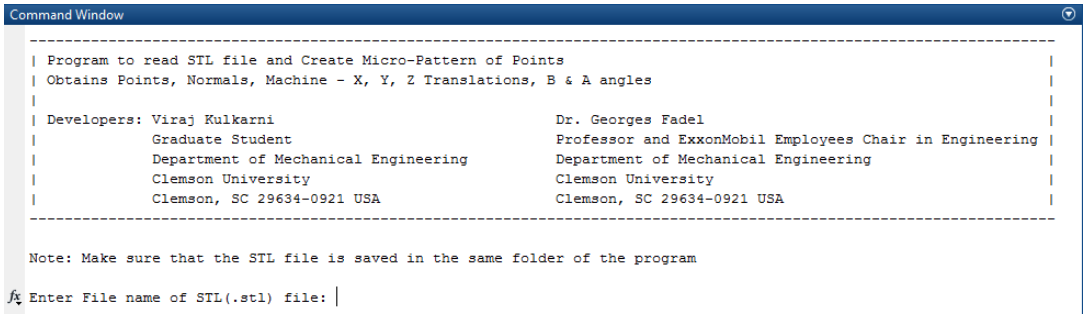
The folder location will be displayed.



4. Type the name of the program: Magic, and press enter

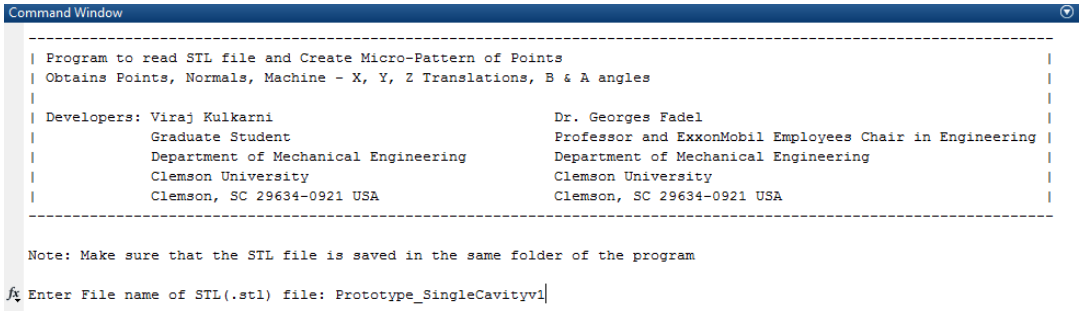


5. The program will be started.

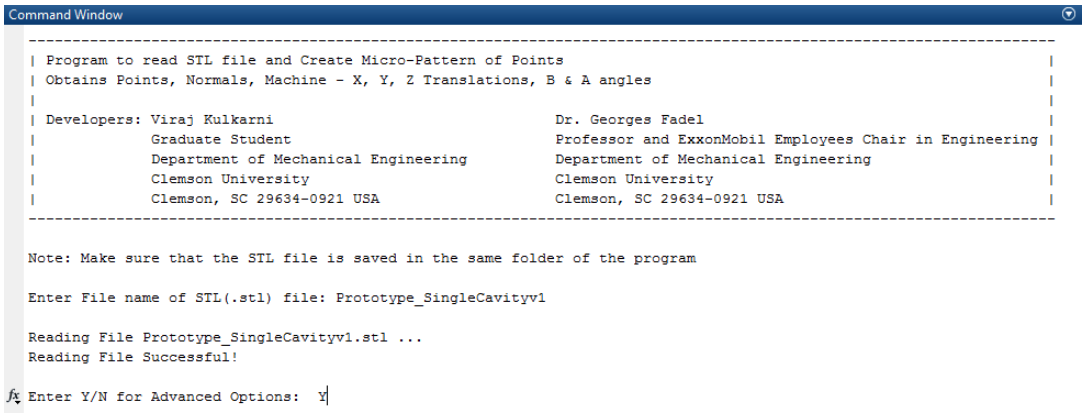


Enter the file name of the STL file and click enter.

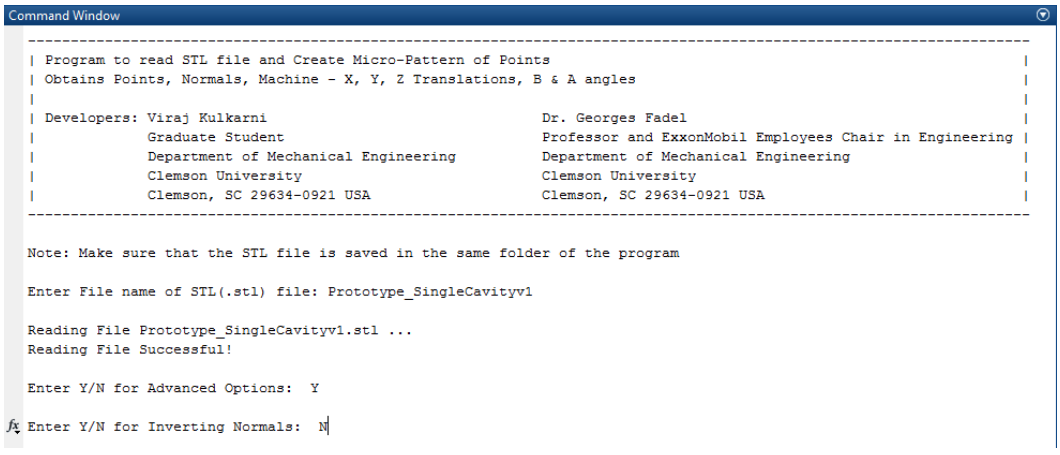
In this case it is 'Prototype_SingleCavityv1'



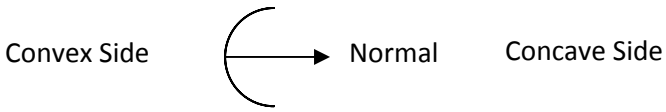
6. After reading the STL file, 'Reading STL file successful' will be displayed. Enter Y or N for the advanced options.



7. The first option is to Invert the Normal

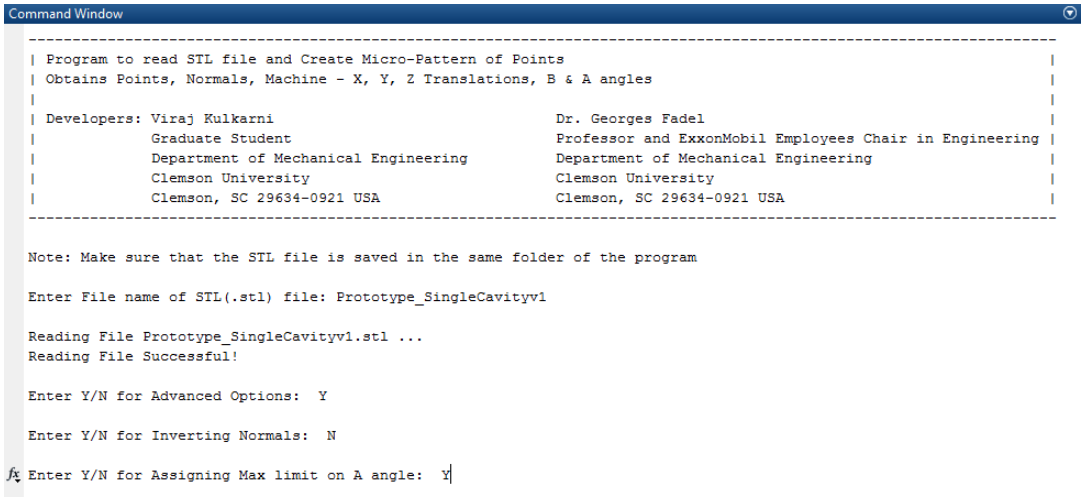


The normal is outward, where the material is not present. But as this is a surface, it can have normal on either sides. Generally the normal generated is at the concave side

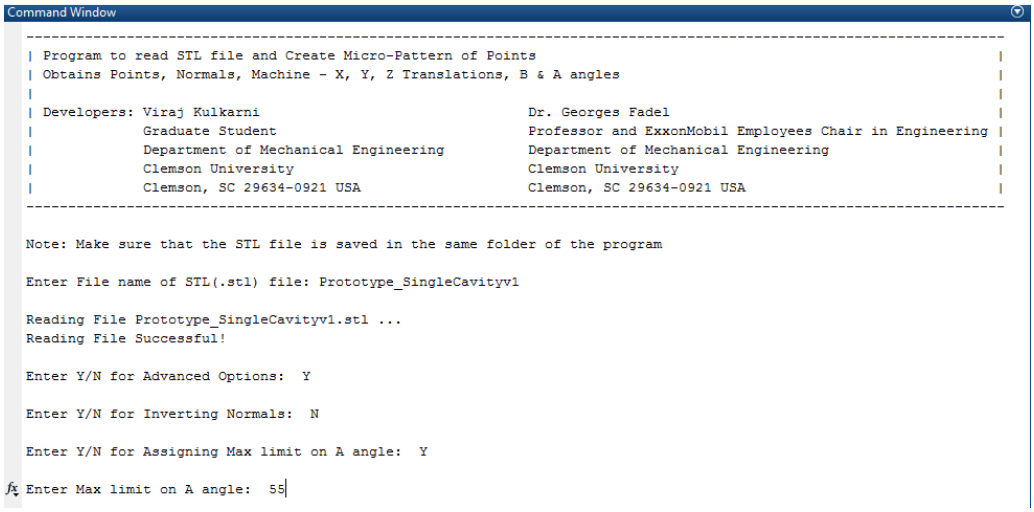


For this example the patterning is to be done on the inner or the concave side, so no inverting of the normals is necessary. The surface on which patterning has to be done on the convex side, normals have to be inverted. Even if the normals are not inverted, it can be noticed from the high A angle values or checking a point on the machine that the wrong side of the part is obtained. Then the program can be restarted inverting the normals.

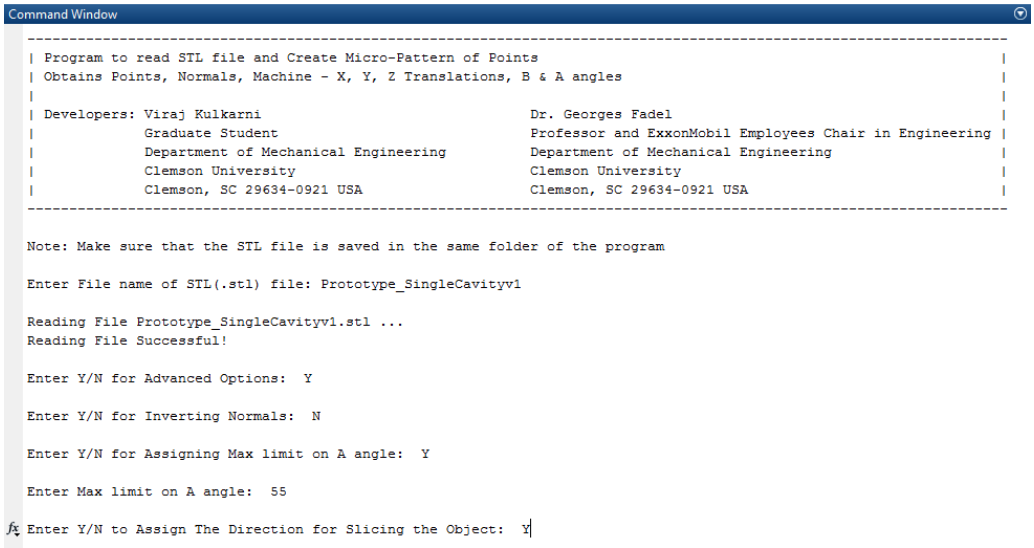
8. The next option is to assign Maximum allowed limit on the A angle.
   Enter 'y' if the limit has to be imposed.



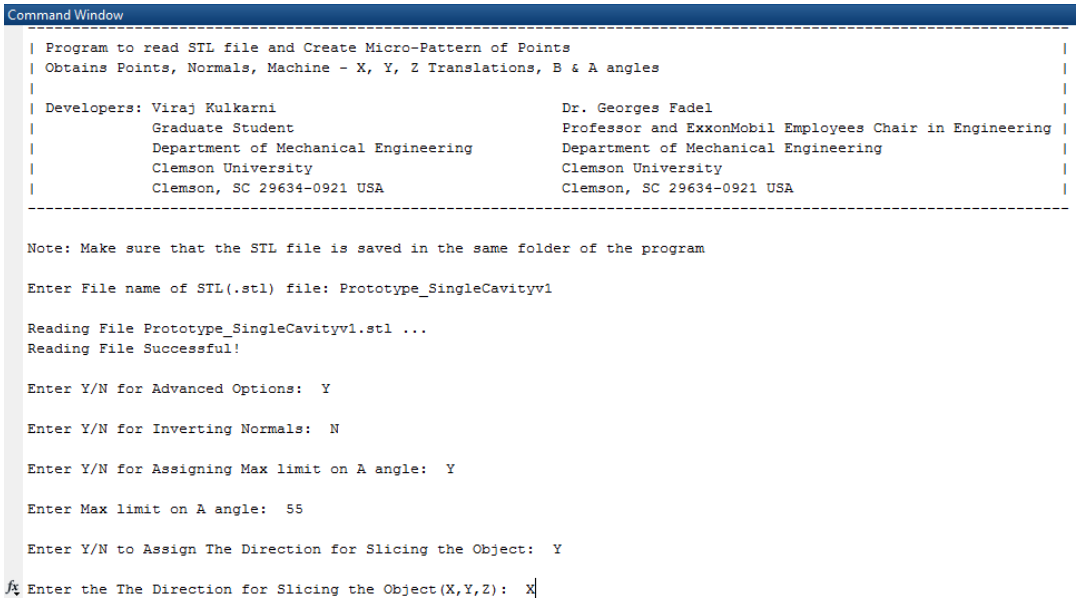
9. Enter the value for Maximum allowed value of A angle



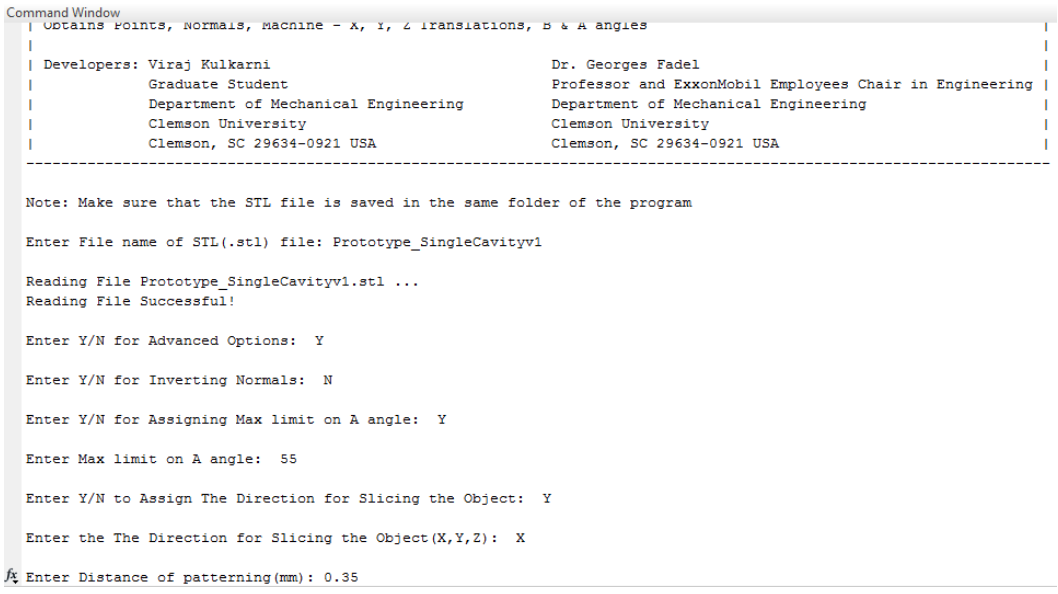
10. Enter Y for assigning direction of slicing or N for default 'X Axis'



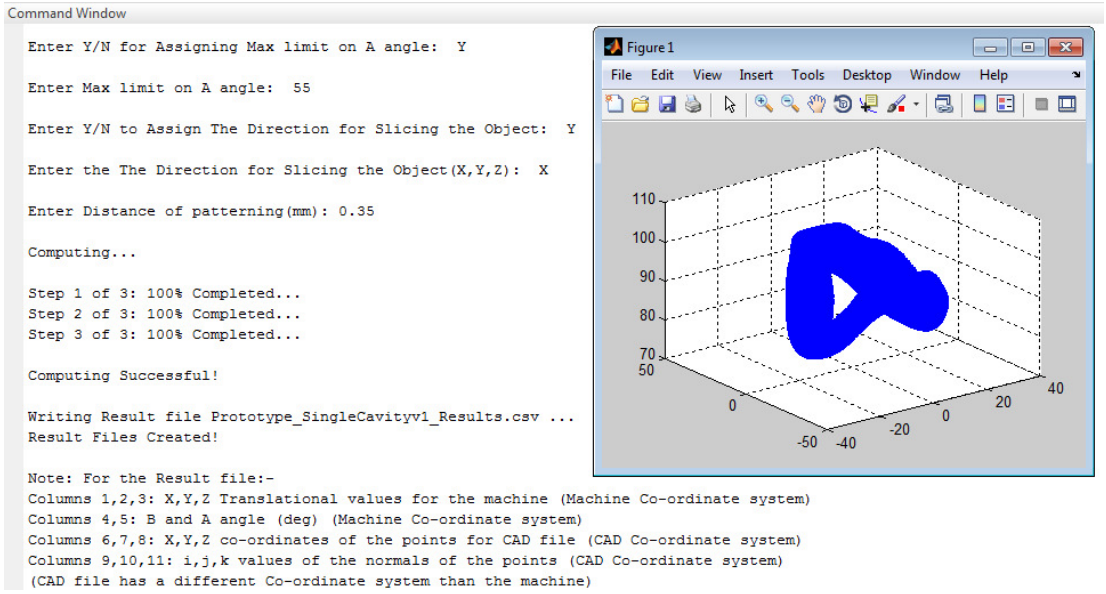
11. Enter X, Y, Z for assigning X-axis, Y-axis or Z-axis respectively for Slicing Direction



12. Enter the distance of applying the pattern in mm



13. The program will start computing and will display its status till the results are obtained.



14. The Result file will be created in the same location.

    For the Result file:-

    Columns 1,2,3: X,Y,Z Translational values for the machine (Machine Co-ordinate system)

    Columns 4,5: B and A angle (deg)(Machine Co-ordinate system)

    Columns 6,7,8: X,Y,Z coordinates of the points for CAD file (CAD Co-ordinate system)

    Columns 9,10,11: i,j,k values of the normals of the points (CAD Co-ordinate system)

    (CAD file has a different Co-ordinate system than the machine)