Fall 11-2022

Manufacturability and Analysis of Topologically Optimized Continuous Fiber Reinforced Composites

Jesus A. Ferrand
Embry-Riddle Aeronautical University, FERRANJ2@my.erau.edu

Manufacturability and Analysis of Topologically Optimized Continuous Fiber Reinforced Composites

By

J. A. Ferrand

A Thesis Submitted to the Faculty of Embry-Riddle Aeronautical University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Aerospace Engineering

November 2022

Embry-Riddle Aeronautical University

Daytona Beach Florida

Manufacturability and Analysis of Topologically
Optimized Continuous Fiber Reinforced Composites

By

J.A. Ferrand

This Thesis was prepared under the direction of the candidate's Thesis Committee Chair, Dr. Ali Y. Tamijani, Department of Aerospace Engineering, and has been approved by the members of the Thesis Committee. It was submitted to the Office of the Senior Vice President for Academic Affairs and Provost, and was accepted in the partial fulfillment of the requirements for the Degree of Master of Science in Aerospace Engineering.

THESIS COMMITTEE

Chair, Dr. Ali Y. Tamijani	Member , Dr. Alberto W. Mello
Member, Dr. Jeff R. Brown	
Graduate Program Coordinator, Dr. Hever Moncayo	Date
Dean of the College of Engineering Dr. James W. Gregory	Date
Associate Provost of Academic Support, Dr. Christopher Grant	Date

ACKNOWLEDGEMENTS

To Dr. Ali.Y. Tamijani, for giving me purpose and pushing me to greater heights.

To Dr. Magdy S. Attia and Dr. Richard J. Prazenica for getting me into the Ph.D. program.

To Dr. Pamela L. Daniels for her guidance and assistance through the years.

To the composers, Mick Gordon, Frank Klepacki and Jim Johnston, and the bands Tool, Avenged Sevenfold, Disturbed, and Slipknot, for collectively creating the majority of the Heavy Metal that went into this thesis and certainly the future work that will be derived from it.

This research was supported by the Office of Naval Research (ONR) under the award number N000142012683 with program manager Dr. Anisur Rahman. I wish to thank ONR for its support.

ABSTRACT

Researchers are unlocking the potential of Continuous Fiber Reinforced Composites for producing components with greater strength-to-weight ratios than state of the art metal alloys and unidirectional composites. The key is the emerging technology of topology optimization and advances in additive manufacturing. Topology optimization can fine tune component geometry and fiber placement all while satisfying stress constraints. However, the technology cannot yet robustly guarantee manufacturability. For this reason, substantial post-processing of an optimized design consisting of manual fiber replacement and subsequent Finite Element Analysis (FEA) is still required.

To automate this post-processing in two dimensions, two (2) algorithms were developed. The first one is aimed at filling the space of a topologically optimized component with fibers of prescribed thickness. The objective is to produce flawless fiber paths, meaning no self-intersections, no tight turns, and no overlapping between fibers. It does so by leveraging concepts from elementary geometry and the Signed Distance Function of a topologically optimized domain. The manufacturable fiber paths are represented using Non-Uniform Rational Basis Splines, which can be readily conveyed to a 3D-printer as

The second algorithm then calls a meshing routine to spatially discretize the topologically optimized domain. It takes input from the first algorithm to automatically create and append, orientations and material flags to the spatial elements produced by the meshing routine. Finally, it generates output that is then input to FEA software. The software is written in the C-programming language using the PETSc library. A load case is validated against MSC NASTRAN.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

AABB	Axis-Aligned Bounding Box
AM	Additive Manufacturing
CFRC	Continuous Fiber Reinforced Composite
LEE	Least-Eccentric Fillet
LIP	Lagrange Interpolating Polynomials
TO	Topology Optimization

$C(u)$	Rational Curve
C_{ijkl}	Stiffness Tensor
e	Eccentricity
M	Polyline Midpoint
n	Unit normal vector
$N_{i,p}$	"Basis spline of degree p at interval i"
O_i	Polyline offset
p	Degree of Basis spline
P	"Point"
R	Fillet Radius
S_{ijkl}	Compliance Tensor
σ_{ij}	Stress Tensor
ε_{ij}	Strain Tensor
t_i	Parameter of line segment i
u	Parameter variable
w_i	NURBS weight at point i
z_i	Parameter of a line

1. INTRODUCTION:

Additive Manufacturing (AM) is a broad label that applies to all construction techniques whereby a component is created by appending material together. This contrasts the rather traditional subtractive manufacturing whereby a component is extracted from a larger block by removing material from said block. However, "AM", the label, nowadays is almost exclusively colloquially used to refer to 3D printing. AM, broadly speaking, is being touted for its ability to reduce waste, save energy, and produce components that its subtractive counterpart cannot [1].

Within AM is a subset, the additive manufacturing of composite materials in which components are made by appending composite materials together. For example, take a traditional unidirectional laminate layup (Figure 1). Yet, within AM of composite materials is yet another subset, that of Continuous Fiber-Reinforced Composites (CFRC). This last subset of CFRC is the subject of this work. In the AM of CFRC, 3D printing techniques are used to lay fibers that attain local orientations as opposed to global ones. With that being said, the discussion of AM is now narrowed to 3D printing only, that is, the acronym CFRC is taken to mean "3D-printed continuous-fiber reinforced composite" (see Figure 2).

The motivation for using CFRCs to eventually phase-out currently used materials stems from the ever-present quest of increasing the strength-to-weight ratio of a component [2]. This is because increases in strength-to-weight correlate with decreases in costs due to maintenance and fuel usage in vehicles. The trend began with the replacement of raw metals (i.e., iron) with metal alloys (i.e., steels, aluminum alloys, titanium alloys). Metal alloys exhibit superior strength to raw metals but remain substantially heavy (like the former). The elastic constitution of metals is *isotropic*, meaning that the response to stress is invariant with orientation [2]. The layman should

take this to mean that the direction in which an isotropic substance is stressed is irrelevant to the mechanics of deformation.

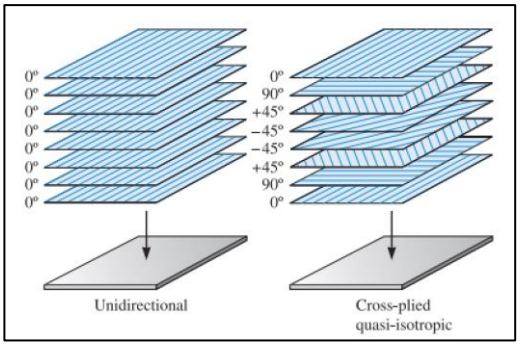


Figure 1: Laminate layup. Taken from [3]

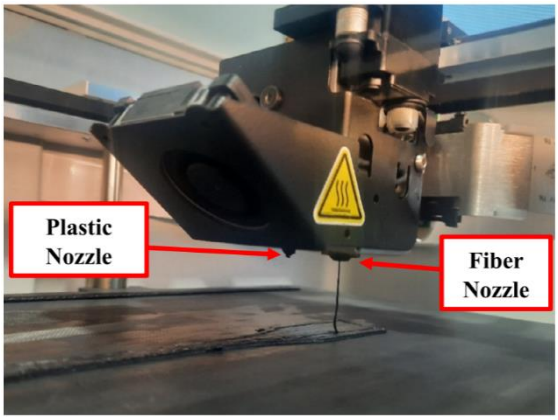


Figure 2: A 3D printer nozzle depositing molten plastic. Taken from [4].

1.1 Limitations of Isotropic Materials (Namely Metals)

The isotropy of metal alloys limits the efficiency of an all-metal component because there is no way to address localized stresses in a component via the substance itself. The only way is to add structural supports (which inherently make the structure heavier) or through redesigning the geometry of the component. For this reason, industries like the automotive and aerospace ones have historically leveraged unidirectional composites to make their vehicles lighter [5].

This is because unidirectional composites have an *anisotropic* elastic constitution, meaning that the direction in which the substance is stressed affects the mechanics of deformation (see Figure 3). This allows designers to tune the orientation of the substance to address localized stresses thus, making it possible to avoid structural reinforcements [2]. This benefit has led the aerospace industry to historically benefit from substantial savings in weight, number of components and Man Maintenance Hours per Flight Hour (MMH/FH) as shown in Figure 4.

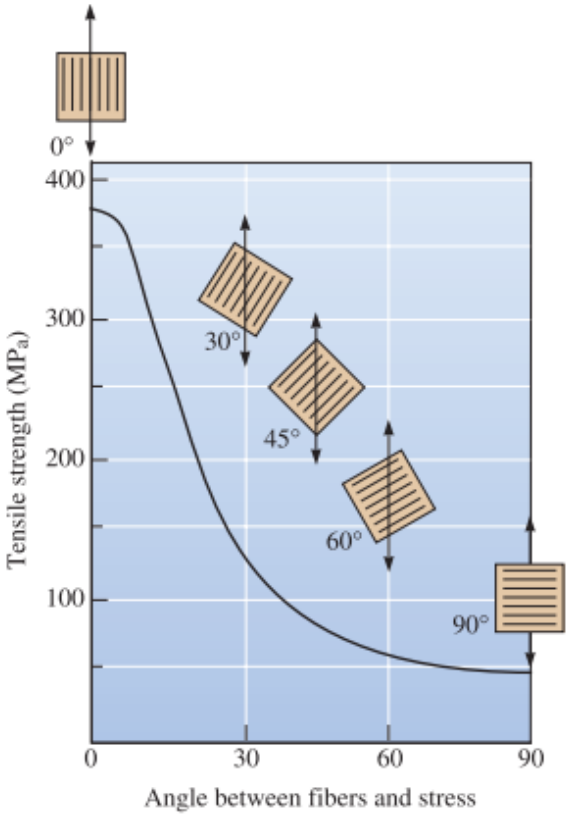


Figure 3: Strength dependency of unidirectional lamina with orientation. Taken from [3]

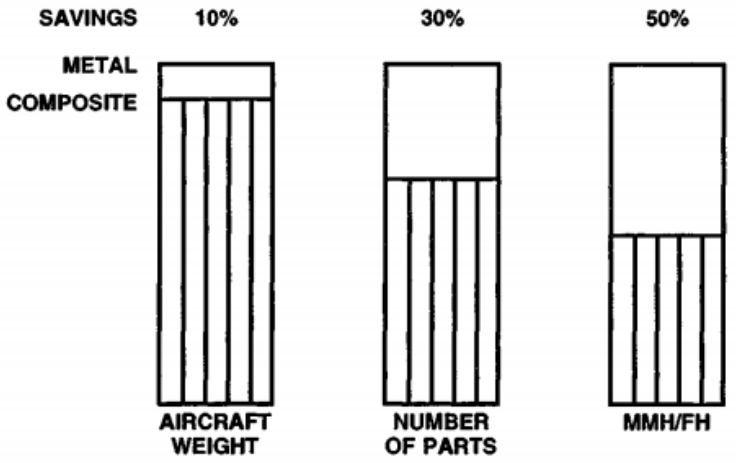


Figure 4: Relative reductions in cost-related aspects of aircraft during early adoption of composites. Taken from [2] .

1.2 Structural Superiority of Continuous Fibers Over Unidirectional Ones

Unidirectional composites, however, are limited by what is in their name: A single direction. If a designer wants to strengthen a local portion of a component via tuning of the one global direction of a unidirectional fiber, they may inadvertently compromise the strength at a different location. This issue is addressed by CFRCs, as the fibers can now attain local orientations that can address all the component's localized stresses (see Figure 5). For this reason, CFRCs are said to produce tailored performance [4].

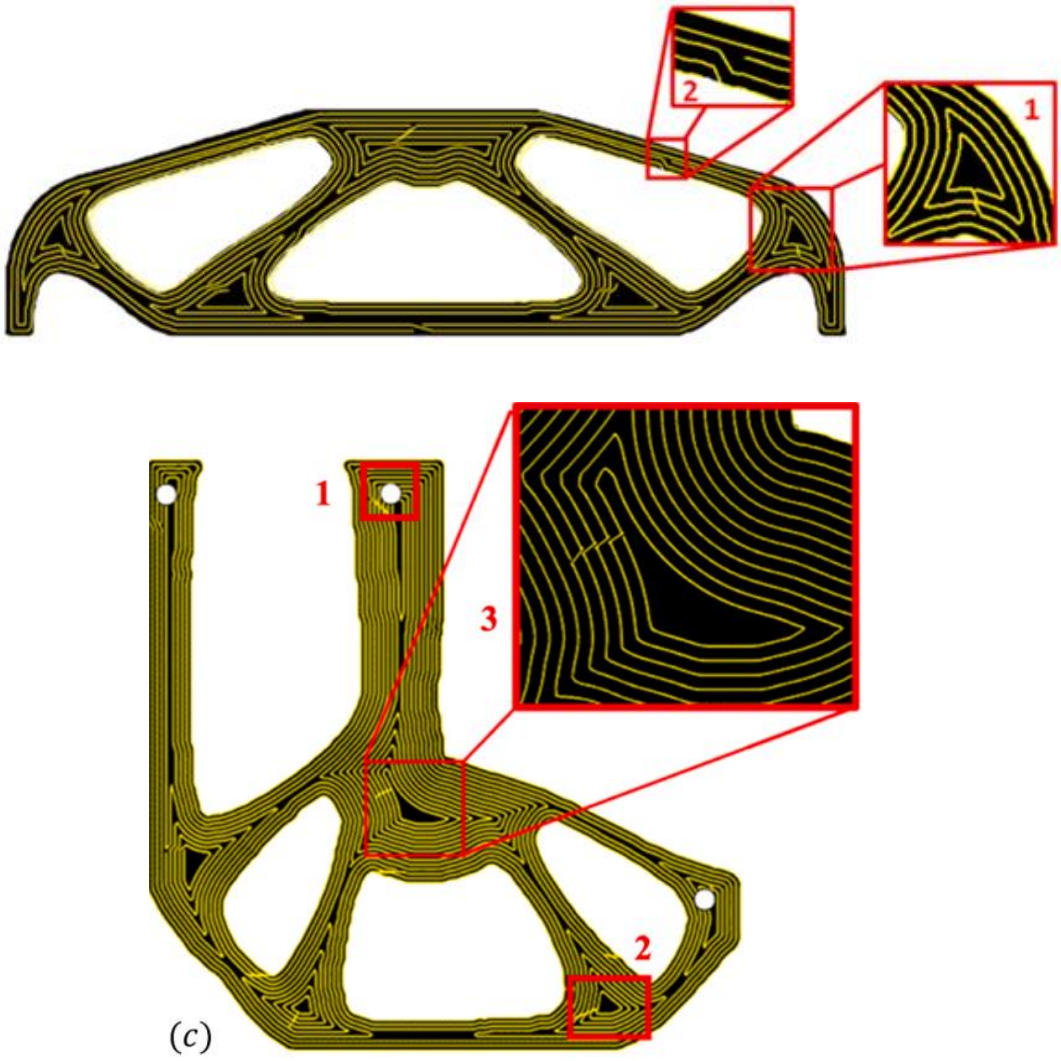


Figure 5: TO-CFRC components. Taken from [4]

In general composites are lighter and stronger (on a per pound basis) than metals. However, advances in material science are bringing the raw strengths of fibers used in CFRC closer to those of metal alloys. Thus, CFRC are presently, particularly relevant in the furthering of the automotive and aerospace industries. [6]

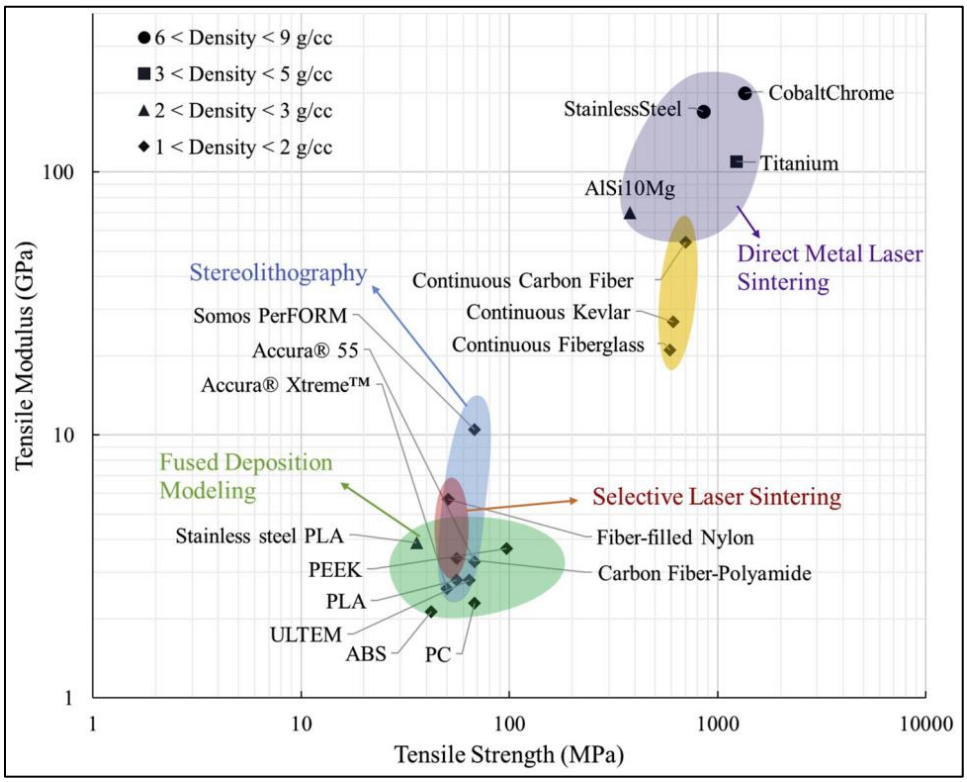


Figure 6: Absolute tensile strength and stiffness of select materials. [7]

Therefore, CFRCs are the next type of material in the quest of increasing the strength to weight ratios. They are, however, not mainstream in the manufacturing industry yet. The reason is two-fold: Firstly, issues stemming from thermoplastic effects on the elastic properties of the CFRC as most are manufactured via melting process. Secondly, issues stemming from the anisotropy of CFRCs [6]. Here is a short list of some but not all issues that are currently holding back the mainstream adoption of additively manufactured CFRC:

Table 1: Some issues holding AM-CFRC back

Manufacturing-Related	**Anisotropy-Related**
1. Voids. 2. Glass transition 3. Support structures 4. Imposition of manufacturing constraints on component design.	1. Determination of local fiber orientations. 2. Unintuitive failure points. 3. Difficulty in factoring local anisotropy into analysis. 4. Analysis is inherently expensive.

This work is predominantly concerned with addressing point 4 of the "manufacturing-related" issues and point 3 of the "anisotropy-related" issues. However, discussion of points 1, 2, and 4 of the "anisotropy-related" column is furthered as they are closely related. For information on the "manufacturing-related" problems the reader is referred to the work by [8] for information avoid the adverse effects of voids, to the work by [5] for information about the "Glass transition" and other thermal physics-related effects, and the work by [9] for information about the support structures.

1.3 It All Stems from Topology Optimization

The issue of determining the fiber orientations is being addressed by those investigating the field of topology optimization. The technology is rather mature for the optimization of isotropic materials and can be traced back to at least the year 1988 [10]. The natural objective function for optimizing structural design is the so-called compliance, a measure of how much a structure deforms under loading [11]. Minimization of this quantity alone leads to designs that do not satisfy stress-based constraints, such as prevention of structural failure. To bring topology optimization closer to practicality stress-based design optimization has been pursued [12].

Topology optimization, however, becomes more challenging when the material is anisotropic (like composite materials) and when one considers the thermoplastic physics of the most common materials used to additively manufacture CFRCs. This issue is twofold: firstly, the topology optimization must now factor additional design variables (the orientations of the fibers) which increases computational cost, and the material is deposited only after melting it. This melting is responsible for the intermediate structure of a component that is being manufactured to sag under its own weight [9]. The deformation of the component due to sagging can be avoided by installing intermediate support structures.

For this reason, researchers are attempting to mitigate the effect of the sagging phenomenon into their topology optimization schemes by adding constraints and objective functions that prevent sagging and reduce the amount of support structures. In the work by [1], an objective function that minimizes the overhang angle of surfaces is investigated, and a constraint that guarantees that support structures are accessible (for removal) is included. Another work, that by [11] explored formulating an objective function based on the volume of the support structures needed to address sagging. At any rate, because of all issues discussed so far, topologically optimized designs currently require substantial post-processing, which is more akin to a "reckoning" than a "polishing".

1.4 Topology Optimization Meets Reality

Firstly, the topologically optimized shape must now be filled with fibers, which have thickness and potentially sharp corners at their endpoints. The sharp corners create discontinuous filling of the space allocated to the component, which then leads to the creation of voids [13]. The thickness of the fibers makes it so that an optimized fiber path must be locally obfuscated to accommodate for the printer's turn radius. The thickness can also require obfuscation of a fiber path if the latter

gets too close (but not quite to) self-intersecting. Clearly, the thickness limits the extent to which the optimized design can be preserved during the manufacturing process [14]. And while these defects can be patched by filling the ensuing voids, the fact remains: The design has been altered from what topology optimization output.

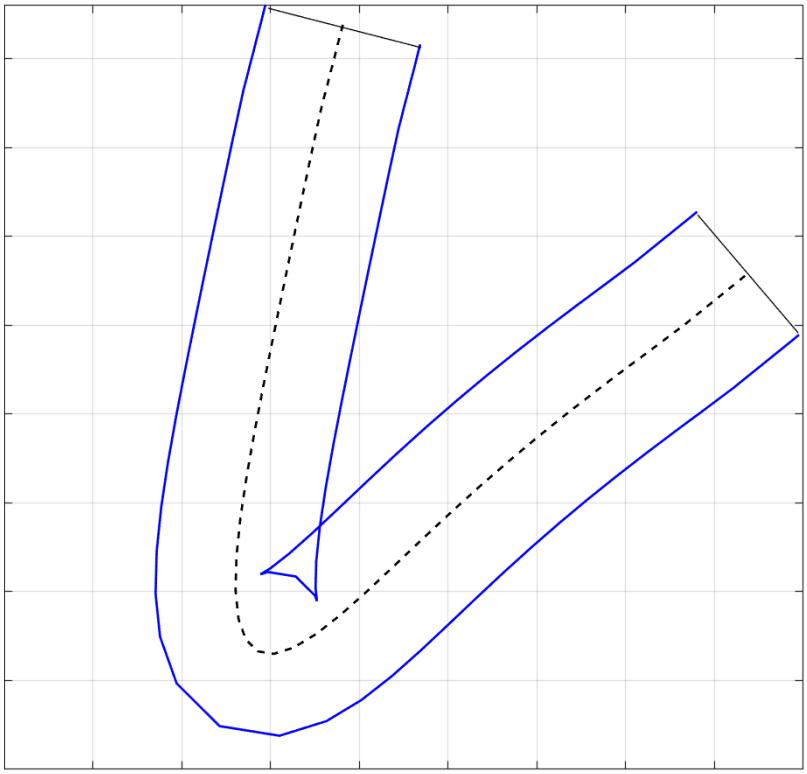


Figure 7: Example of a tight turn

Common design features that are not manufacturable are *tight turns* and *short fibers*. Tight turns are changes in a path that would make the 3D printer's nozzle deposit material over the same region multiple times in a *local* fashion. Short fibers are those with length either less than the nozzle radius or less than the nozzle's displacement resolution (i.e., the shortest step that the nozzle can take). *Overlap* is another issue whereby fibers are placed too close to each other such that the printer's nozzle deposits material over the same region multiple times in a *global* fashion [13].

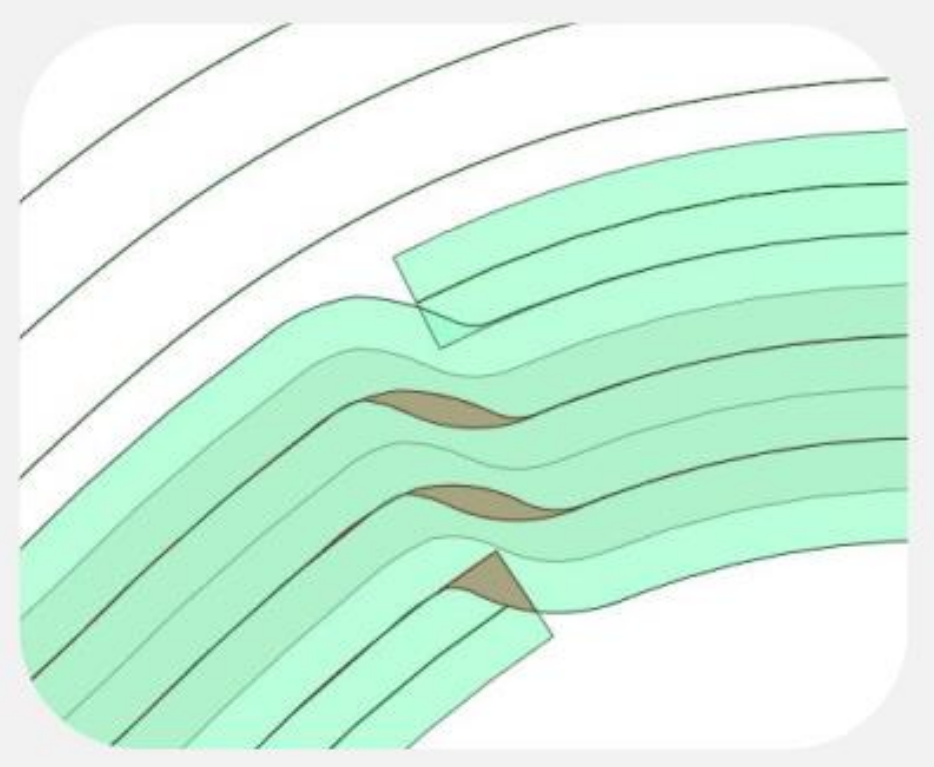


Figure 8: Example of overlapping fibers (Courtesy of 9T Labs).

1.5 Analysis of Continuous Fibers is Expensive

Because of imposing *manufacturability*, a standalone Finite Element Analysis (FEA) is required to assess whether the *manufacturable* component would indeed satisfy the original design requirements [7]. Speaking of FEA, topology optimization already requires intermittent FEA as an intermediate step [10] [14] [12]. This is because fundamentally, topology optimization entails iteratively tuning the shape and measuring the response in the compliance. To find the latter, FEA is required every time the shape is tuned. To make matters worse, the FEA required for CFRC is substantially more intensive than that for isotropic materials or that for unidirectional composites.

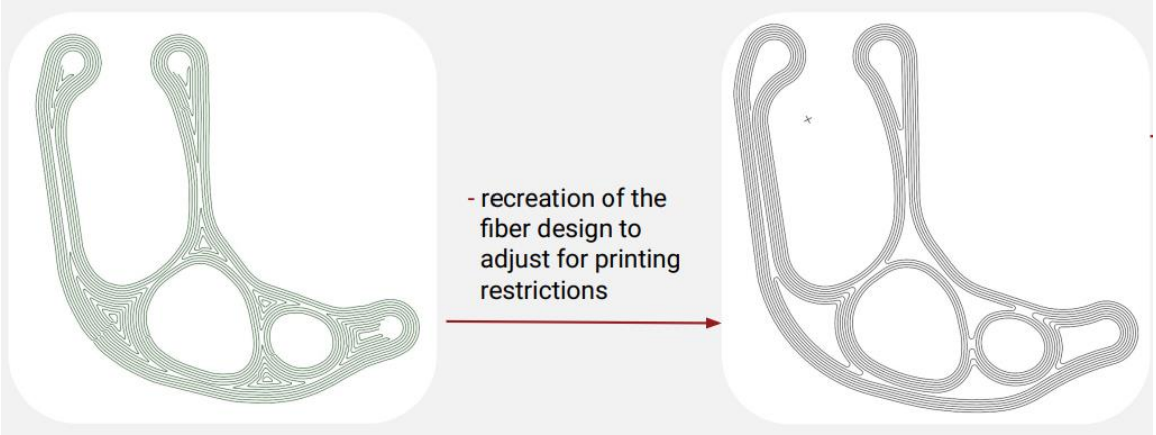


Figure 9:A topologically optimized design that was severely obfuscated to ensure manufacturability (Courtesy of 9T Labs).

This is because of differences in the assembly of the global stiffness matrix in the FEA for continuous fibers versus that in FEA of unidirectional composites and isotropic materials. For isotropic materials, a single stiffness tensor is constructed and applied to all finite elements. For a unidirectional composite, the stiffness tensor is also created but must be transformed to account for the orientation of the global fiber (albeit, once). For a CFRC, the stiffness tensor must be transformed to account for the local orientation of every element [2]. The transformation operation is particularly expensive because in practice it must be deployed as a triple matrix product. The sizes of these matrices are 3 by 3 in 2D simulations, and 6 by 6 in 3D simulations [13].

1.6 Creating the Input File is Cumbersome

Despite the computational cost, analysis of CFRC is feasible. However, generating the fiber orientations and appending them to the finite elements is not fully automated. State of the art software like MSC Nastran can accommodate analysis of CFRC if an appropriate Bulk Data File (BDF) is generated [15]. For example, one can define “PCOMP” entries in the BDF file and then include one angle relative to the global axis per element. MSC Nastran can then complete the FEA but the entire premise hinges on the generation of the BDF file. This was the case for [7] when

they reconciled experimental test results of a CFRC coupon with FEA. They had to generate a custom BDF writer on a relatively simple, structured domain. Doing so for an arbitrary domain is not mainstream.

1.7 Objectives of this Research

This work does not discuss the inner details of topology optimization, rather it is aimed as expediting the conversion from an optimized design to a manufacturable one. That is, this work is concerned with seeking to automate the tedious post-processing that comes after a topology optimization scheme. The post-processing has two major steps: fiber placement, and FEA input file generation, both of which are not yet fully automated in a mainstream way. The objectives of this work are, therefore, as follows:

1.7.1 Automatic Generation of Manufacturable Fiber Paths

The fiber placement is all about thickening the fiber paths to reflect their actual shape. The fibers are not directly obtained from a topology optimization scheme, rather, from a mathematical entity called the Signed Distance Function (SDF). This is because topology optimization is not yet mature enough to automatically tune the fiber orientations. The SDF, however, is a very efficient initial guess for generating the fibers. All the nuances pertaining to manufacturable fiber placement are addressed in Algorithm 1 of this work.

1.7.2 Automatic Generation of FEA File for Manufacturable Component

Once the fibers are placed, the final structural integrity check is performed via Finite Element Analysis (FEA) of the manufacturable component. The creation of an input file for FEA as part of reconciliation between simulation and experimentation often entails manual preprocessing. This task ought to be automated.

1.7.3 Prototype a High-Performance FEA Script

Because the analysis of continuous fibers is expense and most commercial software do not yet have mature interfaces for analyzing CFRCs, the prospect of completing an automatic fiber angle generation software brings with it the question of how to analyze it. It quickly follows that some kind of interface between the first two algorithms and an FEA routine must be established. Furthermore, within the grander TO scheme, FEA is an intermittent step as part of the countless iterations in the former. Therefore, this work also explores the prospects of contributing towards the speedy analysis of CFRCs to possibly assist the field of TO in the future. This is accomplished via a script written in the C-programming language using the high-performance computing library, PETSc.

2. GENERATION OF MANUFACTURABLE FIBERS (ALGORITHM 1)

The objective of Algorithm 1 is to produce fiber paths that are flawless from the manufacturing perspective. This entails creating parametric curves (i.e., centerlines) that satisfy the following four (4) criteria. Firstly, the curves are *offset* by half of the intended filament thickness do not self-intersect. Secondly, the offset that constitutes the *inside* of the fiber does not make *tight turns* (minimum turn radius). Thirdly, no center line shall produce a *short fiber* (minimum print distance). Fourthly, no fiber shall overlap another.

The flow diagram for the manufacturability algorithm (which addresses the four (4) requirements just listed) developed in this work is shown in Figure 10 below. The steps are numerous, however, fundamentally it can be broken down into eight (8) concepts which are explained in greater detail through the remainder of this section.

1) Signed distance function and its level sets.
2) Representing the level sets as discrete curves.
3) Offsetting.
4) Interpolation noise suppression
5) Short fiber filtering.
6) Defect detection.
7) Defect correction
8) NURBS representation of fiber centerline.

2.1 Description

The essence of Algorithm 1 is that the SDF will be used to produce an initial shape for the inside of the thickened fibers. Due to the numerical nature of the SDF, an interpolation scheme is required to obtain the “insides” of the fibers. Due to numerical nuances, the initial “insides” are produced with unacceptable defects such as wiggles, tight turns, or shattered contours. In response

to said flaws, a repertoire of algorithms based on elementary geometry are deployed on the "insides" to produce acceptable curves. When the "insides" are corrected, they are offset outward by half of the fiber thickness to create an initial guess for the fiber centerline. This centerline, which at that stage consists of discrete *xy*-coordinates, is then used to construct a so-called NURBS curve, which is a standard mathematical artifact used to convey shapes. Said NURBS are the final product of Algorithm 1.

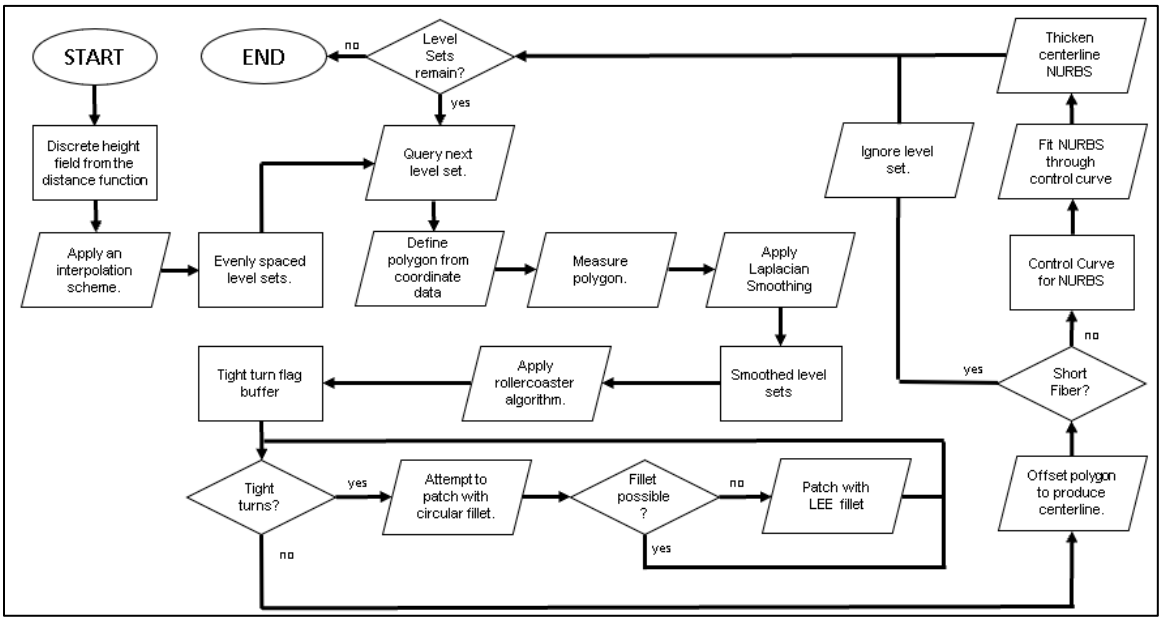


Figure 10: Flow diagram of fiber manufacturability algorithm

2.2 Level Sets of the SDF

In this work, it is assumed that a topology optimization scheme has been deployed to completion and that the final boundaries are provided as discrete sequences of *xy* coordinates. The layman can think of the boundaries as being either an *outer shape* or individual *holes* . Any component must have an all-enclosing boundary which the layman may refer to as the "outer shape." Any interior boundaries perceptible due to abscense of material are "holes" (see Figure 11

for an example). Together, the outer shape and the collection of holes (if any) constitute the *domain*.

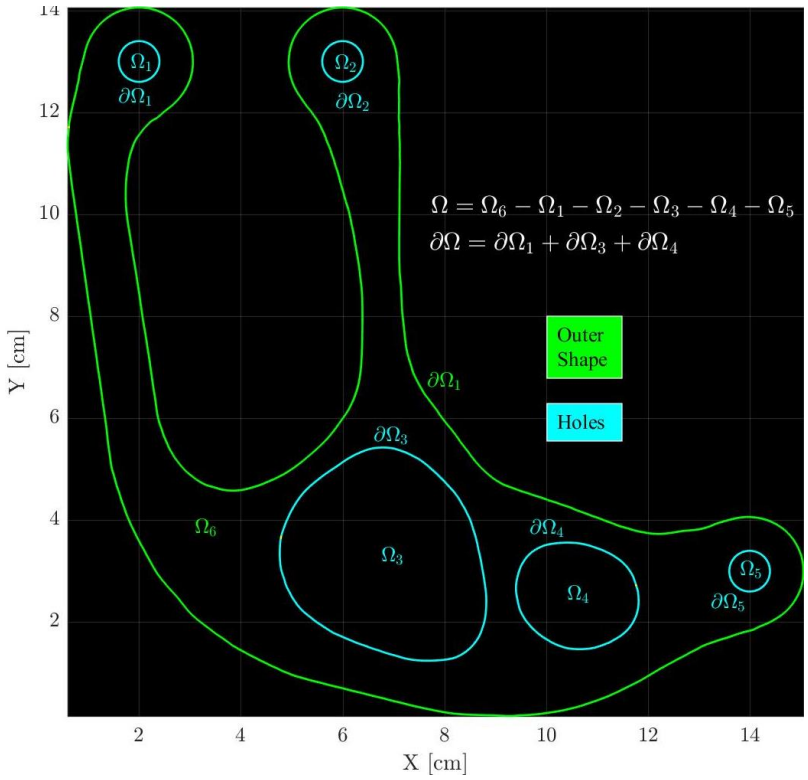
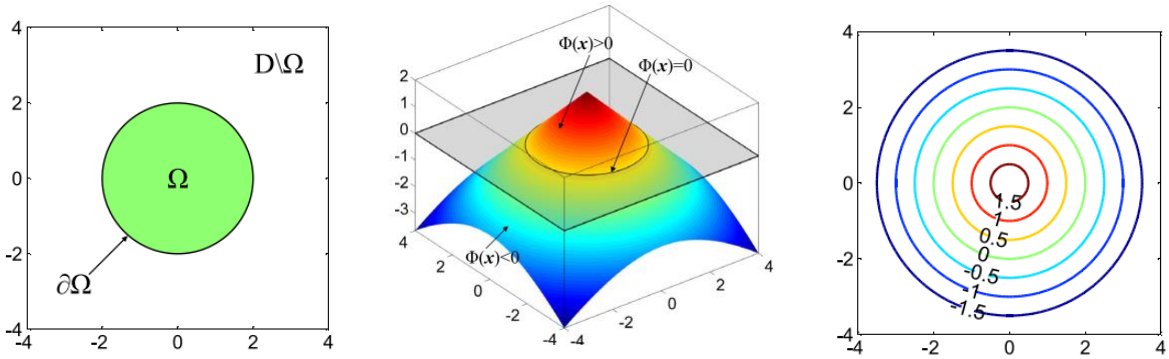


Figure 11: Outer shape vs. holes in the context of distinguishing boundaries.

2.2.1 The SDF in a nutshell

For the purposes of generating manufacturable fiber paths, the boundaries are used to generate the so-called Signed Distance Function (SDF). This is a scalar field that returns the *orthogonal* distance of any point inside the *domain* to the boundaries [14]. The SDF is defined for outside points also with the convention that inside points received a positive value whereas the outside points receive a negative value (see Figure 12.b). In this work, only the interior points are of concern.



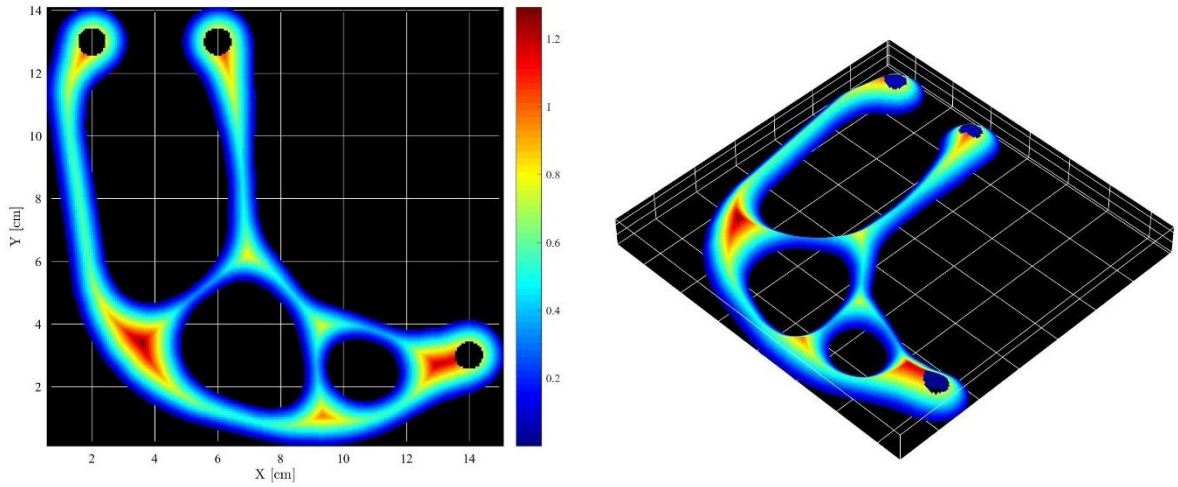
(a) (b) (c)

Figure 12: Signed Distance Function of a circular domain. Taken from [14]

In practice geometries vary in complexity. They can be as simple as the closed circular domain shown in Figure 12 or they can be composite in the sense that they are produced as the intersection or exclusion of other shapes. Figure 11, for example, showcases a composite domain consisting of inclusion as well as exclusion zones. The holes are exclusion zones always (they are never part of the domain). Because of this distinction, composite shapes are typically conveyed in terms of Boolean operations between geometric shapes such as “add”, “subtract”, “intersect”, etc.

2.2.2 Three-dimensional rendering of the SDF

The signed distance function can be graphed along a third axis to produce “mountain” like graphs like the one shown in Figure 12.b, or more pertinently to this work, the one shown in Figure 13. The layman should build the following intuition regarding the SDF: Imagine fronts emanating from the boundaries that offset inward at the same constant rate. At the places where the fronts coalesce, a peak is formed. The more distance the fronts get to travel before coalescing, and the more fronts involved in the collision, the greater the peak. As an example, use the colormap given in Figure 13 (a) to gauge the heights.



(a) (b)

Figure 13: (a): Scalar field of the SDF evaluated over the domain in Figure 11, units are [cm]. (b): A perspective projection of the L-shape shown to illustrate the heights.

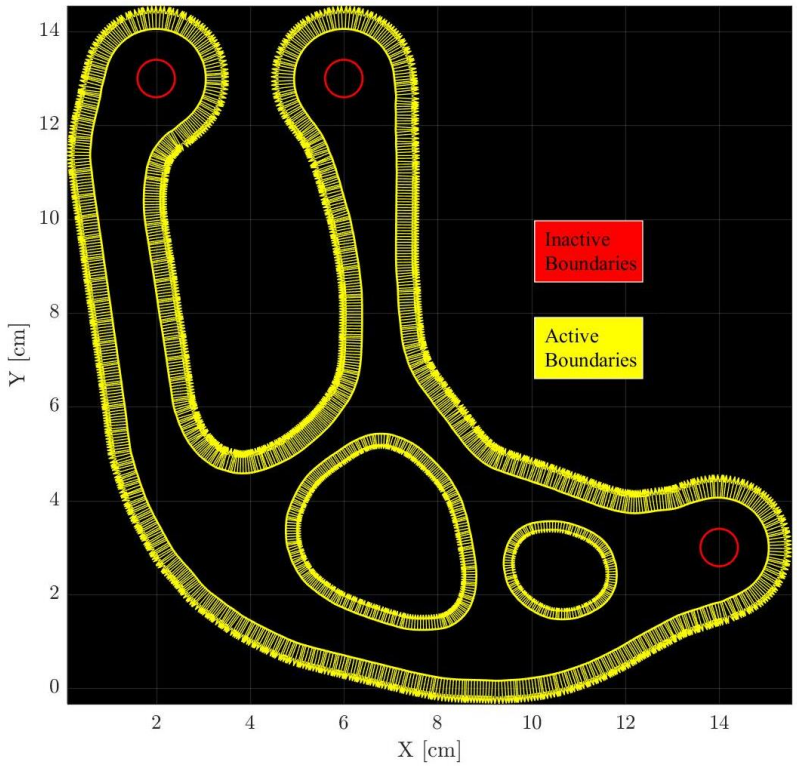


Figure 14: Active vs. inactive boundaries in the context of the signed distance function.

Note that the SDF requires the input of the offsetting boundaries. For example, the SDF shown in Figure 13 does not form circular peaks around the circular holes. This is because when

generating it, the boundaries of the circular holes were not included. Thus, the layman should think of the possibility of toggling the boundaries between states of activity and inactivity (see Figure 14). For purposes of manufacturability and performance, inclusion of the circular holes as active boundaries weakens the component because it introduces stress concentrations (hence they are toggled off).

2.2.3 Level Set Extraction

At any rate, once the SDF is evaluated, one can extract so-called *level sets*, contours that correspond to constant values of the SDF. These can be thought of as being obtained by slicing the SDF at equally spaced heights with a plane as shown (progressively) by the collection of Figure 15 through Figure 26. The objective of this slicing procedure is to compute the intersection of the cutting plane with the SDF at each query level. When this is done for all query levels, one obtains curves like the ones shown in Figure 12.c or more pertinently, the ones shown in Figure 27.

The exact intersection between the cutting planes and the SDF cannot be computed in practice because the SDF does not have an analytic solution for sophisticated shapes. Elementary shapes like *conic sections* (i.e., circles, ellipses, parabolas, and hyperbolas) have analytic solutions to the SDF, but arbitrary contours like the composite domain shown in Figure 11 do not. For this reason, a numerical scheme for generating the SDF is warranted. In this work, a stencil is created over the domain and the SDF is then made available (through third party code) as three-dimensional discrete data. The contours shown in Figure 27are obtained by calling MATLAB®'s "contour" function on the discrete data. This function by default applies a linear interpolation scheme to produce the approximate intersection of the cutting planes with the discrete SDF.

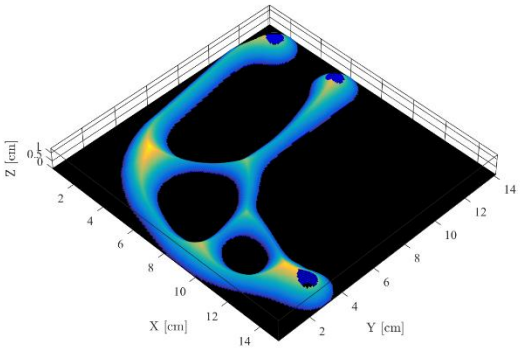


Figure 15: Level 0

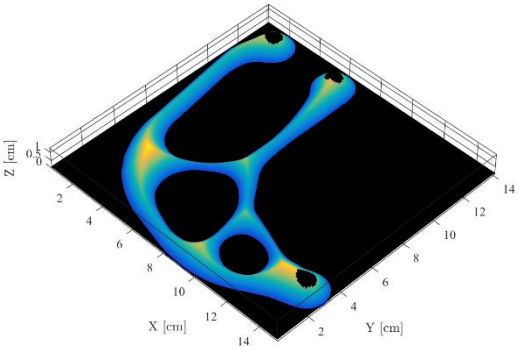


Figure 16: Level 1

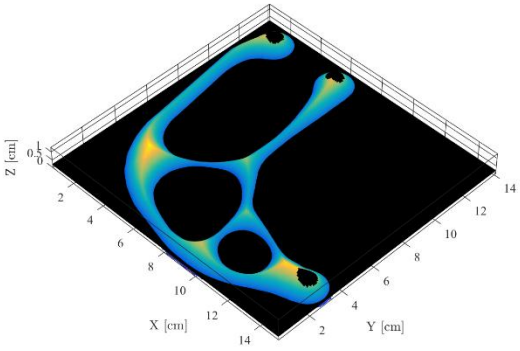


Figure 17: Level 2

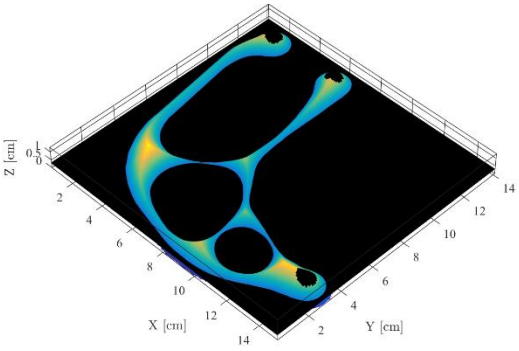


Figure 18: Level 3

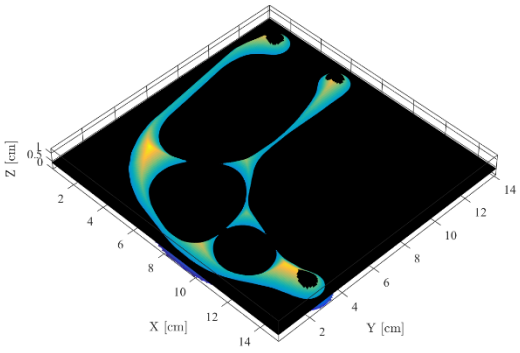


Figure 19: Level 4

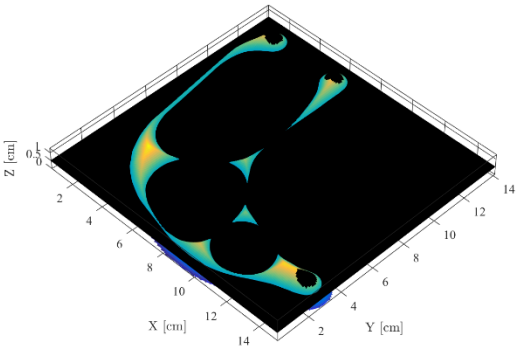


Figure 20: Level 5

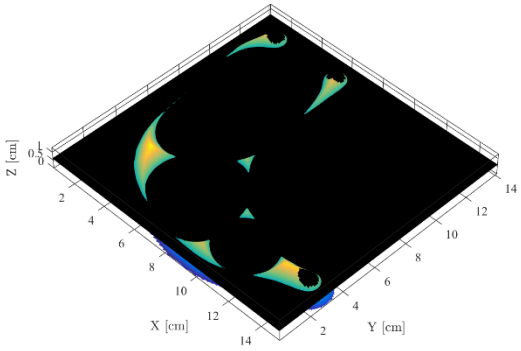


Figure 21: Level 6

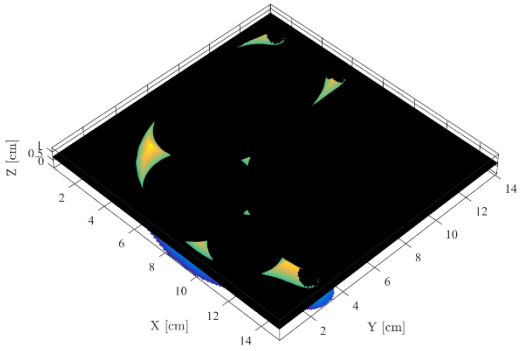


Figure 22: Level 7

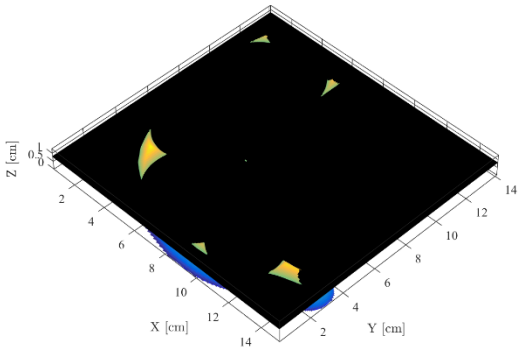


Figure 23: Level 8

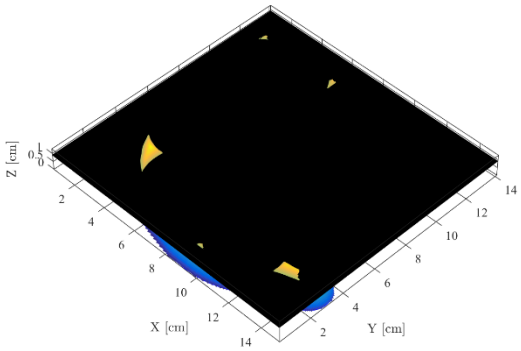


Figure 24: Level 9

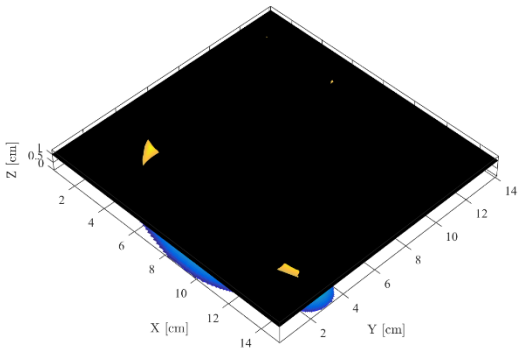


Figure 25: Level 10

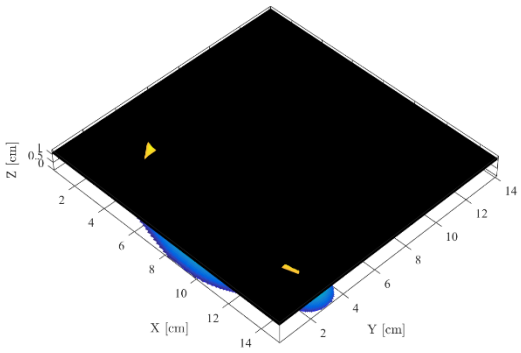


Figure 26: Level 11

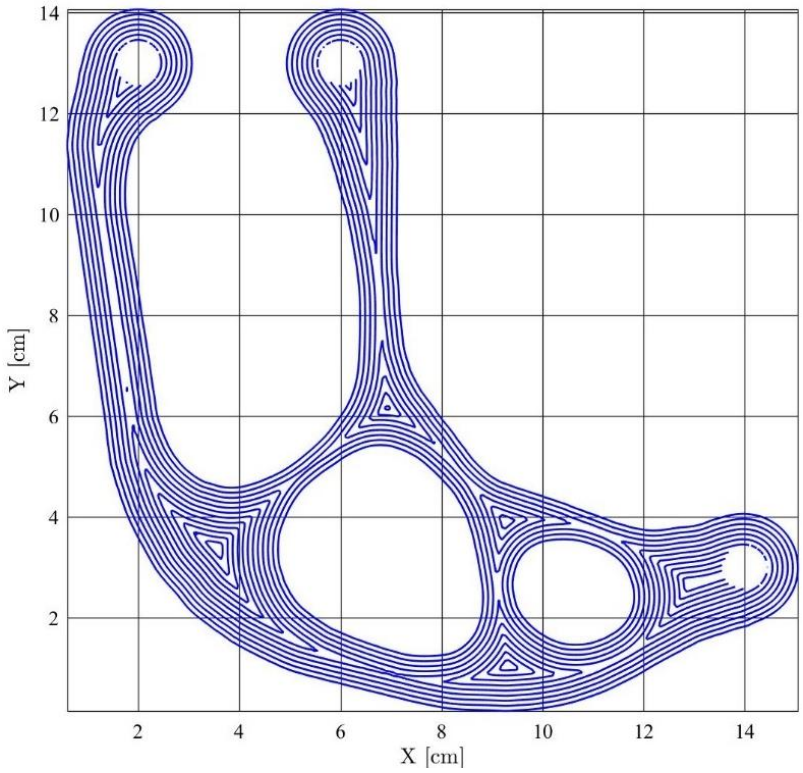


Figure 27: Level sets extracted from the SDF shown in Figure 13.

To conclude discussion of Concept 1, carefully inspect Figure 27. To the naked eye, the contours might seem reasonably accurate, however, zoomed-in views will reveal a vast array of issues. Figure 28 highlights regions where flaws are located (labelled “A” through “H”). Zoomed-in views are provided by Figure 29 through Figure 36. The bulk of these illustrated issues has a twofold classification: Interpolation noise, and interpolation obfuscation. The former issue is addressed in the discussion of Concept 4: Noise suppression, and the latter is partially addressed by the combined discussions of Concept 2, Concept 3, and Concept 5. The obfuscation issue is not fully addressed in this work and will be the subject of future research.

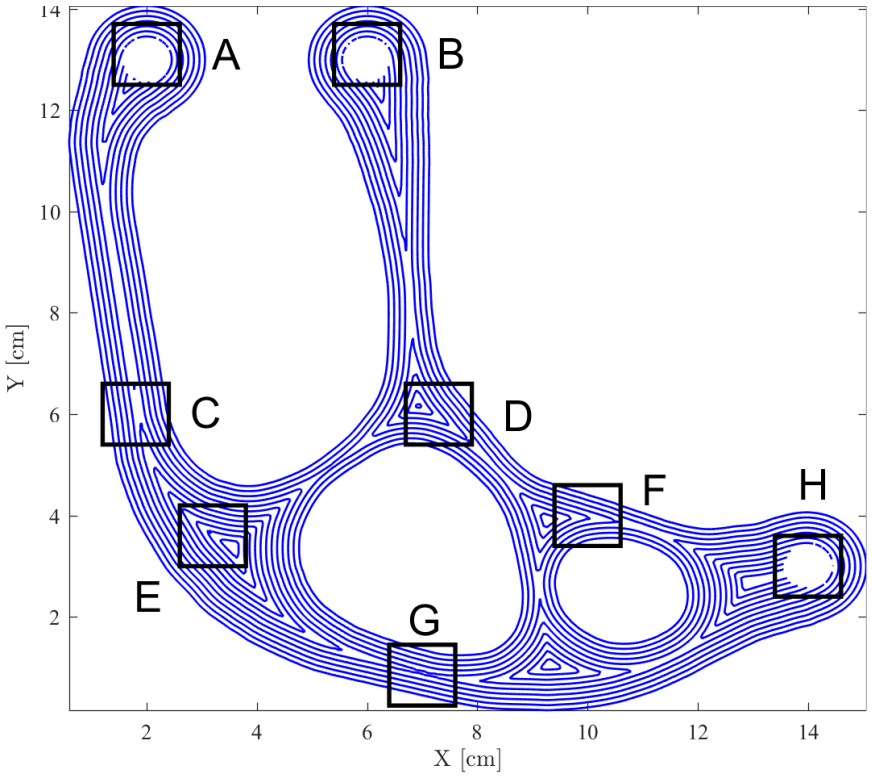


Figure 28: Sample flaws produces by the SDF.

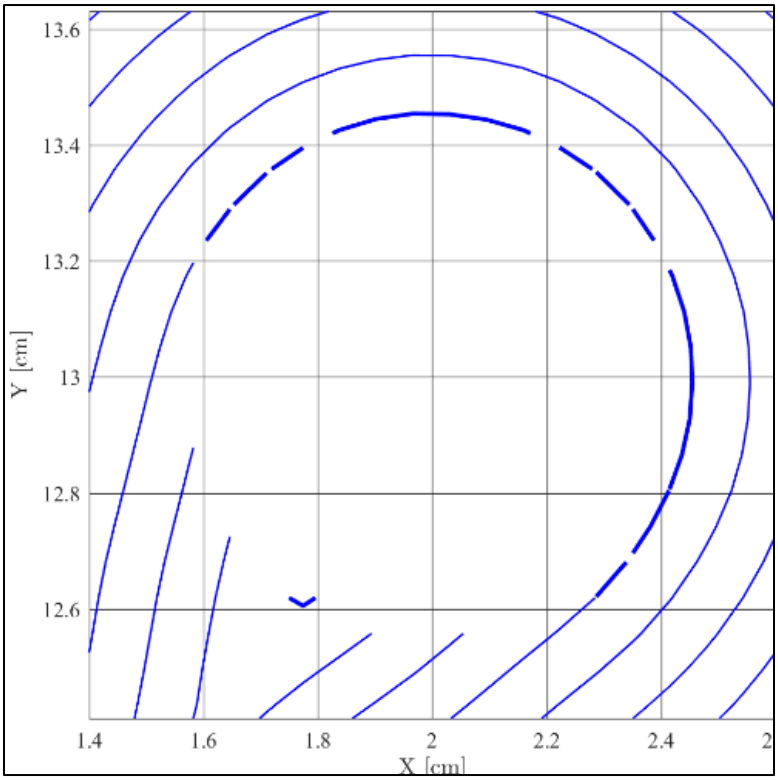


Figure 29: Zoom "A" into Figure 28

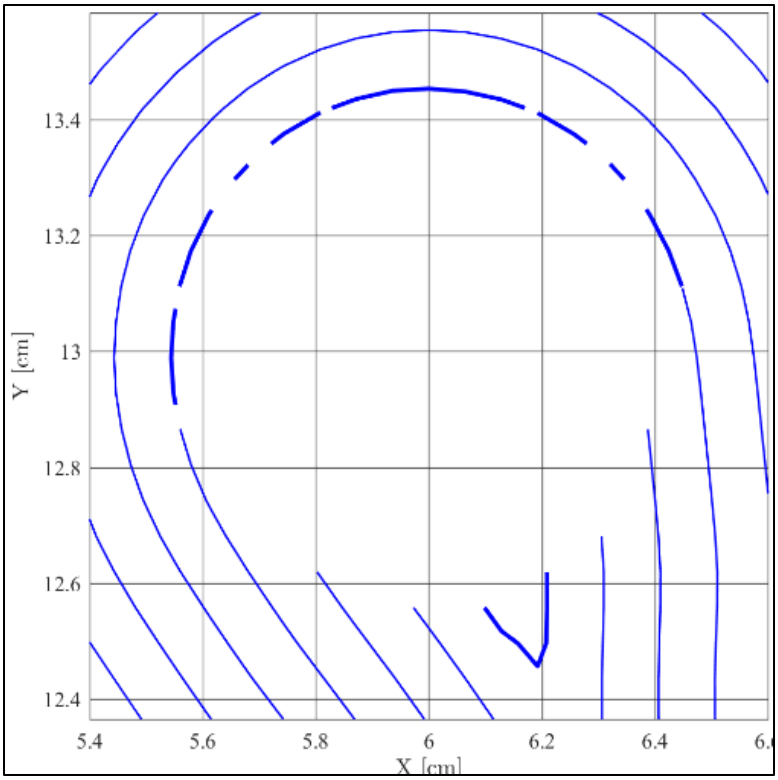


Figure 30: Zoom "B" into Figure 28

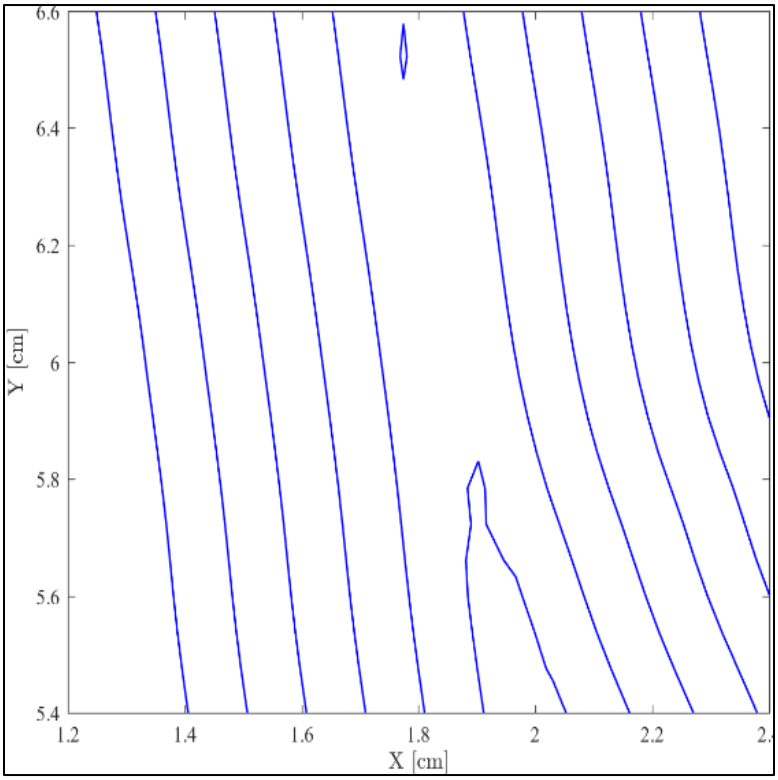


Figure 31: Zoom "C" into Figure 28

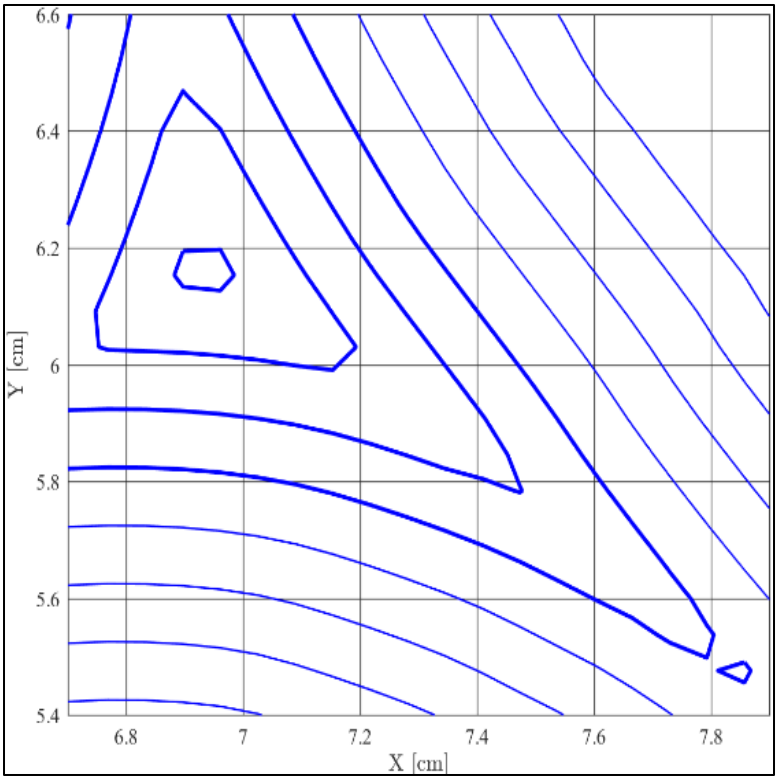


Figure 32: Zoom "D" into Figure 28

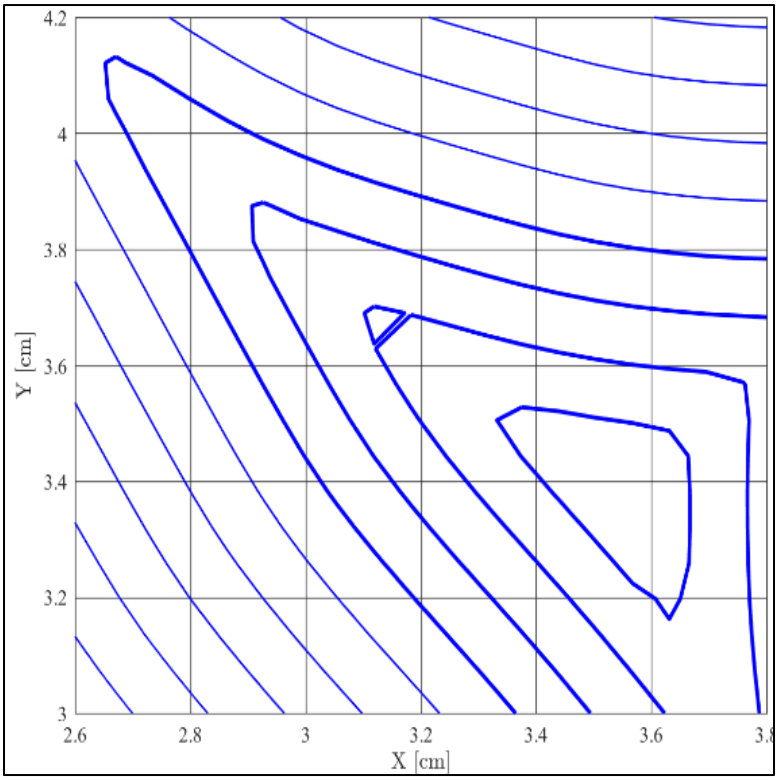


Figure 33: Zoom "E" into Figure 28

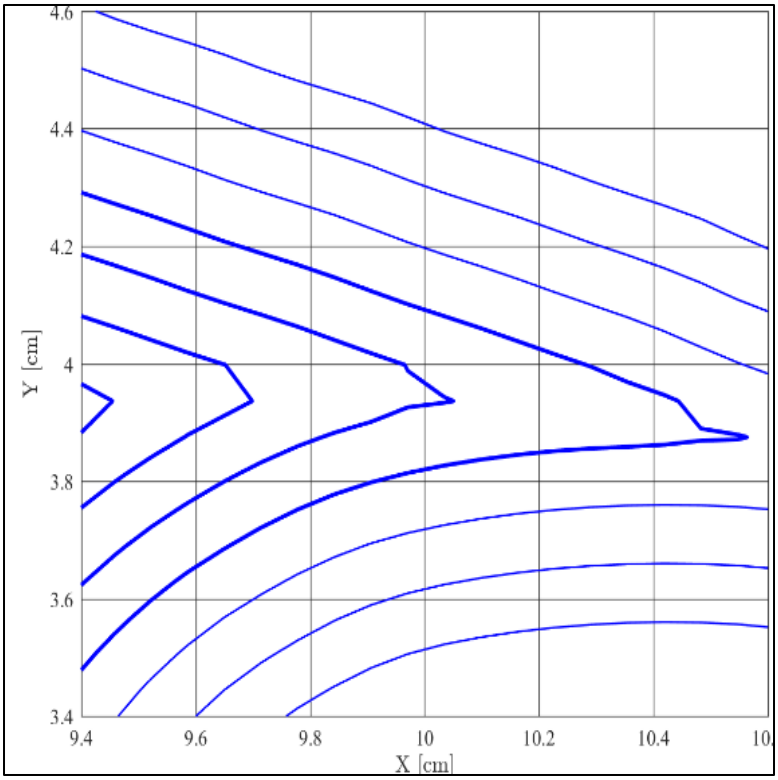


Figure 34: Zoom "F" into Figure 28

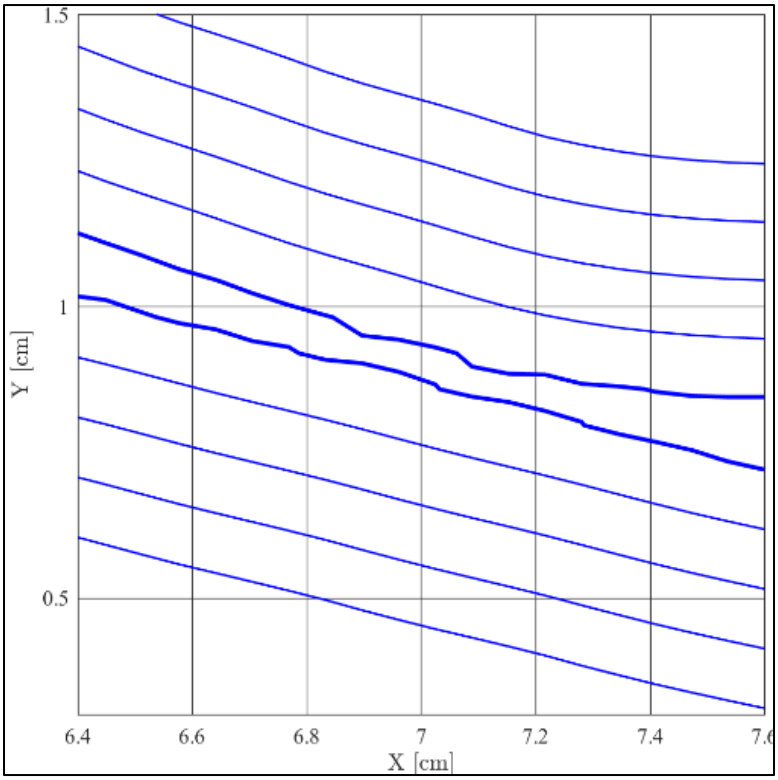


Figure 35: Zoom "G" into Figure 28

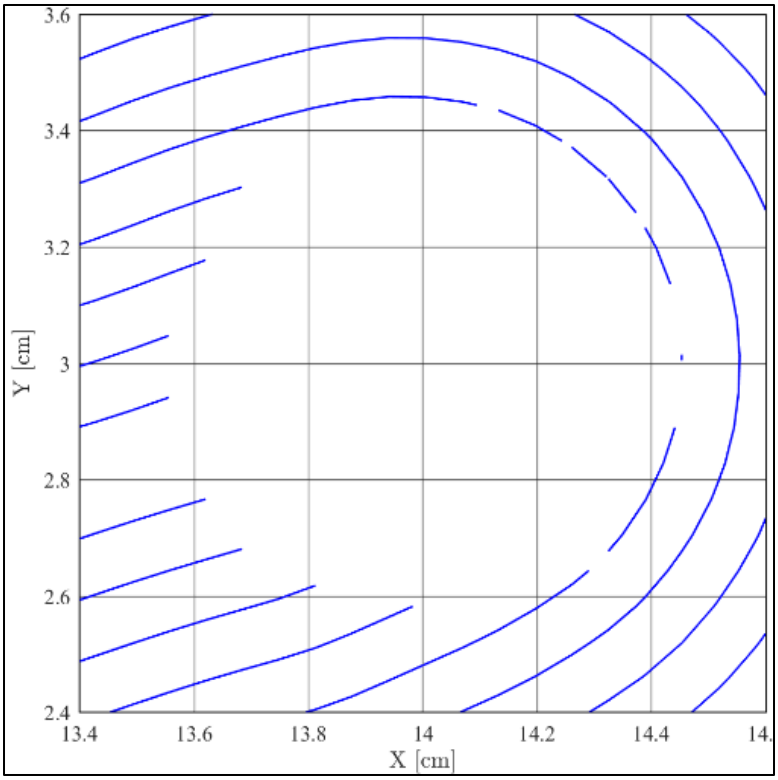


Figure 36: Zoom "H" into Figure 28

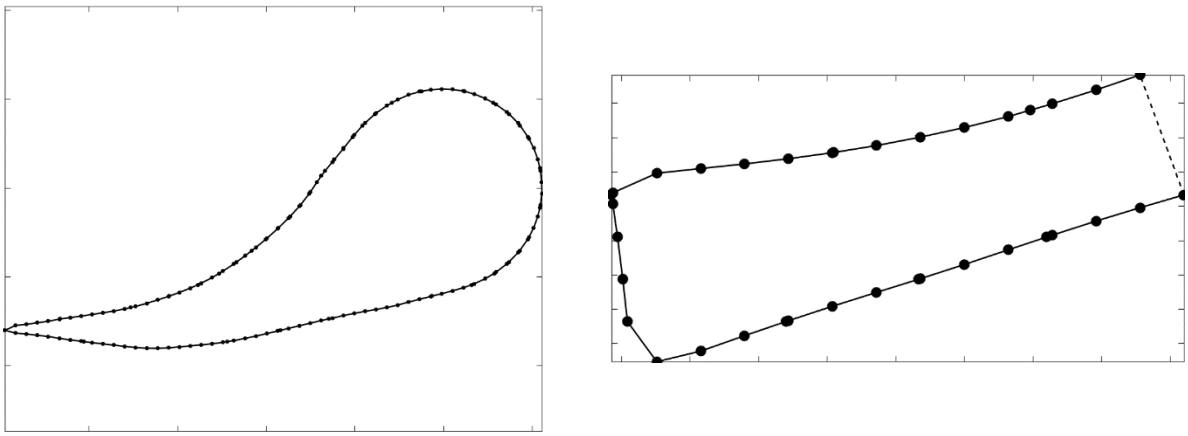
2.3 Polyline of representation of SDF levels

The level sets obtained by interpolation of the SDF at the query heights produces Figure 27. These contours consist of *xy*-coordinate data which alone are meaningless. The first step to create fiber paths from this data is to quantify aspects of the discrete curve implied by the *xy* data. Qualitatively, there are two questions that need to be answered: Firstly, is the fiber short? Secondly, is the level set oriented inward or outward? The second question is addressed in this section. The answer to the first question requires build-up that starts here and will conclude with the discussion of Concept 5. In the interim, the discussion of Concept 2 centers around two quantities: *perimeter* and *signed area*. The former is important for filtering the *short fibers* and the latter is important to carry out the *offset* operation. Furthermore, it is important to distinguish *open* contours from *closed*

contours (Figure 37). Here, the question of making a computer program determine these geometric aspects is discussed.

2.3.1 Closed vs. Open Polylines

Programmatically speaking, both types of polylines are identical in terms of their input: A number "*n*" indicating the number of points, and a sequence of "*n*" *xy* coordinate pairs. When the level sets are closed, a suitable mathematical representation is that of a *polygon*, which is a polyline with identical start and end point. When the polylines are open, one may still think of a polygon but must bear in mind that the first and last point of the sequence are not connected. To handle the distinction (open vs. closed curves) and to implement a repertoire of numerical operations on discrete curves, a data structure called "polygon.m" was developed in MATLAB®.



(a) (b)

Figure 37: Closed (a) vs. Open (b) polylines.

2.3.2 Polyline Perimeter

The quantity used to assess whether a center line constitutes a short fiber is the *perimeter*, a measure of the length of a curve. For both open and closed polylines the perimeter is given by the sum of the Euclidean distance of all their edges. If the polyline is closed, there is an implied edge

defined as the connection of the first and last *xy*-coordinate pairs of the input sequence. This is not the case for an open curve. Define the Euclidean distance between two points "a" and "b" in two dimensions as:

$$L(P_a, P_b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2} \tag{1}$$

Then the perimeter of a discrete open curve with *n* points number "1" through "*n*" is:

$$P_{\text{open}} = \sum_{i=1}^{n-1} L(P_i, P_{i+1}) \tag{2}$$

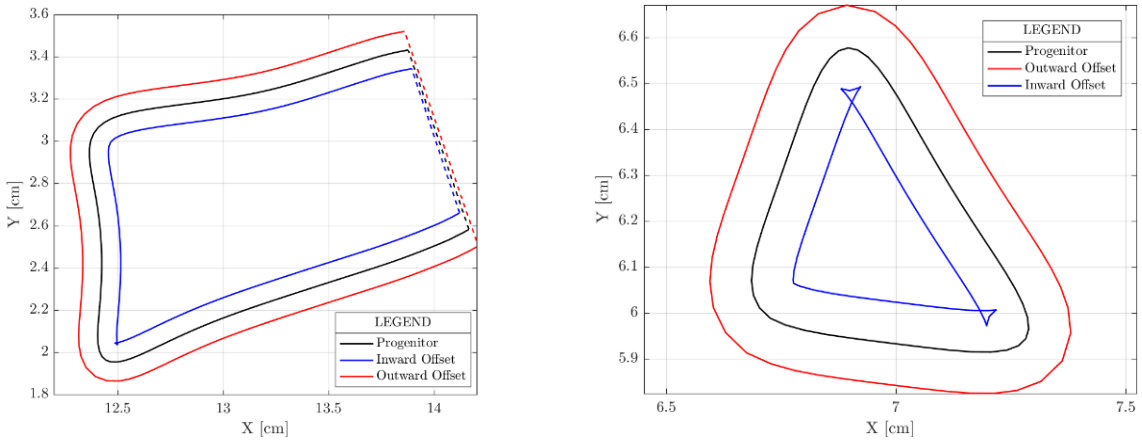
Then, the perimeter of a discrete closed curve also with *n* points number "1" through "*n*" is:

$$P_{\text{closed}} = L(P_1, P_n) + P_{\text{open}} \tag{3}$$

Where $L(P_1, P_n)$ is the last edge for a closed discrete curve, which by definition is absent for the open curve. That is, if the open curve in Figure 37 had been flagged as closed, the dashed line would seize to be phantom and would have instead been drawn as a solid line.

2.3.3 Polyline Signed Area

One of the key steps in creating the center line is the *offset* operation, which consists of defining a new curve (referred to here as "inheritor") from an already existing curve (referred to here as "progenitor") by displacing the original points in a direction normal to the curve. The newly displaced points form the defining sequence of the so-called *inheritor* and were all displaced by an equal amount. This operation can be performed *inwards* or *outwards*. For clarity, inspect Figure 38 below and note that the inheritor may exhibit self-intersections depending on the direction of the offset.



(a) (b)

Figure 38: Inward vs. outward discrete curve offset for (a) an open polyline and (b) a closed polyline.

The offset operation can be blindly carried out by computing the normal vectors to the curve from the *xy* sequence using cross products. Doing so, however, leaves the user at the mercy of the sequence's *orientation*. That is, the order in which the points are listed in the *xy* sequence (which can be interpreted as *clockwise* or *anticlockwise*) determines the direction of the offset [16]. See the red arrows shown in Figure 39 and Figure 40 for clarity. Now, one may compute the unit normal blindly and reverse the direction of the normal as needed (to toggle between inward and outward offset directions). To make the process automatic, a program needs to determine the orientation.

A quantity that can be used to readily answer this question is the *signed* area of a polygon. This is computed using the so-called *shoelace* formula:

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i) \tag{4}$$

This formula forms triangles by using points P_i, P_{i+1} and the origin as the corners and then computes the magnitude of the cross production of the sides op_i and op_{i+1}. If the signed area is positive, then the polygon is oriented inwards, conversely, if the signed area is negative, then the

polygon is oriented outwards. In Figure 39 and Figure 40 two sequences of *xy* coordinates defining the same closed shape are shown. The sequences contain identical entries but in one sequence the order is reversed. The slender blue triangles are representative of the *shoelace* formula. Red arrows show the ordering of the *xy*-coordinates.

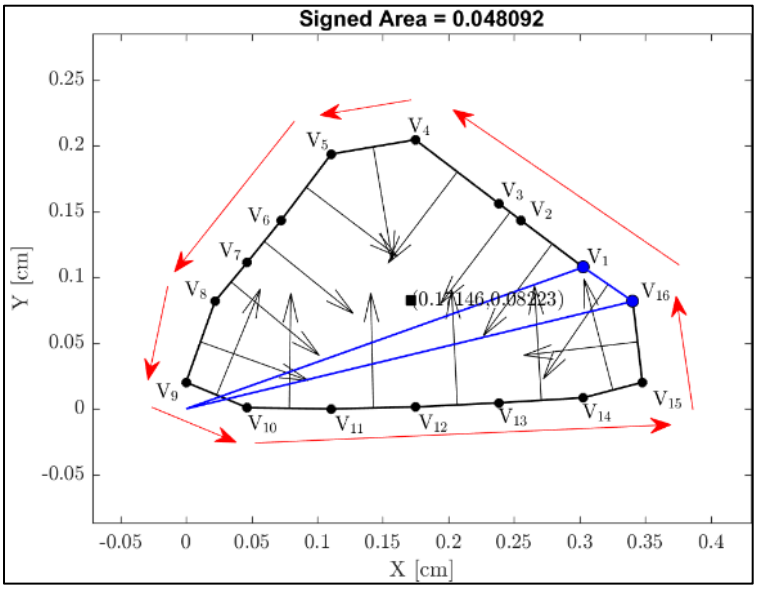


Figure 39: A positively oriented polygon

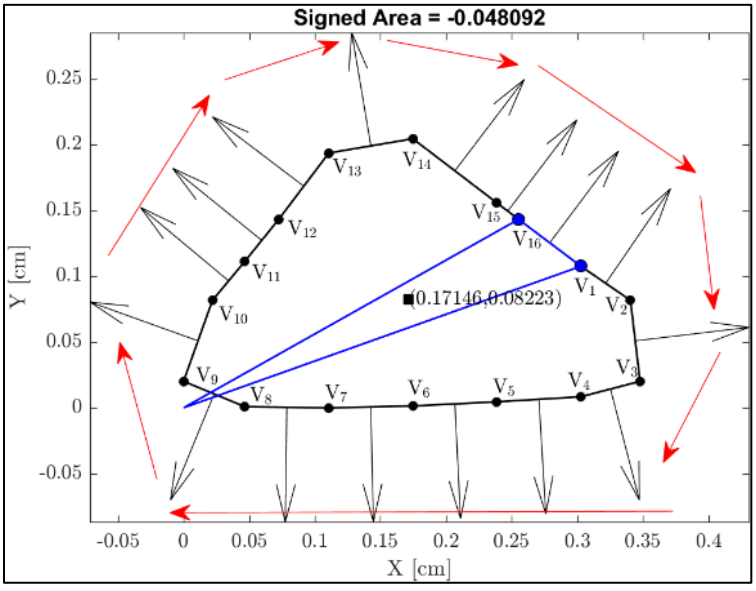


Figure 40: The polygon in Figure 39 with reversed orientation.

Without the signed area, the only other way to determine orientation is in a "post-mortem" way by first blindly offsetting the polygon to obtain the inheritor and then comparing its size against that of the progenitor (think of Figure 38). If the offset occurred in the wrong direction, one would have to re-offset the polygon in the opposite direction. The signed area computation requires 2 multiplications and 1 addition per point, whereas offsetting from scratch requires 6 multiplications, 12 additions, and 1 division per point (this will be shown in the discussion of Concept 3). For this reason, the signed area is found for all level sets to determine their orientation as it is the lowest-cost alternative [16].

2.4 Polyline Offset

With an appropriate way to measure and determine the orientation of the level sets discussed in Concept 1, one may formulate the offset procedure. Let $(n_x)_i$ and $(n_y)_i$ denote the x and y components of the unit normal at some side i of the polygon. Then, for some edge "i" said quantities are given by:

$$n = \begin{bmatrix} (n_x)_i \\ (n_y)_i \end{bmatrix} = \frac{1}{L(P_i, P_{i+1})} \begin{bmatrix} y_i - y_{i+1} \\ x_{i+1} - x_i \end{bmatrix} \tag{5}$$

Where $L(P_i, P_{i+1})$ is given by Equation (1). Note that because these are the components of a unit vector, the following is true about $(n_x)_i$ and $(n_y)_i$:

$$\|n\| = (n_x)_i^2 + \left(n_y\right)_i^2 = 1 \tag{6}$$

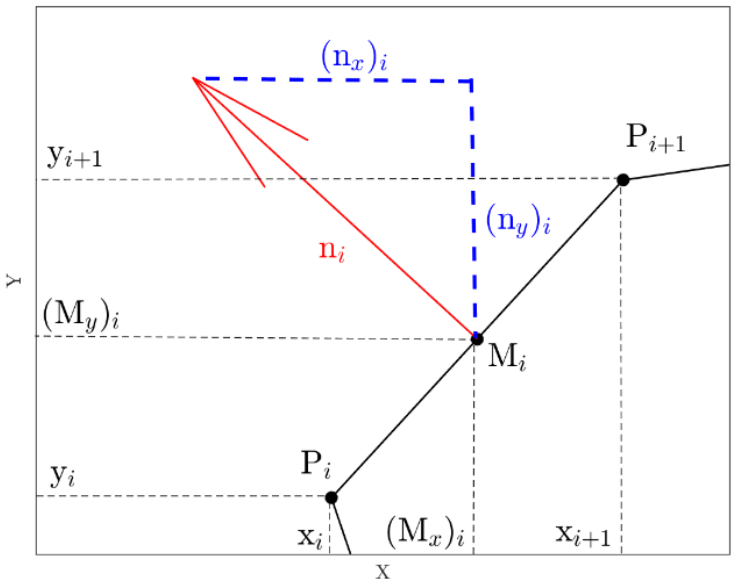


Figure 41: Nomenclature for an edge's normal and midpoint.

The pair of $(n_x)_i$ and $(n_y)_i$ encode a direction, which alone is meaningless. To be able to perform the offset operation lines need to be defined out of these unit vectors. For that, a pair of *xy* coordinates is needed per pair of $(n_x)_i$ and $(n_y)_i$. Denote said points as M_i (for "middle") with components $(M_x)_i$ and $(M_y)_i$ and define them as:

$$M_i = \begin{bmatrix} (M_x)_i \\ (M_y)_i \end{bmatrix} = \frac{1}{2}\begin{bmatrix} x_{i+1} + x_i \\ y_{i+1} + y_i \end{bmatrix} \tag{7}$$

The quantities discussed in Equation(5), Equation(6), and Equation (7)are visualized in Figure 41.

2.4.1 Extrusion in the direction of normals

Equation (7) is defined for edges except the phantom one ($1 \leq i \leq n - 1$), that is, if one is talking about a discrete open curve. If the curve is closed, one must consider the middle point of the last edge ($i = n$):

$$M_n = \begin{bmatrix} (M_x)_n \\ (M_y)_n \end{bmatrix} = \frac{1}{2}\begin{bmatrix} x_1 + x_n \\ y_1 + y_n \end{bmatrix} \tag{8}$$

Offset lines are now defined as:

$$O_i = M_i + dn_i \tag{9}$$

Where O_i is an intermediate offset point produced by displacing along the *n* direction by some distance *d* starting from point M_i. As it concerns this work, $d = t/2$ always. The reason the offset is of half the fiber thickness is because the resulting *inheritor* polygon will define the centerline of the fiber. These O_i will be used as the staging points to define intersection lines. Denote the intersection lines as L_i and define them as:

$$L_i = O_i + z_i(P_{i+1} - P_i) \tag{10}$$

Where z_i is a parameter. To compute the "i^{th}" point of the inheritor polygon, one must, for all points on the progenitor polygon, intersect L_i with L_{i+1}. The intersection requires the following two equalities:

$$O_i^x + z_i(x_{i+1} - x_i) = O_{i+1}^x + z_{i+1}(x_{i+2} - x_{i+1}) \tag{11}$$

$$O_i^y + z_i(y_{i+1} - y_i) = O_{i+1}^y + z_{i+1}(y_{i+2} - y_{i+1}) \tag{12}$$

This system can be solved by first eliminating the parameter z_i and then solving for the parameter z_{i+1}. The result is:

$$z_i = \frac{O_{i+1}^x - O_i^x + z_{i+1}(x_{i+2} - x_{i+1})}{x_{i+1} - x_i} \tag{13}$$

Substitution of z_i into either Equation (11) or Equation (12) produces z_{i+1}. Doing so with Equation (11) yields:

$$z_{i+1} = \frac{\left(O_{i+1}^{y} - O_{i}^{y}\right)(x_{i+1} - x_i) - (O_{i+1}^{x} - O_{i}^{x})(y_{i+1} - y_i)}{(x_{i+2} - x_{i+1})(y_{i+1} - y_i) - (y_{i+2} + y_{i+1})(x_{i+1} - x_i)} \tag{14}$$

With z_{i+1} now known; it can be plugged back into Equation (10) to thus yield point "i+1" on the inheritor polygon. The process is repeated for all points on the progenitor polygon. The graphical interpretation of the process entailed by Equation (9) through Equation (14) is illustrated in Figure 42.

The point labelled "$L_i = L_{i+1}$" defines a point on the inheritor curve and each of the arrows shown adheres to the convention in Figure 41. The point-wise offset operation illustrated in Figure 42 is valid for all points in a closed curve. For an open curve, the first offset point is taken to be O_1, and the last offset point is taken to be O_n. Once the offset curve is produced, the perimeter can be measured and determine if the fiber is *short*. If the inheritor curve does not have a perimeter that is too short, then, one proceeds to the NURBS representation discussed in Concept 7.

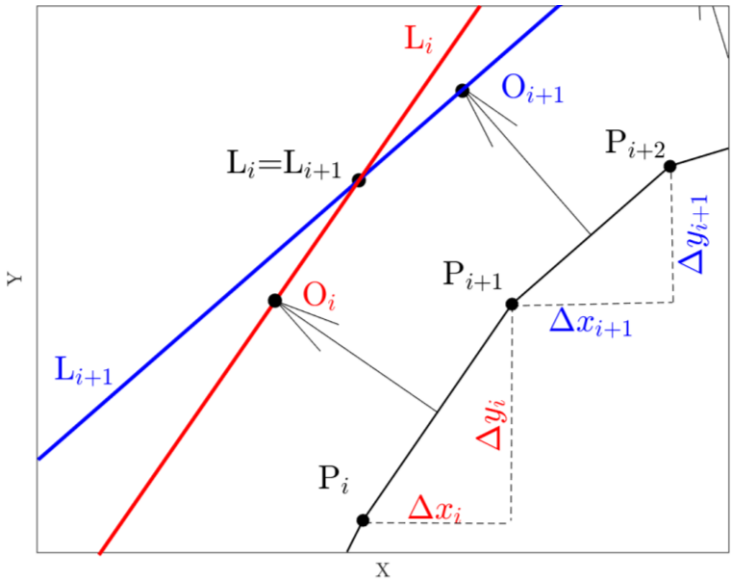


Figure 42: The Offset operation.

2.5 Suppression of Interpolation Sharpness and Noise

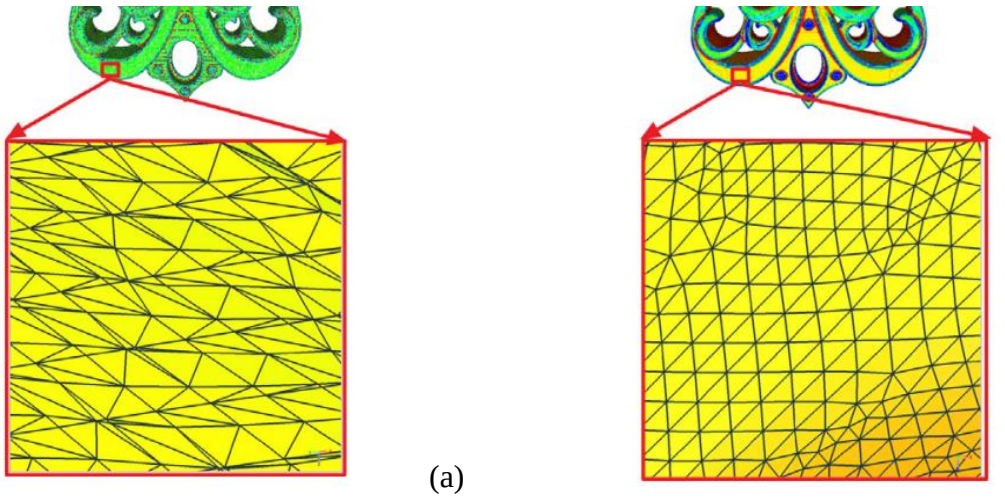
In an ideal world, one only needs Concepts 1 through 3 to generate the center lines of the fibers. However, as evidenced by the flaws showcased in Figure 28, additional measures must be taken. Here, a process to address the "noise" and "sharpness" problems (see Figure 34 and Figure 35) is discussed. The concept is that of *numerical diffusion*, a process typically described in literature by labels such as "smoothing," "averaging," or "regression." To characterize numerical diffusion, the ideas of "noise" and "sharpness" are briefly entertained.

2.5.1 Numerical Diffusion as a noise suppressor

"Noise" is a term used to refer to perceived dispersion, imperfections, or randomness in a discrete data set. Take for example, the practice of generating a mesh from data generated by 3D laser scanning of an object. Because the laser's light is affected by fluctuations in its own medium, the data it generates for the mesh is distorted [17]. Take Figure 43 as an example, there (a) shows the result of the raw laser scan whereas (b) shows the result of applying a "smoothing" scheme.

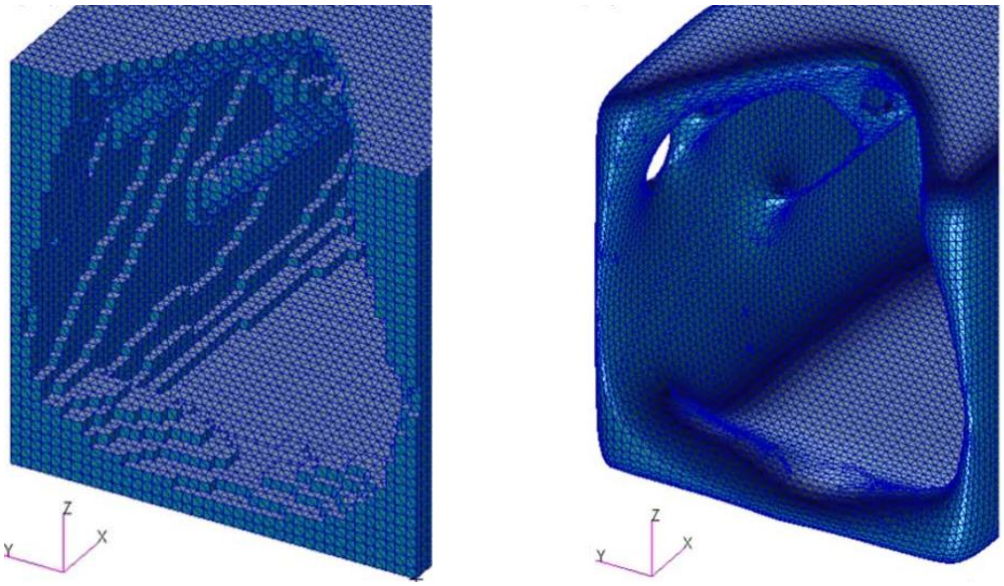
2.5.2 Numerical Diffusion as a smoother

"Smoothing" should be thought of as the antonym of "sharpening." Take for example a very coarse, or jagged but nonetheless exact feature like the ones shown in Figure 44 (a). Even though the features are exact, *locally*, they are akin to the approximation of a continuous, "average" feature. If one displaces the points that make up the exact sharp feature by what would appear to be a local tendency towards an average, one gets the Figure 44 (b).



(a) (b)

Figure 43: Example of noise in mesh generation (a) and noise suppression (b). Adapted from [17]



(a) (b)

Figure 44: Sharp features (a) and their "smoothed" counterparts (b). Adapted from [18]

2.5.3 Laplacian Smoothing

Numerical diffusion can be implemented in many forms. Here, the special case of Laplacian smoothing is discussed. Figure 43 and Figure 44 are examples of applying a discrete *Laplacian operator* of some kind based on some aspect of the geometry. There are many ways to apply the

Laplacian operator (see the work by [18] for four ways of doing so). Here, only the so-called *Explicit Classic Laplacian* scheme is considered:

$$\hat{P}_i = P_i + \lambda \nabla^2 P_i \tag{15}$$

Where λ is the scalar multiple (chosen by the user), ∇^2 is the Laplacian operator, P is some point on the discrete curve, and the subscript i denotes the point's position in a sequence. Equation (15) is in *analytic* form. In practice ∇^2 is always approximated numerically. To make matters even more open-ended there are multiple ways to compute ∇^2. In Figure 43 the so-called *graph Laplacian* is used to produce ∇^2, whereas in Figure 44 the so-called *mesh Laplacian* is used instead.

2.5.4 Finite Difference Laplacian Smoothing

The distinctions in the last subsection are beyond the scope of the work but are mentioned for the sake of completeness. For the SDF contours, the *finite difference Laplacian* is used to estimate ∇^2:

$$\nabla^2(P_i) = \nabla \cdot \nabla(P_i) \approx \sum_{j=\frac{n-1}{2}-s}^{\frac{n-1}{2}+s} c_j P_j\,, \qquad n = 3,5,7,\ldots \tag{16}$$

Where n is the number of *stencil* points (assumed to be an odd integer equal to three or greater), and s is an integer shift that equals zero for *central* differences, equals $-(n-1)/2$ for *forward* differences, equals $+(n-1)/2$ for *backward* differences, and can equal any integer values between $-(n-1)/2$ and $+(n-1)/2$ for *lopsided* differences. The reason that n is assumed to be at least three is because that is the minimum number of stencil points needed to approximate a second order derivative (like the Laplacian) using a *central* scheme. The *forward*, *central*, and *backward* schemes for the Laplacian using $n = 3$ are shown next:

$$\nabla^2(P_i) \approx P_i - 2P_{i+1} + P_{i+2} \tag{17}$$

$$\nabla^2(P_i) \approx P_{i-1} - 2P_i + P_{i+1} \tag{18}$$

$$\nabla^2(P_i) \approx P_{i-2} - 2P_{i-1} + P_i \tag{19}$$

Lopsided schemes for the finite difference Laplacian require $n = 5$ or higher. Regardless, most fibers consist of closed discrete curves, therefore central difference schemes are used the most. Some fibers, however, are open, hence the need for the parameter *s,* as it is unnatural to consider the first point of an open curve to be part of the neighborhood of the last point (and vice versa).

2.5.5 Finite Difference Schema for $n \geq 5$

Equation (17), Equation (18), and Equation (19) are first, second, and first order schema respectively involving a stencil of three points. Higher order schemes exist and were deemed necessary as part of this work because of three (4) issues associated with Laplacian Smoothing: "Stalling," Feature loss, "Shrinkage", and numerical instability. A fifth issue, that of open-curve warping was encountered and will be illustrated but not addressed. All for issues are discussed in the subsections after the next one. The justification for high-order scheme will be made in the next subsection in the form of examples, for now, a quick way to derive them is presented.

In literature, the finite difference coefficients are typically derived using Taylor expansions over an evenly spaced stencil [19] [20]. This approach is limiting when deriving high-order schema because it entails solving a linear system [21]. The solution to said system produces the desired coefficients for the desired scheme but it also produces the coefficients for other, unneeded schema. An alternative way to derive them is with Lagrange Interpolating Polynomials (LIP) which do not require inversion of a linear system.

Introducing the LIP, they are the weighted sum of the Lagrange constituents of a given sequence of *indeterminates* (*x*) and another sequence of *determinates* (*y*) [19]. Denote the Lagrange polynomial as $L(x)$ and its constituents as $\phi(x)$. Then:

$$L(x) = \sum_{j=1}^{N} y_j \phi_j(x) \tag{20}$$

$$\phi_j(x) = \frac{\prod_{i \neq j}(x - x_i)}{\prod_{i \neq j}(x_j - x_i)}, \tag{21}$$

That is, the Lagrange polynomial is a sum of its constituents weighed by the values of the determinates. Each basis constituent $\phi(x)$ is associated with an interpolation point. At any rate, one may derive a finite difference scheme for the k^{th} derivative using n stencil at some location $x = x_0$ points by evaluating:

$$\frac{d^k P_n(x)}{dx^k} \approx \sum_{j=0}^{n} P_j \left(\frac{d^k \phi}{dx^k}\bigg|_{x=x_0} \right) \tag{22}$$

That is, the c_j from Equation (16) is the sequence of the k^{th} derivatives of the Lagrange basis constituents [21].

$$c_j = \frac{d^k \phi}{dx^k}\bigg|_{x=x_0} \tag{23}$$

To use Equation (22) one must input the coordinate sequence P_j. The differences in the coordinates will imply even or uneven spacing and that will be reflected by symmetry or asymmetry respectively in the weights c_j. Here, use of Equation (22) is relegated to sequences of the form:

$$\left[-\frac{\mathrm{n}-1}{2}, -\frac{\mathrm{n}-1}{2}+1, \ldots, -1, 0, +1, \ldots, \frac{\mathrm{n}-1}{2}-1, \quad \frac{\mathrm{n}-1}{2}\right]$$

To produce *central* schemes, or of the form:

$$\left[-\frac{\mathrm{n}-1}{2}+\mathrm{s}, \quad -\frac{\mathrm{n}-1}{2}+1+\mathrm{s}, \ldots, \mathrm{s}-1, \mathrm{s}, \mathrm{s}+1, \ldots, \mathrm{s}+\frac{\mathrm{n}-1}{2}-1, \quad \mathrm{s}+\frac{\mathrm{n}-1}{2}\right]$$

To produce *forward*, *backward*, and *lopsided* schemes, where *s* is the same parameter seen in Equation (16). To recover the finite difference schemes with even spacing in which $\Delta x = 1$, set $x_0 = 0$. This instructs the LIP that the interpolation is centered around the origin. Thus, in this work, during the Laplacian smoothing operation, a local origin is defined for all points on the discrete SDF level sets.

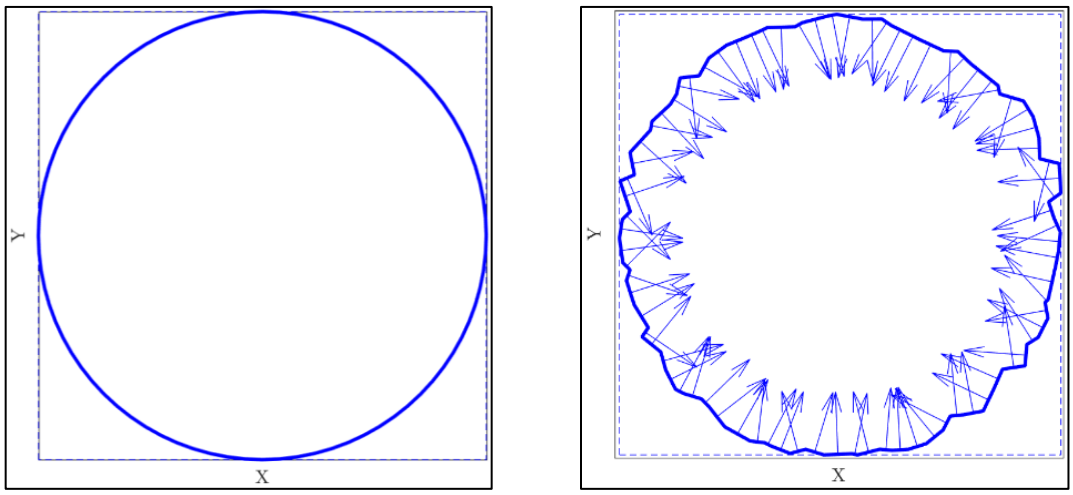
2.5.6 Numerical characterization of Laplacian Smoothing on polylines

With the help of Lagrange interpolation polynomials, any high-order finite difference scheme can now be quickly derived. In this subsection, a simple test case is presented to showcase the relative performance of schemes with $n = 3$, $n = 5$, and $n = 7$. The test consists of generating a regular 80-sided polygon whose edges all have length of 2 as shown in Figure 45 (a). The polygon is then subjected to Gaussian noise by adding random numbers between 0 and 0.75 to its *xy* coordinates to produce something like in Figure 45 (b).

The following metrics are tracked over multiple "passes" of Equation (15), the *Explicit Classic Laplacian* using finite differences: The polygon's area, perimeter, *x*-coordinate of centroid, and *y*-coordinate of centroid. The results are summarized by Table 2 through Table 5. All percent changes are relative to the polygon's original values: Area of 509, Perimeter of 160, and *x* and *y* coordinates of centroid of 0. The centroid of a polygon is computed using variants of the *shoelace* formula (Equation (24) and Equation (25)).

$$x_c = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1}y_i)(x_i + x_{i+1}) \tag{24}$$

$$y_c = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1}y_i)(y_i + y_{i+1}) \tag{25}$$



(a) **(b)**

Figure 45: Test polygon (a), Test polygon subject to Gaussian Noise (b).

These results hint at the shrinkage phenomenon as evidenced consistent percent decrease in the polygon's area. The same trend is observed in the polygon's perimeter. The change in area appears to be proportional to the value of λ and not necessarily affected by the choice of scheme. The changes in the perimeter appear to be influenced by both the scheme and the value of λ. Finally, the *xy*-coordinates of the centroid shift slightly in response to the Explicit Laplacian. Therefore, this operation is characteristic of scaling, and to some extent translation.

Table 2: Percent changes in polygon's area

λ	N	Pass 5	Pass 10	Pass 15	Pass 20
0.5	3	-2.454	-5.417	-8.291	-11.079
0.3	5	-1.494	-3.300	-5.070	-6.807
0.3	7	-1.492	-3.298	-5.066	-6.802
0.1	7	-0.509	-1.132	-1.744	-2.350

Table 3: Percent changes in polygon's perimeter

λ	N	Pass 5	Pass 10	Pass 15	Pass 20
0.5	3	-3.010	-4.907	-6.535	-8.064
0.3	5	-2.257	-3.222	-4.126	-5.013
0.3	7	-2.961	-4.106	-5.020	-5.901
0.1	7	-2.882	-3.447	-3.820	-4.149

Table 4: Percent changes in polygon's x-coordinate of centroid

λ	N	Pass 5	Pass 10	Pass 15	Pass 20
0.5	3	-0.3958	-0.8615	-1.2650	-1.6298
0.3	5	-0.4984	-0.7453	-0.9239	-1.0831
0.3	7	-0.4864	-0.7454	-0.9025	-1.0460
0.1	7	-0.8225	-1.0310	-1.1466	-1.2360

Table 5: Percent Changes in polygon's y-coordinate of centroid

λ	N	Pass 5	Pass 10	Pass 15	Pass 20
0.5	3	+0.0912	-0.1446	-0.3898	-0.6292
0.3	5	+0.2176	+0.0330	-0.1704	-0.3720
0.3	7	+0.2883	+0.1047	-0.1079	-0.3204
0.1	7	-0.2554	-0.1973	-0.2134	-0.2642

2.5.7 Shrinkage

This is a phenomenon built into the Laplacian smoothing itself and is well documented in all applications that use it. Simply put, the overall shape of a curve will globally shrink. Figure 46 showcases this phenomenon on one of the shorter SDF contours. The finite difference Laplacian smoothing scheme used there was central with $\lambda = 0.3$ and 7 stencil points. Shrinkage is exacerbated mainly by the choice of λ. The more the operation is repeated (compounded by arbitrarily high choices for λ) the more pronounced the effect is [18]. In Figure 46, the red curve is the original SDF level set. The contour numbers denote how many passes of the Explicit Laplacian were carried out to get to said contours. This can potentially deviate the fibers substantially from the topologically optimized design.

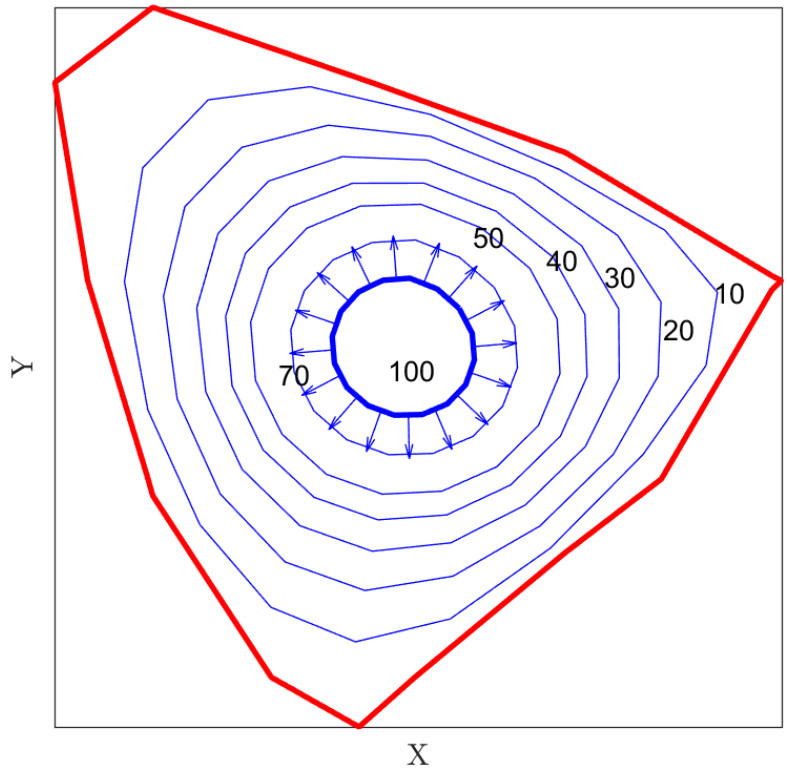


Figure 46: Example of Shrinkage due to Laplacian smoothing.

2.5.8 Feature Loss

In this work, Laplacian smoothing was originally hypothesized as a suitable technique to address the *tight* turn problem. However, during testing, fibers that were clearly sharp (like those in Figure 32 and Figure 34) showed substantial improvement in their turns, however, the number of passes required was in the dozens per fiber. This made the process slow, but seemingly worthwhile at first.

Unfortunately, the fibers can become substantially morphed to the point where they deviate unacceptably from the original topologically optimized design. Furthermore, for most of the fibers, the tight turns represented less than 10% of the overall fiber, which rendered Laplacian smoothing highly inefficient. This is because it was deployed as a *global* as opposed to a *local* operation. To exemplify the feature loss phenomenon, Figure 47 and its exploded views (Figure 48 and Figure 49) were created.

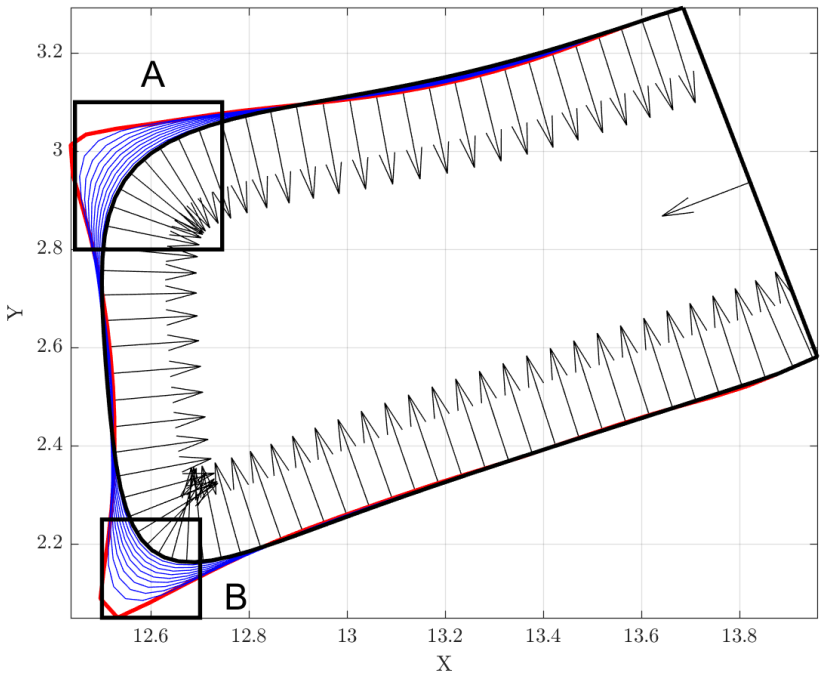


Figure 47: Example of Feature loss due to Laplacian Smoothing

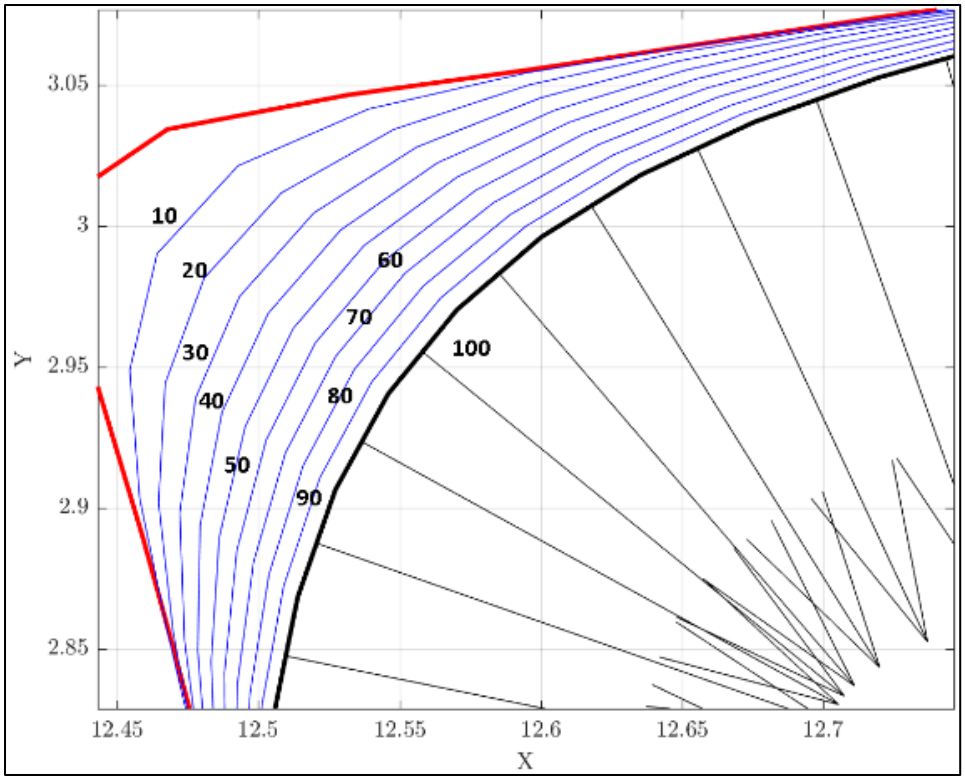


Figure 48: Zoom "A" into Figure 47

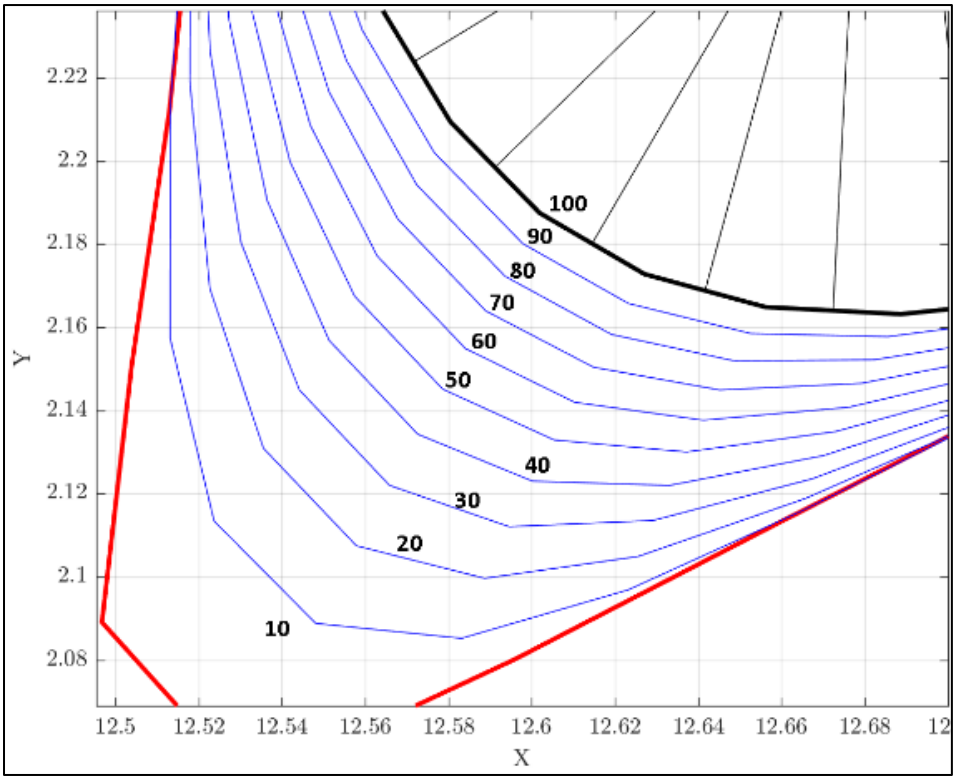


Figure 49: Zoom "B" into Figure 47

2.5.9 Numerical Instability

Broadly speaking, the term “numerical stability” refers to whether the rounding or truncation errors in a numeric scheme remain bounded through its iterations [20]. Here, the term is used to characterize a potential danger of Laplacian smoothing and that is the potential for unbounded distortion of the original shape. This is exemplified by a 3-point stencil using $\lambda = 0.55$ in Figure 50.

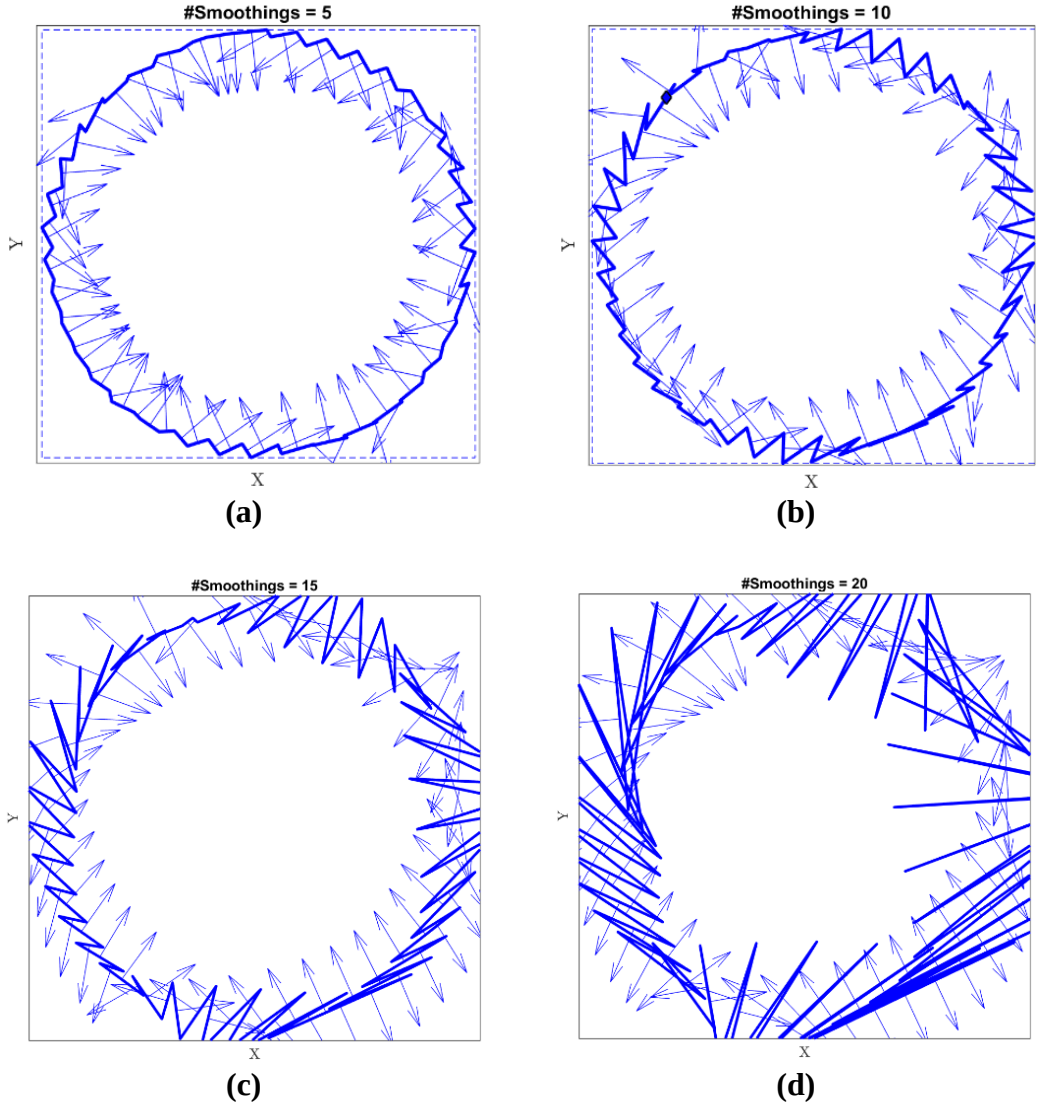
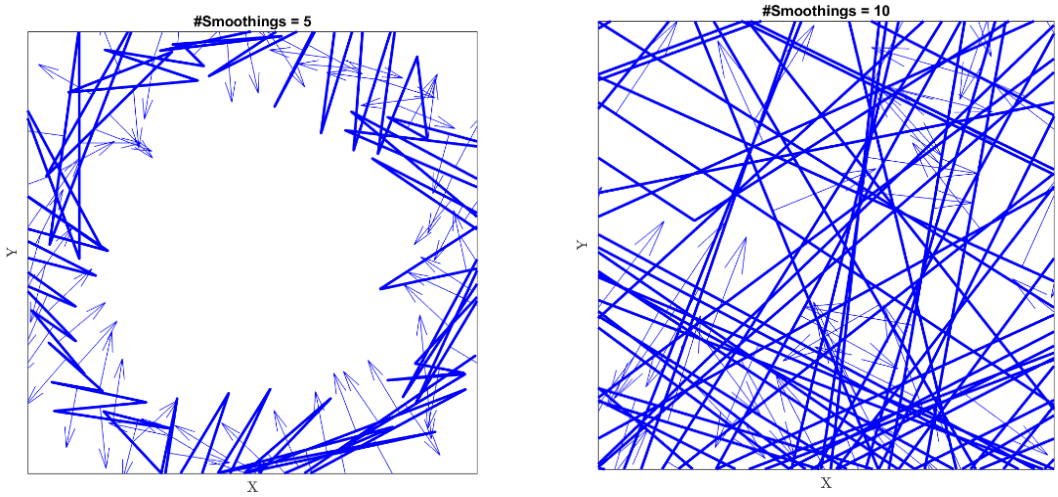


Figure 50: A numerically unstable example Explicit Laplacian scheme with $\lambda = 0.55$, $n = 3$.

In this example, the polygon gets progressively distorted to the point where it is essentially obliterated after 20 passes. The instability showcased in Figure 50 is driven by the value of λ and exacerbates with number of stencil points. In Figure 51, an additional two stencil points were used, and the instability grew so quickly that the extent of the polygon's destruction is now akin to a seemingly random point generation.



(a) (b)

Figure 51: A numerically unstable example with λ = 0.55, n = 5.

The phenomenon exhibited in Figure 50 and Figure 51 is an example of *Laplacian growth* and has been documented in fields of physical simulation involving diffusion equations [22]. Laplacian growth must be avoided at all costs to guarantee that the preparation of the progenitor polyline leading up to the offset operation is flawless. This phenomenon guides an intuition of selecting arbitrarily low values of λ and applying a minimal number of passes. Stability analysis was beyond the scope of this work, so no guidelines for gauging a stability limit for λ are presented here. The smoothing of the progenitor polylines was thus carried out through trial and error.

2.5.10 Stalling: The Onset of Laplacian Growth

When the choice for λ was sufficiently close to the stability threshold of the scheme, the convergence rate on the "smoothing" aspect of the Laplacian smoothing is susceptible to slow downs. This can be described as subsequent iterations of the Explicit Laplacian undoing the progress of the previous iteration. One would think of it as damped oscillations between values, however, in practice it can easily be a delayed numerical explosion. There were extreme cases where the slowdown is so lethargic, that the shrinkage phenomenon begins to dominate before any meaningful smoothing takes place.

Consider the following retrial of the distorted polyline in Figure 45 with a central scheme of with 3 stencil points at first with $\lambda = 0.35$, and then with $\lambda = 0.501$. The results after 100 passes of the Explicit Laplacian are Figure 52 and Figure 53 respectively. The former quickly recovers the circularity of the original polyline and should have only received 10 passes. The latter exhibits noise well into the 100^{th} pass, and is thus "stalled," meaning that one was better off not even deploying the scheme in the first place.

The phenomenon is characterized by the formation of ripples, which really mark the onset of Laplacian Growth. When the ripples form, it is only a matter of time before Laplacian growth takes place and becomes appreciable to the naked eye. Quantification and characterization of this rippling phenomenon and that of Laplacian growth is beyond the scope of this work.

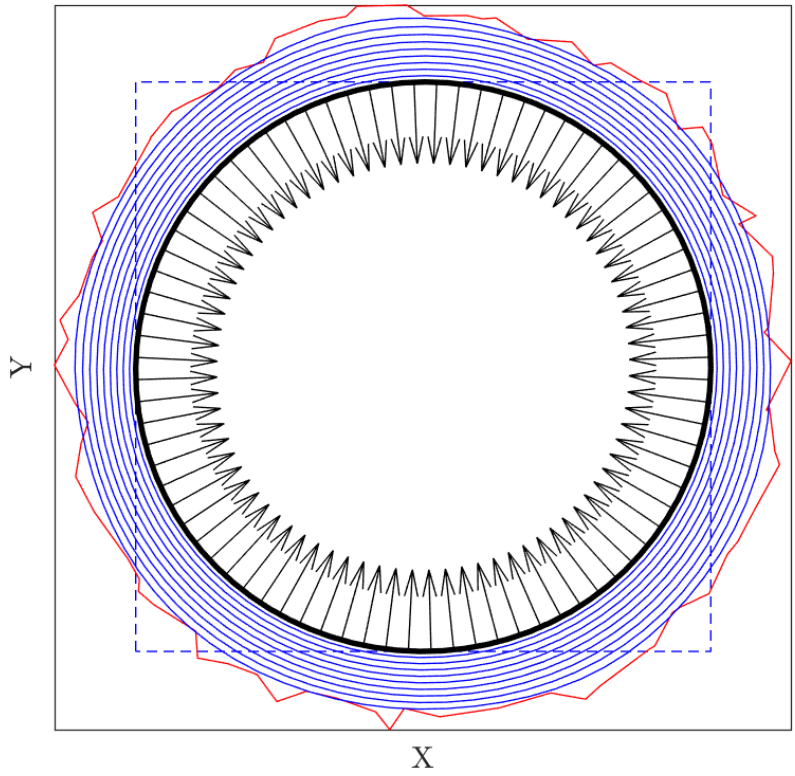


Figure 52: A hundred passes of Laplacian smoothing on Figure 45 (a) (λ = 0.35)

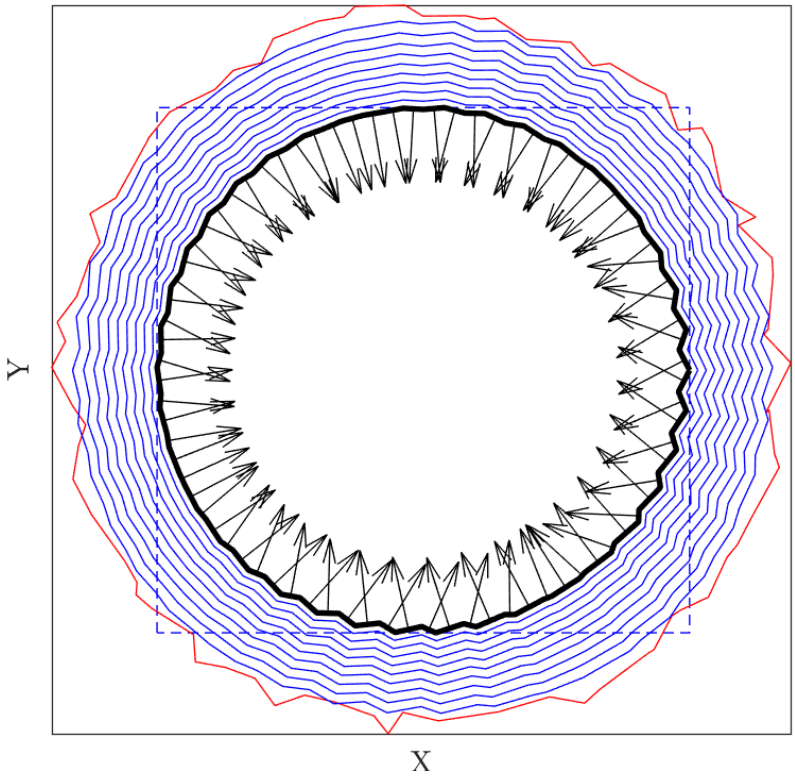


Figure 53: A hundred passes of Laplacian smoothing on Figure 45 (a) (λ = 0.501)

2.5.11 Open polyline warping

During testing, of Laplacian smoothing of open polylines with forward, backward, and lopsided schemes at the endpoints a phenomenon whereby the translation problem and distortion problems are exacerbated was encountered. Numerically, this is due to ∇^2 being disproportionately larger on one end of the curve than at the other. The effect can be characterized as the open polyline being warped, or strongly pulled from one end while seemingly fixed to the other end. The stencil had five (5) points and the schemes were derived using the LIP. The finite difference schemes used to perform the Explicit Laplacian are summarized in Table 6 through Table 10 .

Three (3) examples of the warping of polylines are shown in Figure 54 through Figure 56. In all of said figures, the "(a)" label shows the pre-Laplacian shape, and the "(b)" labels show the successive degeneration of the curve as passes of the Laplacian scheme were performed. In total, all three shapes received 100 passes of the Explicit Laplacian with $\lambda = 0.1$. This phenomenon was not fully understood during testing was avoided by forfeiting all use of the forward, backward, and lopsided schemes.

Table 6: Forward difference scheme for ∇^2 with 5-point stencil

c_j	+35/12	-26/3	+19/2	-14/3	+11/12
ID	i	i + 1	i+2	i +3	i +4
stencil	■				

Table 7:Lopsised forward difference scheme for ∇^2 with 5-point stencil

c_j	+11/12	-5/3	+1/2	+1/3	-1/12
ID	i- 1	i	i+1	i +2	i +3
stencil		■			

Table 8: Central difference scheme for ∇^2 with 5-point stencil

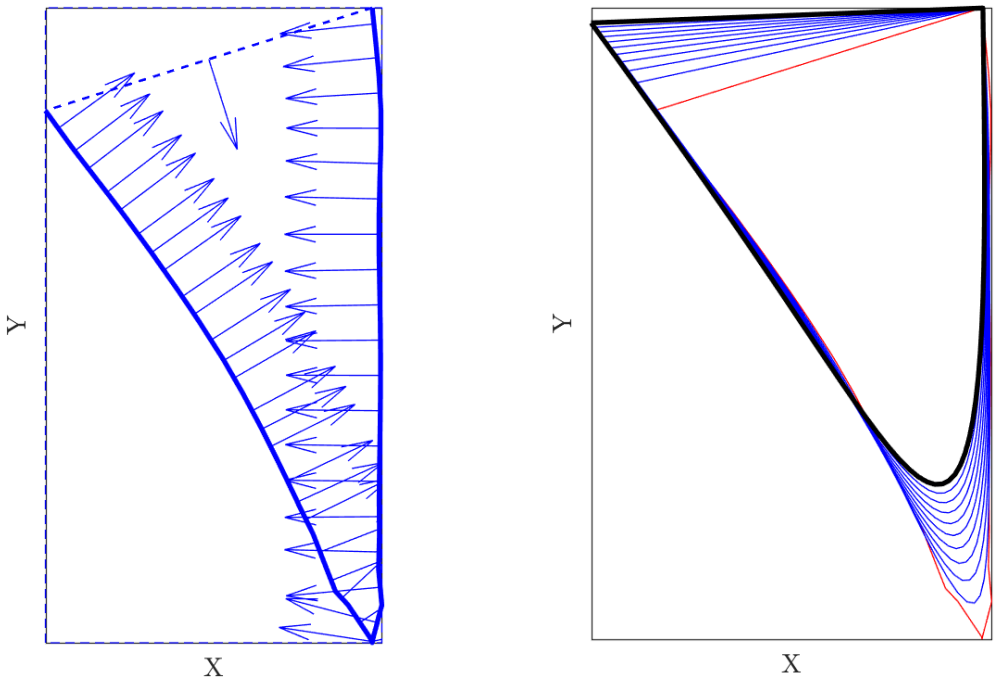
c_j	-1/12	+5/3	-5/2	+5/3	-1/12
ID	i- 2	i - 1	i	i +1	i +2
stencil					

Table 9: Lopsided backward difference scheme for ∇^2 with 5-point stencil

c_j	-1/12	+1/3	+1/2	-5/3	+11/12
ID	i- 3	i - 2	i-1	i	i +1
stencil					

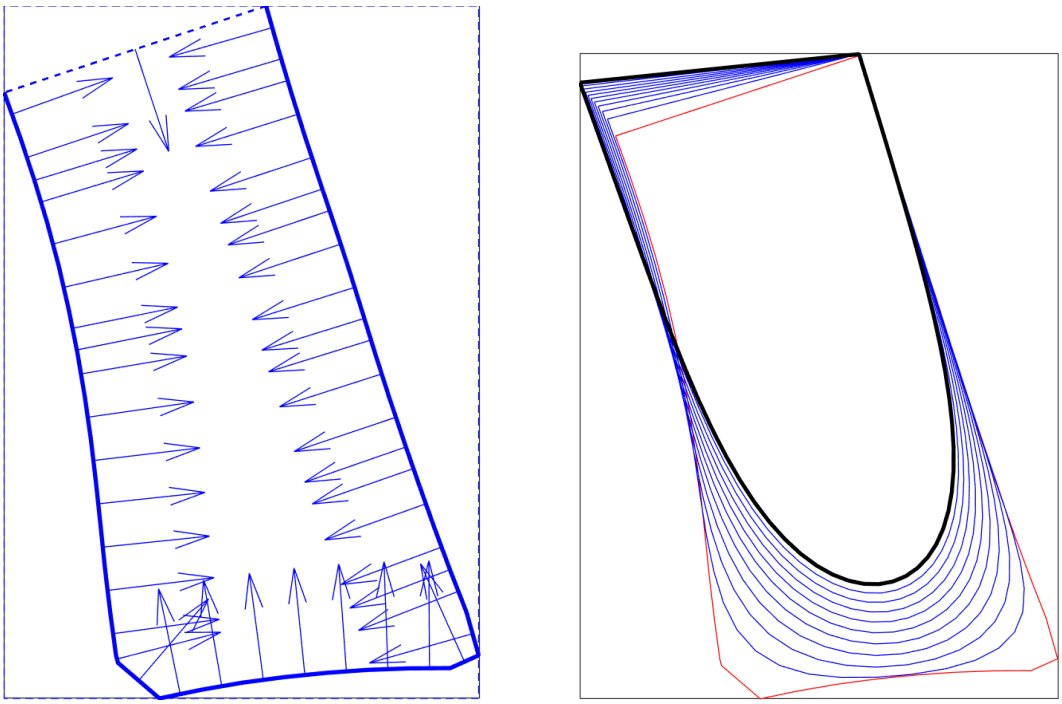
Table 10: Backward difference scheme for ∇^2 with 5-point stencil

c_j	+11/12	-14/3	+19/2	-26/3	+35/12
ID	i- 4	i - 3	i-2	i -1	i
stencil					



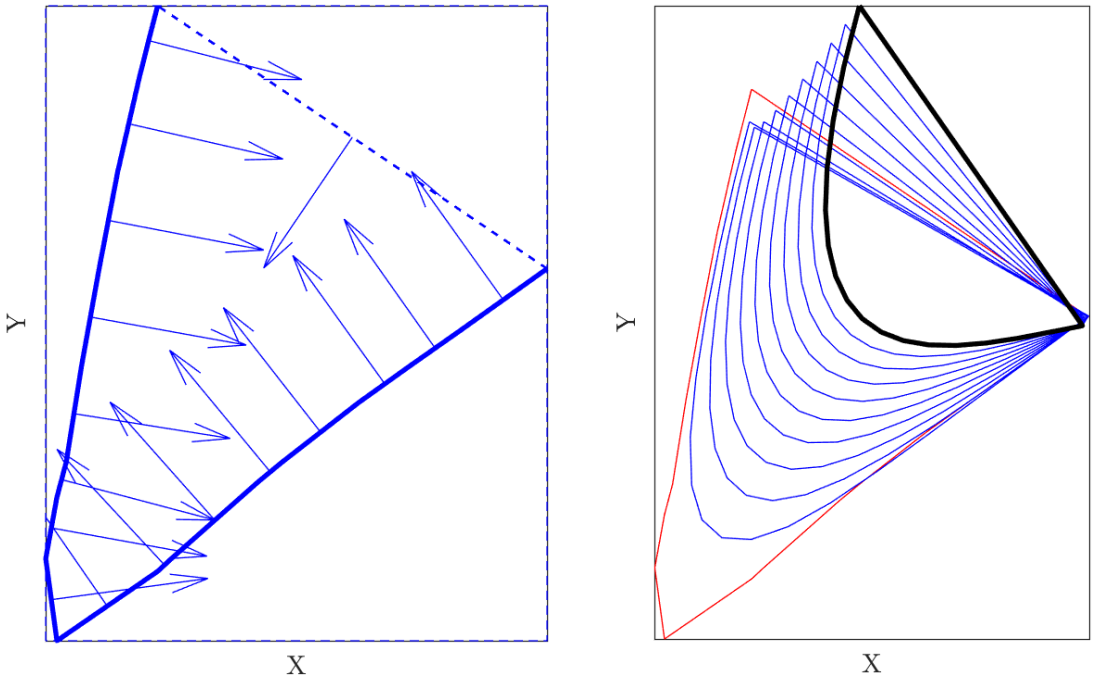
(a) (b)

Figure 54: Open polyline warping example 1



(a) (b)

Figure 55: Open polyline warping example 3



(a) (b)

Figure 56: Open polyline warping example 3

2.6 Defect detection

Application of the noise suppression principles covered so far, brings Algorithm 1 closer to fulfilling its objective. Despite implementation of noise suppression, the offset operation is not yet safe to perform. Look back at Figure 38 and notice the self-intersecting *clips* on both inward curves (there are two obvious ones in (b), but there is a small one at the bottom-left corner of the open curve shown in (a)). Notice that the progenitor curves in Figure 38 are in fact smooth. The self-intersections are due to the progenitor turning too tightly. If a 3D printer nozzle were to travel such a centerline, it would create an overflow of molten plastic. Tight turning must be avoided and will be addressed now.

2.6.1 Rollercoaster Algorithm

The SDF contours produced back in Concept 1 are meant to be the insides of the fibers. The strategy then is to offset them outward to almost eliminate the risk of producing self-intersecting "clips" (see Figure 38). There are two considerations though. The first one is that the centerline produced by the outward offset is not the actual path, it will be used to construct something called "NURBS" discussed in the next concept. The second consideration is the fact that even if a curve is technically smooth, its turn radius might be too small for the printer to traverse.

In this work, a technique for detecting sharp turns **before** offsetting the progenitor polygon and thickening the centerline NURBS has been developed. This is beneficial because creation of flawed NURBS centerlines and subsequent (yet to be developed) correction procedures are avoided entirely. The intuition for this algorithm was derived from the work by [23] in which they rolled a circle along a progenitor 2D NURBS to diagnose (in a "post-mortem" fashion) self-intersections in the offsets (see Figure 57).

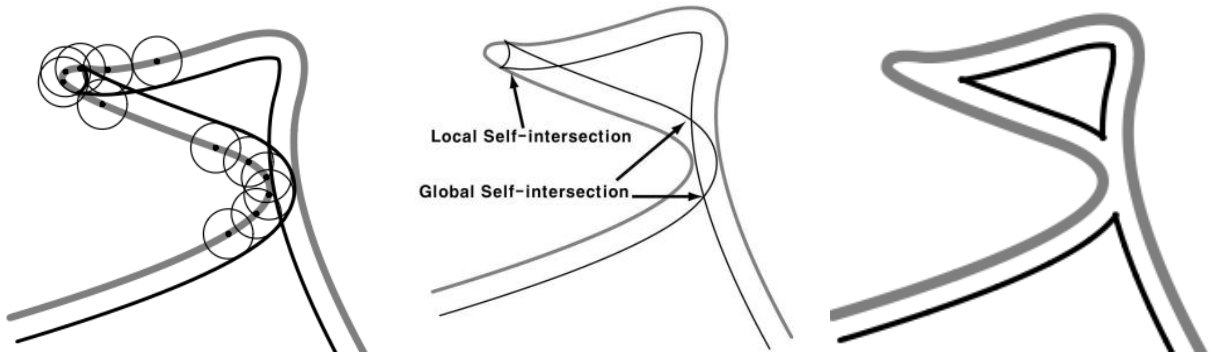


Figure 57: Circular sweep along a progenitor NURBS for generation of offset curved (a). Post-mortem diagnosis of self-intersections (b) is followed by a trimming procedure (c). Adapted from [23].

Seong, Elber, and Kim then proceed to develop their trimming algorithm in which the self-intersections are filtered to produce non-intersecting domains. Their algorithm is unsuitable for the manufacturability of fiber paths because the pair of trimmed offset curves no longer feature constant thickness. Instead of rolling the circle along a progenitor NURBS, the algorithm developed here will roll the circle on the inside of a progenitor polygon. To do so, an auxiliary inheritor polygon, termed here as "Rails", is offset inwards by some amount and then the circle travels it. This is illustrated in Figure 58, where the black lines are representative of the SDF contours (which are already supposed to be the inside of the fiber.

The formulation for the circle's center is identical to the procedure discussed in Concept 3, except that the offset distance is some value *R* instead of half the fiber thickness. The circle's center essentially travels an intermediate offset polygon. The ensuing sequence of displacements is akin to the circle riding the inside of the polygon, hence the analogy to a rollercoaster. As its re-centers, the circle is used to check whether it includes points on the progenitor polygon. The inclusion test of a circle vs. a point is whether:

$$(x_c - x_i)^2 + (y_c - y_i)^2 < R^2 \tag{26}$$

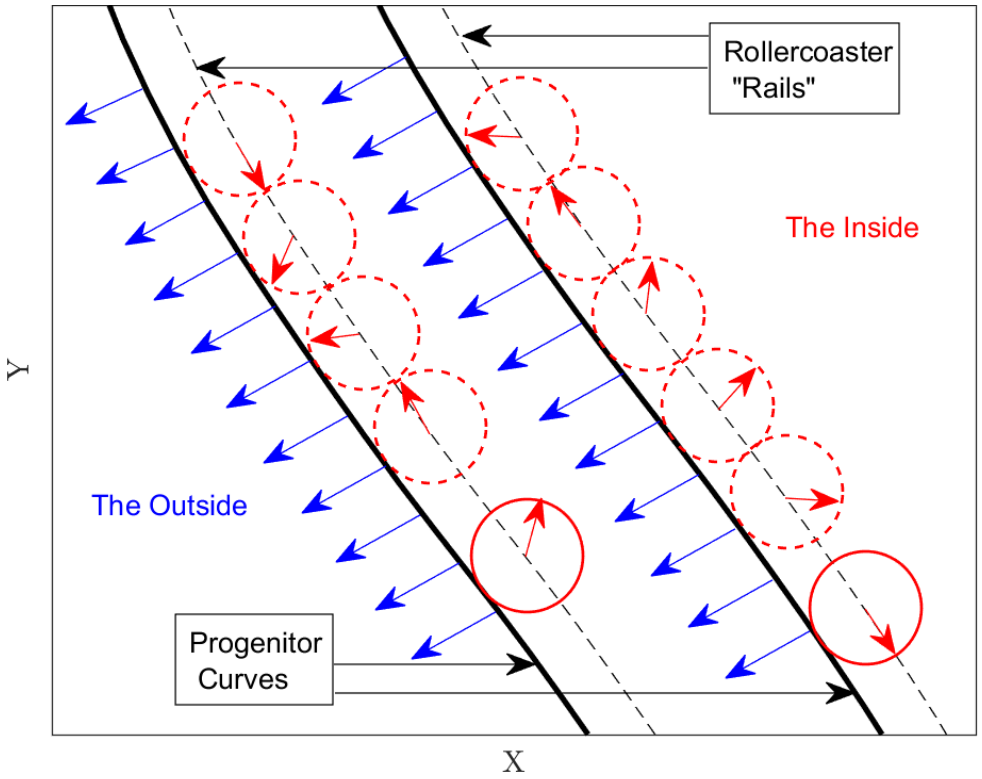


Figure 58: "Rollercoaster" algorithm

To prevent the inclusion detection operation from becoming an $O(n^2)$ endeavor, the polygon's coordinates are first sorted in ascending order along their *x* coordinates. This allows the rolling circle to perform local, instead of global inclusion checks which reduces the complexity to O(n log(n)). Prior to the rolling, a flag buffer is allocated to denote which points were included by the circle. The buffer is initialized to a value of "+1" for all points. During the rolling operation, the registers of the flag buffer corresponding to points found to be in the circle are set to "-1" After the rollercoaster ride is over, the flag buffer is inspected for continuous bands of "-1" to thus identify tight turns. Here three examples of rollercoaster's flag buffer are explained. Note that because the numbering of the vertices coincides with the numbering of the edges (as in vertex "1"

is at the beginning of edge “1”), the points that form the tight turns are, the one immediately before the first “-1” in a band, and the one at the end of the band (i.e., the very last “-1” in the band).

Table 11: Rollercoaster flag buffer example 1

ID	1	2	**3**	**4**	**5**	**6**	7	8	9	10
Flag	+1	+1	**-1**	**-1**	**-1**	**-1**	+1	+1	+1	+1

This first buffer has a single distinct band of “-1’s”, thus, the points corresponding to IDs 2 and 6 flagged as the band for the tight turn and then input into a blending routine discussed in the next subsection.

Table 12: Rollercoaster flag buffer example 2

ID	**1**	**2**	**3**	4	5	6	**7**	**8**	**9**	10
Flag	**-1**	**-1**	**-1**	+1	+1	+1	**-1**	**-1**	**-1**	+1

In this next example, two bands of “-1’s” are present. Whether they form tight turns will depend on whether the curve is open or closed. If the curve is closed, then the blending routine must be given points 10 and 3. If the curve is open, then the first band may be neglected. The second band entails a tight turn formed by points 6 and 9.

Table 13: Rollercoaster flag buffer example 3

ID	**1**	**2**	3	4	**5**	**6**	**7**	8	**9**	**10**
Flag	**-1**	**-1**	+1	+1	**-1**	**-1**	**-1**	+1	**-1**	**-1**

In this third example, there is a middle band of “-1’s” which is treated just like before, so the points with IDs 4 and 7 are sent to the blender. Pay close attention to the disjoint -1’s at the beginning and the end. If the curve is closed, that means that the beginning and end points form a

tight turn, thus, points 9, 10, 1, and 2, form a tight turn. The blender must thus be fed the coordinates of points 8 and 2.

2.6.2 Limitations of the Rollercoaster

The algorithm is relatively cheap and effective, but not quite perfect. During testing, the Rollercoaster algorithm exhibited two weaknesses. Firstly, it can potentially fail when deployed on highly *concave* polygons that wind into themselves. Secondly, it can fail to fully flag a tight turn (as in, several points on a tight turn are flagged but it misses at least one point to not fully flag a band). As an example of these failure conditions, inspect Figure 59 and its exploded views (Figure 60 through Figure 62)

The reason that the rollercoaster algorithm does not bode well with highly concave polygons is because it is inherently an inclusion-based flagger. All it does is check for points that fall inside the circle. It does not leverage the curvature of the SDF contour to recognize that regions of a concave discrete curve like the one showcased in Figure 62 are in fact not part of a tight curve. It is possible to detect these “false positives” by taking the dot product of the direction implied by the first segment in the band with the direction of the segment at the end of the band. If the dot product is positive, then the band is a false positive and no blending should take place.

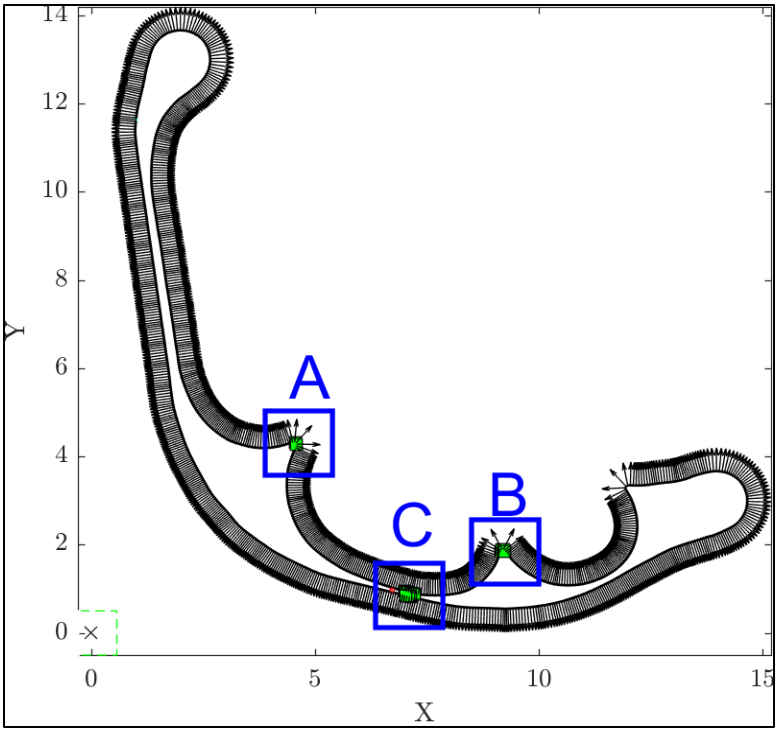


Figure 59: An example of a highly concave SDF contour

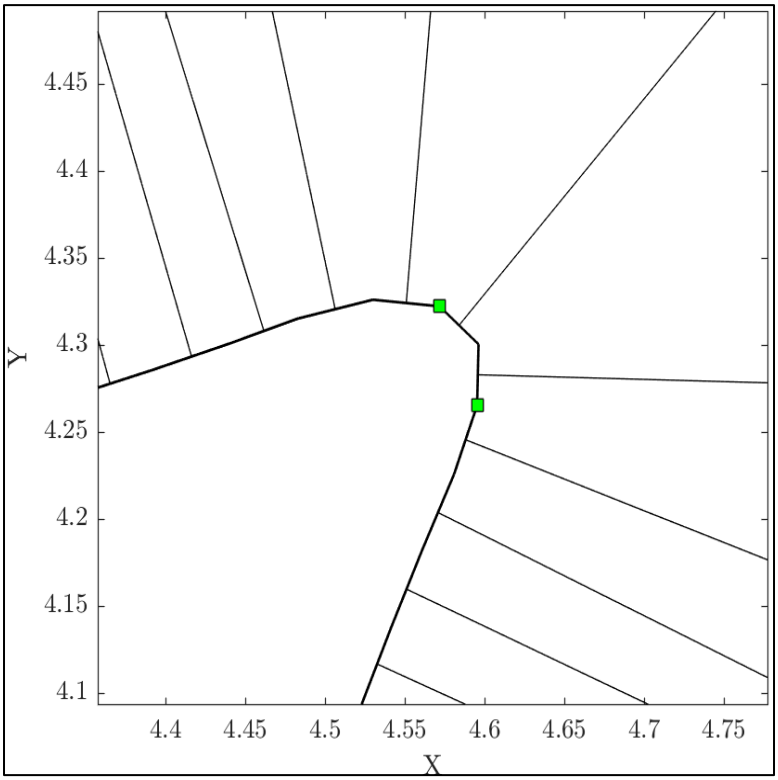


Figure 60: Zoom "A" into Figure 50

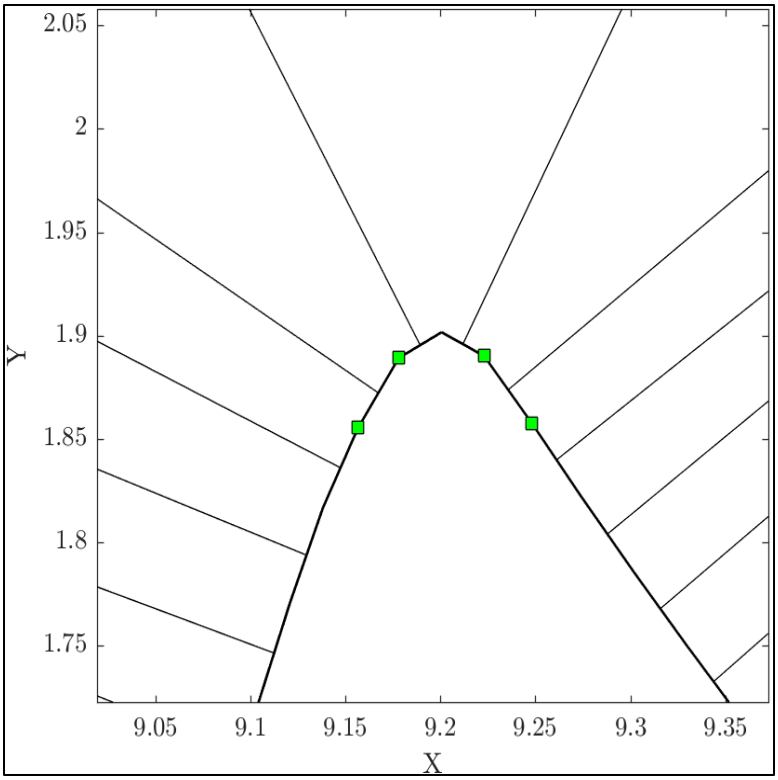


Figure 61: Zoom "B" into Figure 59

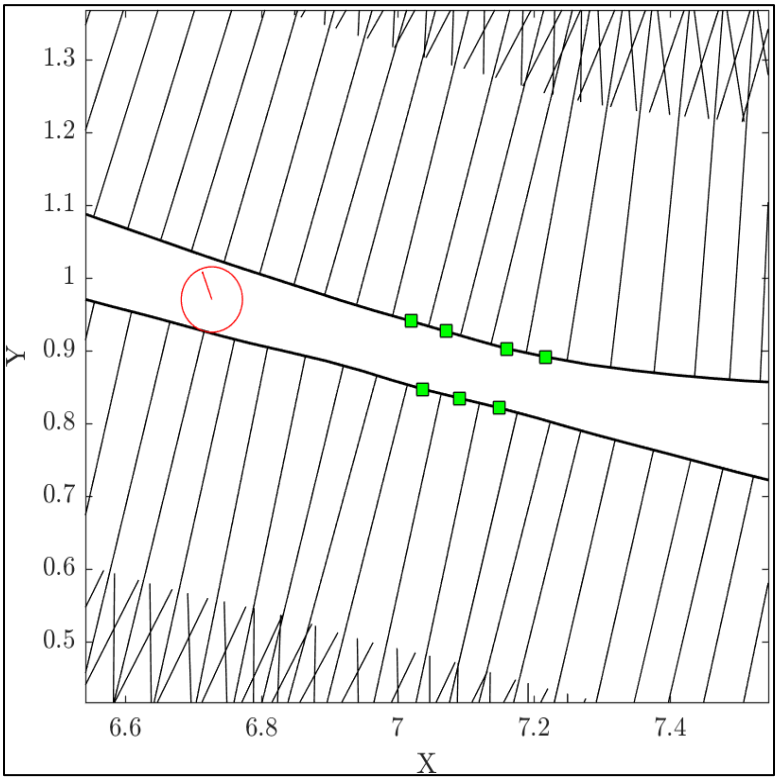
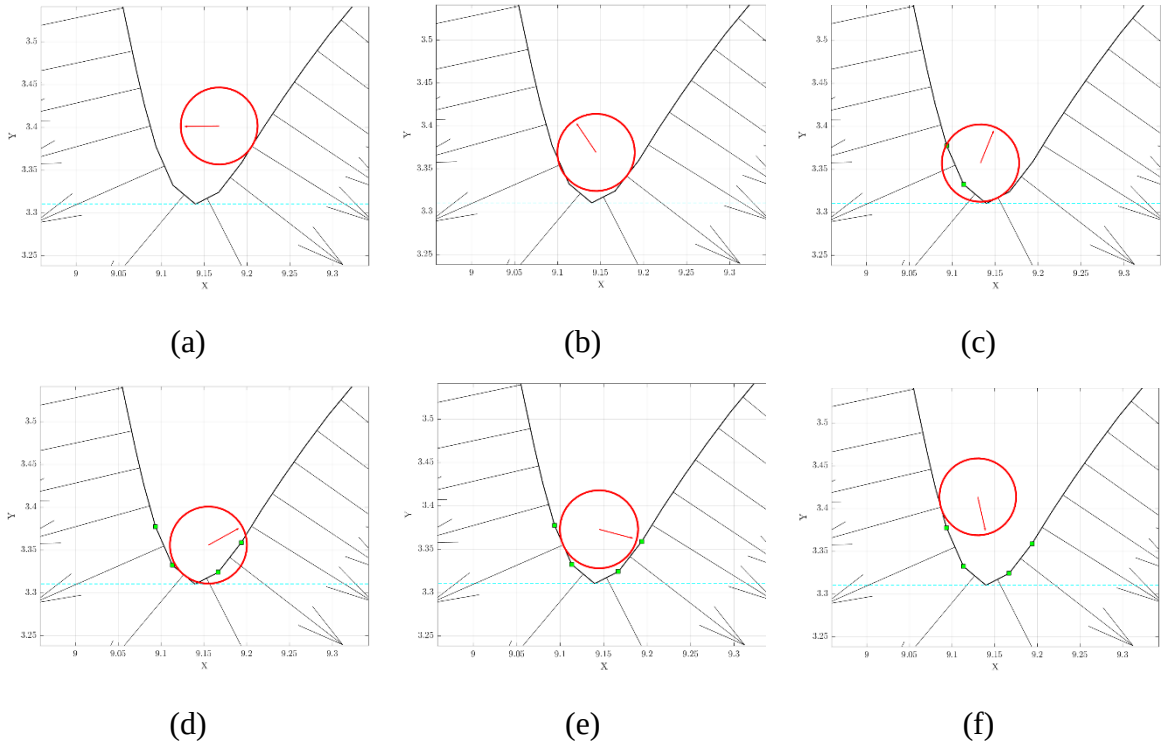


Figure 62: Zoom "C" into Figure 59

The failure to flag a tight turn is related to the circle's placement as it travels the inside of the curve. The circle's center is offset inwards from the midpoint by the prescribed radius. This, by definition, makes the circle tangent to the progenitor curve. Then, for edges that do not sufficiently "turn", the circle will not encompass their respective points. This is illustrated by the sequence of images (a) through (f) in Figure 63.



(a) (b) (c)

(d) (e) (f)

Figure 63: Rollercoaster failing to fully flag a tight turn.

This phenomenon is consistently reflected in the flag buffer as what would appear as two distinct bands of -1's with a single +1 in the middle. A remedy for this was found by "sinking" the offset of the rollercoaster circle's center by a small amount (say, 5% of the naïve offset). This drastically improves the performance of the flagging of tight turns. Although, it exacerbates the extent to which a false positive is flagged.

Table 14: Rollercoaster flag buffer example during failure to flag.

ID	1	**2**	**3**	**4**	5	**6**	**7**	**8**	9	10
Flag	+1	**-1**	**-1**	**-1**	+1	**-1**	**-1**	**-1**	+1	+1

2.7 Defect Correction

The last matter that pertains to manufacturability is the question of how to blend defects. In this work, two types of defects were encountered: Tight turns and self-intersections. Both can be detected by the rollercoaster algorithm using a sort-assisted search. Regardless, the target blend region is specified as two integer IDs, that of the point immediately before the defect, and that of the point immediately after the defect. A suitable blending curve must overwrite all points in between said integer ID's to finally make the offset operation from Concept 3 safe.

2.7.1 Circular Fillets

For the purposes of this work, the simplest way to correct a defect is using circular fillets, as the mathematical formulation is simple and the motion it entails on the printer's nozzle is the limit case for how tight it can turn. The idea is to use start and endpoints of the target segments to define lines and find where in between said lines a circle of desired radius can fit. Denote the line segments as S_1 and S_2:

$$S_1 = \begin{bmatrix} x_1(t_1) \\ y_1(t_1) \end{bmatrix} = \begin{bmatrix} x_1^s + t_1(x_1^e - x_1^s) \\ y_1^s + t_1(y_1^e - y_1^s) \end{bmatrix} \tag{27}$$

$$S_2 = \begin{bmatrix} x_2(t_2) \\ y_2(t_2) \end{bmatrix} = \begin{bmatrix} x_2^s + t_2(x_2^e - x_2^2) \\ y_2^s + t_2(y_2^e - y_2^s) \end{bmatrix} \tag{28}$$

Where the superscript "*s*" denotes a starting location, the superscript "*e*" denotes an ending location, and t_1 and t_2 are parameters that indicate the percentage along the segment. For either segment, when t equals 0, the coordinates $x(t)$ and $y(t)$ evaluate to the starting point. When t equals

1, the coordinates evaluate to the endpoint. The condition for a circular fillet is satisfied when the normals extruded from both S_1 and S_2 by the desired radius coincide. This can be written as two equations:

$$x_1^s + t_1\Delta x_1 - R\frac{\Delta y_1}{\|S_1\|} = x_2^s + t_2\Delta x_2 - R\frac{\Delta y_2}{\|S_2\|} \tag{29}$$

$$y_1^s + t_1\Delta y_1 + R\frac{\Delta x_1}{\|S_1\|} = y_2^s + t_2\Delta y_2 + R\frac{\Delta x_2}{\|S_2\|} \tag{30}$$

This forms a linear system in t_1 and t_2 which has the solution of the form:

$$t_1 = \frac{A_4B_1 - A_2B_2}{|A|} \tag{31}$$

$$t_2 = \frac{A_1B_2 - A_3B_1}{|A|} \tag{32}$$

Where the constants A_1, A_2, A_3, A_4, $|A|$, B_1, and B_2 are given by:

$$A_1 = x_1^e - x_1^s \tag{33}$$

$$A_2 = x_2^s - x_2^e \tag{34}$$

$$A_3 = y_1^e - y_1^s \tag{35}$$

$$A_4 = y_2^s - y_2^e \tag{36}$$

$$|A| = A_1A_4 - A_3A_2 \tag{37}$$

$$B_1 = x_2^s - x_1^s + R\left(\frac{\Delta y_1}{\|S_1\|} - \frac{\Delta y_2}{\|S_2\|}\right) \tag{38}$$

$$B_2 = y_2^s - y_1^s + R\left(\frac{\Delta x_2}{\|S_2\|} - \frac{\Delta x_1}{\|S_1\|}\right) \tag{39}$$

2.7.2 Limitations of the Circular Fillet

There are two problems with using circular fillets as a blending curve. The first one is that, even though an analytic solution exists (meaning that the system is deterministic), the problem is in practice ill-posed. There are four (4) possible circles that can be computed because two lines that cross create four quadrants as seen in Figure 64. In the example illustrated there S_1 is the segment between P_1 ($x_1 = 0$, $y_1 = 0$) and P_2 ($x_2 = 2$, $y_2 = 2$), S_2 is the segment betweenP_3 ($x_3 = -1$, $y_3 = 3$) and P_4 ($x_4 = 1$, $y = -1$), and the target fillet radius was $R = 1$.

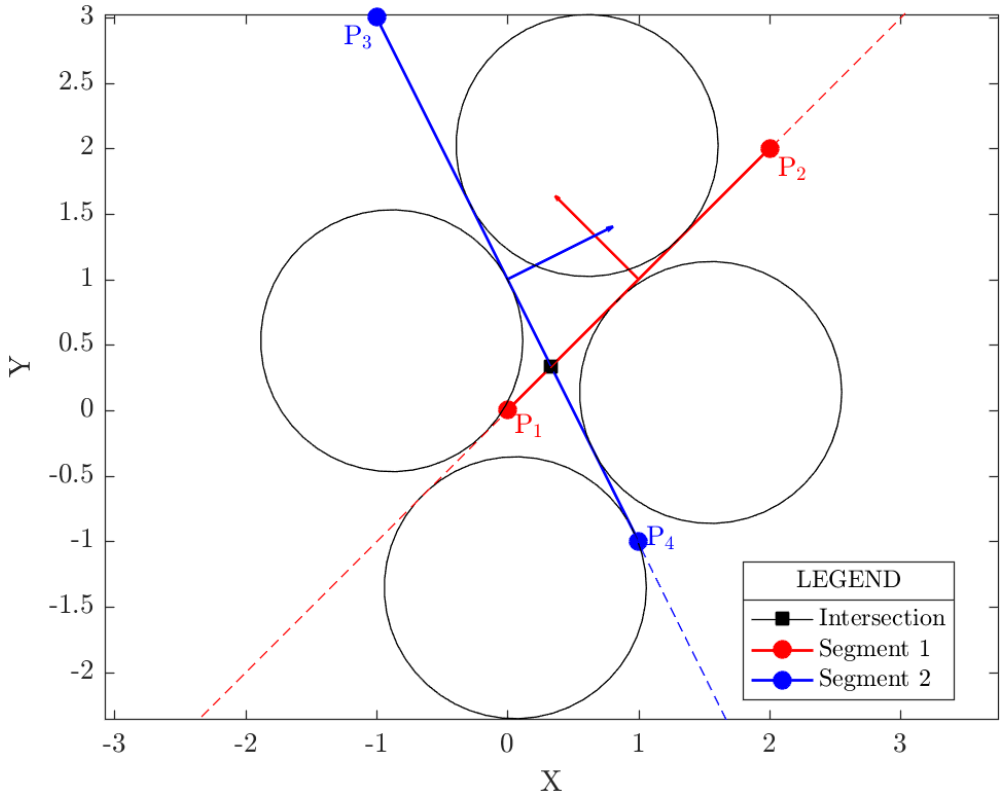


Figure 64: Four possible circles from which to generate fillets

The issue of selecting the correct side is addressed by specifying a direction that points towards the tight turn from the midpoint between the target segments. The dot products of this direction with the unit normals of both segments is computed. The condition for selecting the correct fillet out of the four possibilities is that the dot products both be negative. If a dot product is negative,

the corresponding segment (S_1 or S_2) must have its orientation reversed, meaning that the *s* and *e* subscripts are swapped in Equation (27) and Equation (28) if S_1 and or S_2 need reversal respectively.

The second limitation of the circular fillet is that it may require values for t_1 and or t_2 that are greater than one or less than zero. When this occurs, it means that it is impossible to fit a circular fillet through the segments. The fillet exists only as a mathematical construct that blends two imaginary infinite lines in this case. For the purposes of fiber manufacturability, the only way to keep on using circular fillets is by expanding the target band until the fillet is possible between the segments. To avoid further distortion of the curve due to successive circular filleting, the Least-Eccentric Ellipse (LEE) is discussed next.

2.7.3 Least Eccentric Ellipse (LEE) fillet

Circles are not flexible enough to accommodate four constraints, as they are defined by three quantities: a radius, an *x*-center, and a *y*-center. The next more flexible curve available is the ellipse, which can be defined by five quantities: an *x* center coordinate, a *y* center coordinate, a minor axis (b), a major axis (a), and an orientation (θ). This allows an ellipse to intersect two points with prescribed tangent directions, thus, elliptical fillets can be produced such that the target segments are kept.

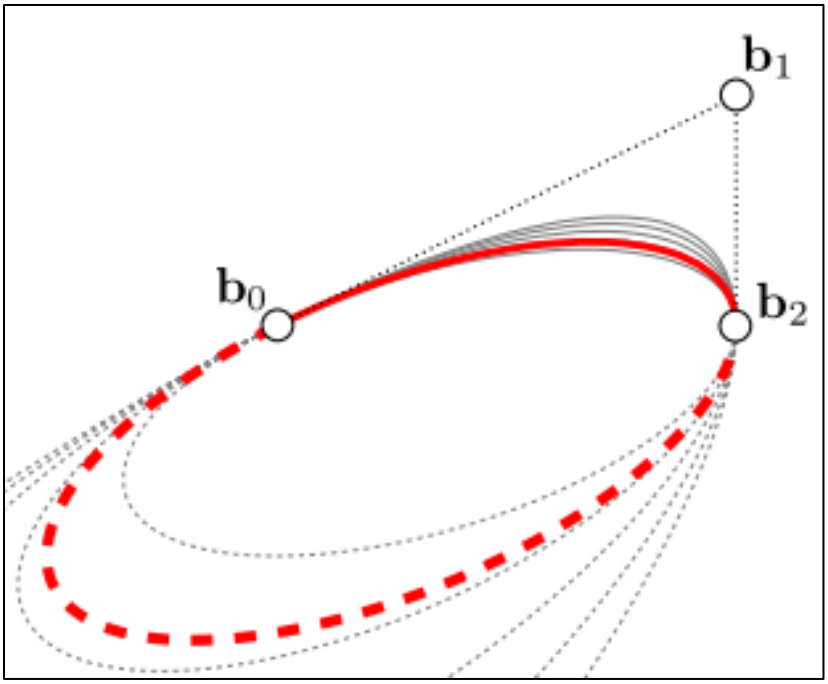


Figure 65: Infinite number of ellipses that blend two segments. Taken from [24]

The problem of finding an ellipse that intersects two points with prescribed tangency is underdetermined, meaning that infinitely many solutions exist (see Figure 65). The next most reasonable condition to impose is that the ellipse be as round as possible (to make the fillet closer to its circular counterpart). The roundest possible ellipse is that with minimum *eccentricity* (denoted with e) and in the context of finding elliptical fillets is termed Least-Eccentricity Ellipse [24]. The formulation of said ellipse is cumbersome but well studied. For details, the reader is referred to the work by Fermiani, Chuang, and Razdan. Using their procedure, however, the blending ellipse can be readily defined. Now, one must generate the blending arc, (i.e., the fillet).

2.7.4 Creation of LEE arcs

First, two segments (S_1 and S_2) are queried from the output of the rollercoaster algorithm. These segments provide the coincident points and the tangent directions. This information and the LEE formulas define the ellipse. The question is, how does one generate the appropriate elliptical arc?

To answer that question, first one needs a way to generate points on the ellipse. The formula used to do so is the polar parametrization [25]:

$$r(\theta) = \frac{b}{\sqrt{1 - (e\cos(\theta))^2}} \tag{40}$$

Where r is the radial distance from the ellipse's center to the ellipse with bearing θ measured counterclockwise from the major axis. By Inputting values of θ into equation (40), discrete points on the LEE can be generated if the orientation of the major and minor radii of the ellipse are known. The next question is, what range of θ values should be input? Obviously, the range of angles to input must correspond to the blending points. Therefore, vectors from the blending points to the center can be computed and then an angular measure is obtained using the dot product (Equation (41) below).

$$c \cdot d = c_x d_x + c_y d_y = \|c\|\|d\|\cos(\theta_{cd}) \tag{41}$$

Where c and d are two arbitrary 2D vectors, $\|c\|$ and $\|d\|$ are their Euclidean lengths (or "magnitudes") respectively, and θ_{cd} is the angle between them. Represent the blending points P_1 and P_2 by their coordinate pairs x_1, y_1 and x_2, y_2. Denote the center of the ellipse and orientation of the major axis by the pairs x_c, y_c and a_x, a_y respectively. Now form the vectors from the blending points to the center as follows:

$$V_1 = \begin{bmatrix} V_{x1} \\ V_{y1} \end{bmatrix} = \begin{bmatrix} x_c - x_1 \\ y_c - x_2 \end{bmatrix} \tag{42}$$

$$V_2 = \begin{bmatrix} V_{x2} \\ V_{y2} \end{bmatrix} = \begin{bmatrix} x_c - x_2 \\ y_c - y_2 \end{bmatrix} \tag{43}$$

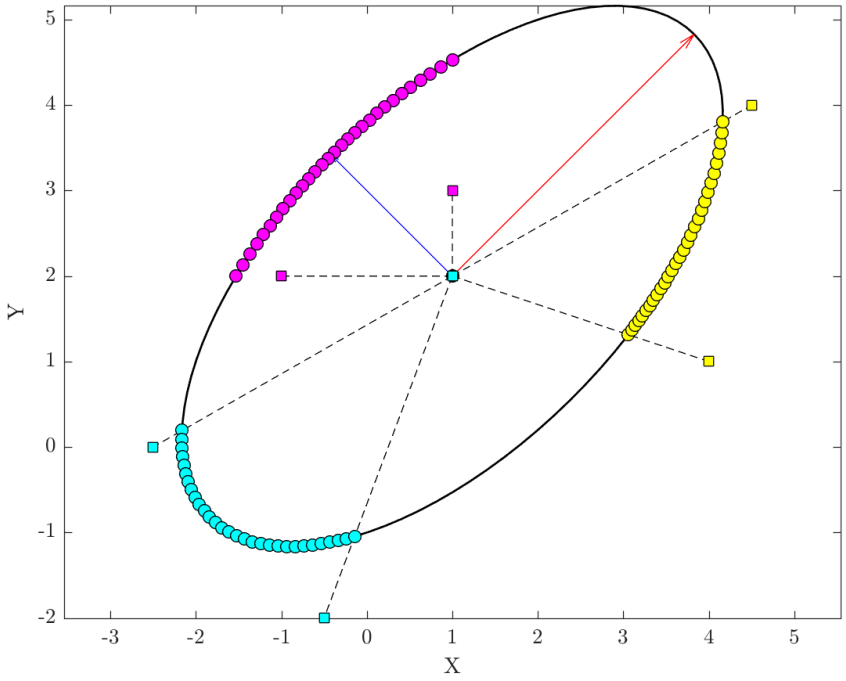


Figure 66: Examples of elliptical arcs

Then, solve for two angles by plugging into the rearranged versions of Equation (41) shown below. Note that a_x and a_y are components of a *unit vector*, thus $\|a\| = 1$ (and note the bounds on the angles):

$$\theta_1 = \cos^{-1}\left(\frac{a_x V_{x1} + a_y V_{y1}}{\|V_1\|}\right), \qquad \theta_1 \leq \pi \tag{44}$$

$$\theta_2 = \cos^{-1}\left(\frac{a_x V_{x2} + a_y V_{y2}}{\|V_2\|}\right), \qquad \theta_2 \leq \pi \tag{45}$$

The angles θ_1 and θ_2 are those subtended between the lines connecting the blending points to the center and the line along the orientation of the major axis respectively. These angles, however, are alone useless because they are akin to a direction without a starting point. At this stage, all that one knows is that V_1 and V_2 make angles θ_1 and θ_2 respectively with the major axis. This limitation is a consequence of the definition of the *inverse cosine* function ($\cos^{-1}$), it can only return values between 0 and π.

To produce the elliptical arc, one must know on what sides of the major axis angles θ_1 and θ_2 lie. To "orient" the angles θ_1 and θ_2 one can use the orientation of the minor axis (denoted by the pair of *unit vector* components b_x and b_y). Dot products of V_1 and V_2 with the minor axis reveal whether θ_1 and θ_2 (respectively) are on the side in the direction of the minor axis or opposite to the direction of the minor axis. This allows one to put signs on the angles θ_1 and θ_2 depending on the values of the dot products. The dot product formulas are now modified:

$$\theta_1 = -\operatorname{sgn}(V_1 \cdot b)\cos^{-1}\left(\frac{a_x V_{x1} + a_y V_{y1}}{\|V_1\|}\right), \qquad -\pi \le \theta_1 \le \pi \tag{46}$$

$$\theta_2 = -\operatorname{sgn}(V_2 \cdot b)\cos^{-1}\left(\frac{a_x V_{x2} + a_y V_{y2}}{\|V_2\|}\right), \qquad -\pi \le \theta_2 \le \pi \tag{47}$$

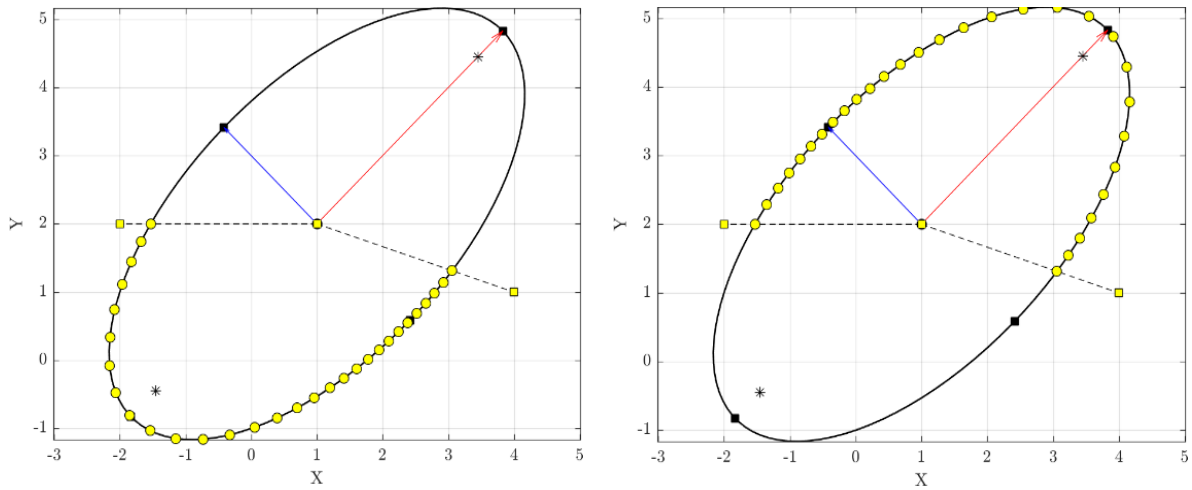
Where "sgn" is the *sign function*, which as the name suggests returns the sign of the input (either "+1" or "-1"). This lets one know whether the angles θ_1 and θ_2 are "above" or "below" the major axis. At this stage, one knows the range of angular values to plug into equation (40) to generate the elliptical arc. Points on the arc are then generated via:

$$\bar{x} = \begin{bmatrix} x \\ y \end{bmatrix} = r(\theta)\begin{bmatrix} \|b\| b_y \cos(\theta) - \|a\| a_y \sin(\theta) \\ \|b\| b_x \cos(\theta) - \|a\| a_x \sin(\theta) \end{bmatrix} \tag{48}$$

Where the angle θ is between θ_1 and θ_2, and r is obtained from Equation (40). Below (Figure 66) is an example of an ellipse in which three distinct arcs were generated using the procedure described by Equation (27) through Equation (29). Notice that the arc's generation depends on the angle subtended by points P_1 and P_2, which can be anywhere. The dotted lines show the angles subtended by arbitrary points when traced back to the ellipse's center.

There is, however, yet another ambiguity: There are two possible arcs that can be generated from this information. There is a *short* arc and then there is the *long* arc (Figure 67). For purposes

of preventing tight turns in fiber paths, the long arc is useless and only the short arc is desired. To guarantee that P_1 and P_2 produce the short arc, add 2π to θ_1 and θ_2 (individually) only if they are negative. Then find the difference between the angles. If the difference is greater than π, thus Equation (48) is now primed to generate the long arc. In that case, subtract 2π from the greater of θ_1 and θ_2 (after the initial addition of 2π) and proceed to evaluate Equation (48).



(a) (b)

Figure 67: Short (a) vs. Long (b) arc

At last, LEE fillets can now be created between two segments. Figure 68 and Figure 69 are examples of tight turns (flagged by the rollercoaster algorithm as green squares) blended with LEE fillets. Said figures showcase the culmination of applying concepts 1 through 6 to the flaws shown in Figure 32 and Figure 33 respectively.

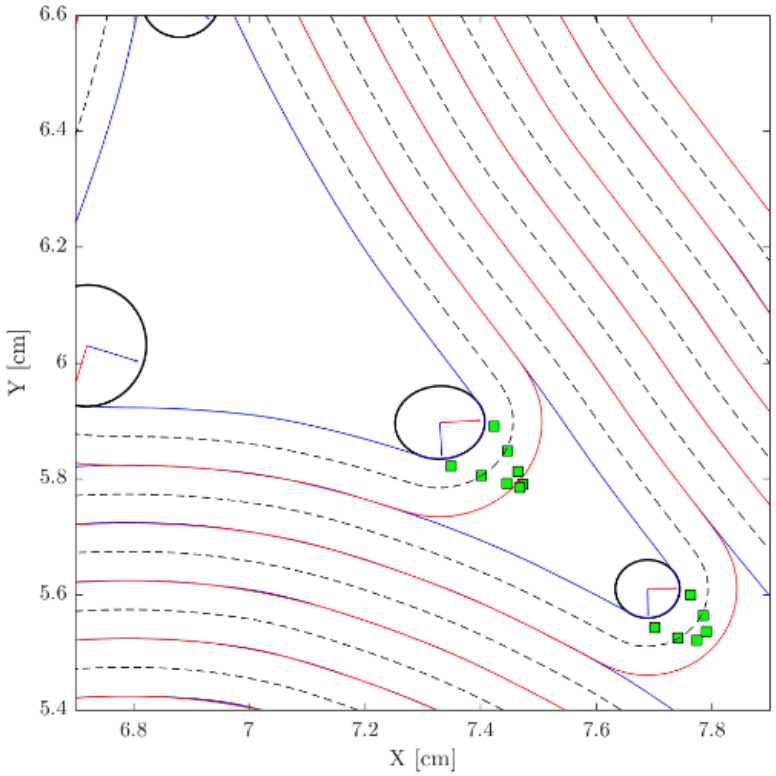


Figure 68: Applying Rollercoaster detection and LEE fillets to correct Figure 32

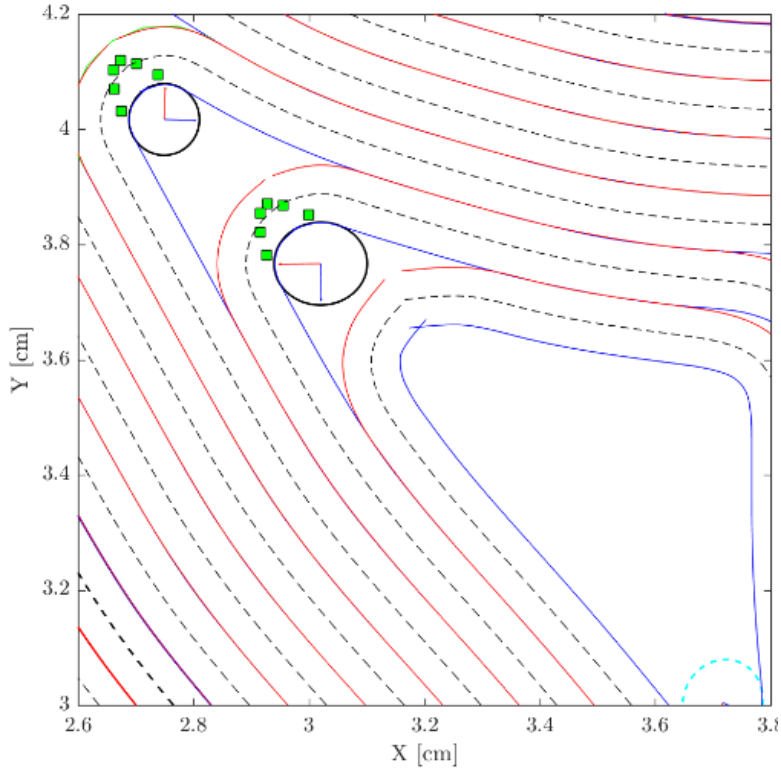


Figure 69: Applying Rollercoaster detection and LEE fillets to correct Figure 33

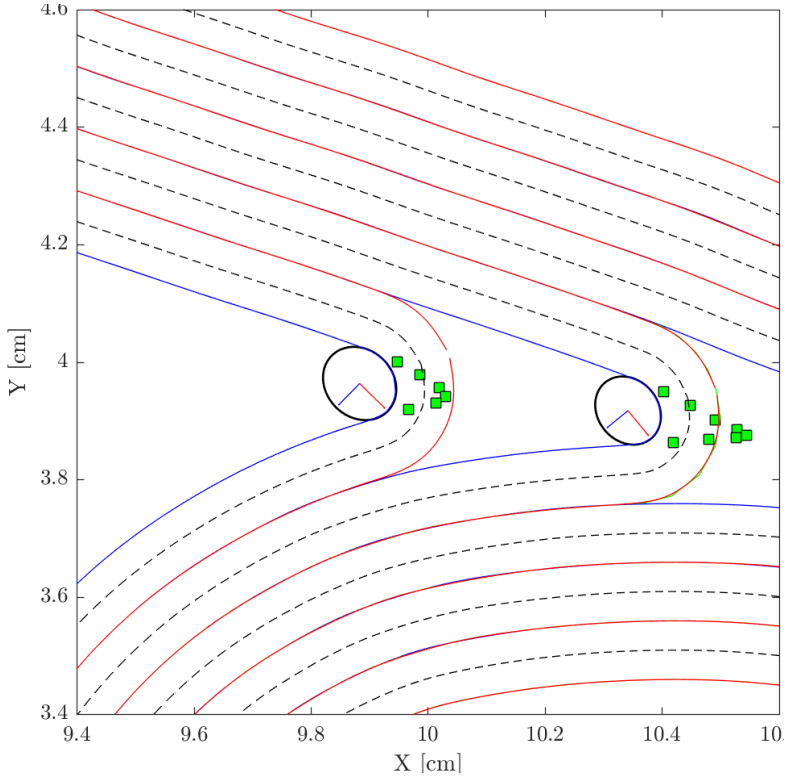


Figure 70: Applying Rollercoaster detection and LEE fillets to correct Figure 34.

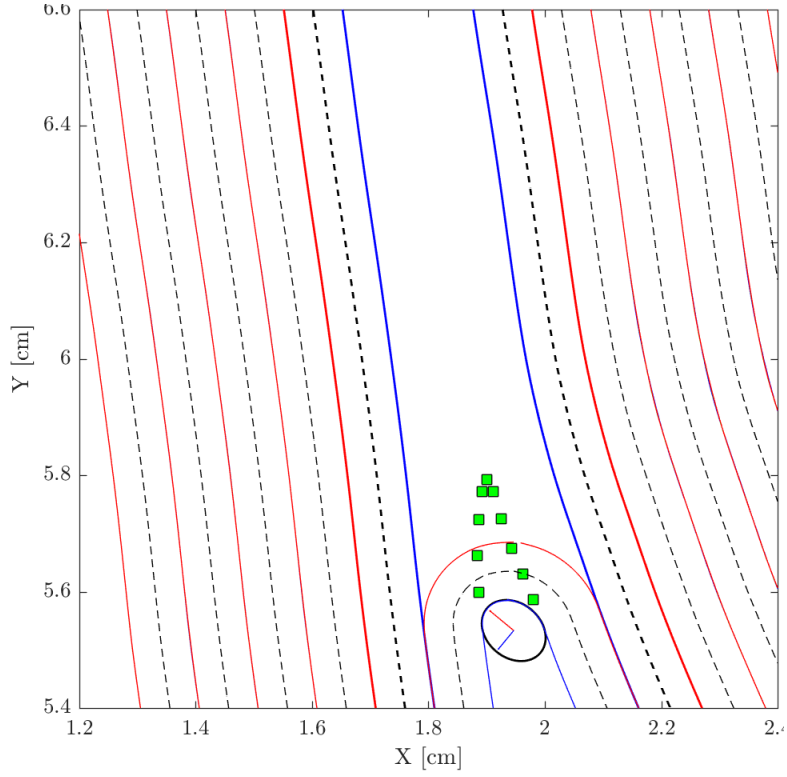


Figure 71: Applying Rollercoaster detection and LEE fillets to correct Figure 31

2.8 Parametric Representation of the Centerline Polyline

It is only after the incorporation of Concept 6 that the offset operation discussed in Concept 3 is finally safe to perform. Doing so produces a discrete curve that will be a candidate for defining the centerline of the fibers. One final caveat needs to be discussed to conclude discussion of Algorithm 1, and that is the question of how the fibers will be input to a 3D printing software to create the component.

2.8.1 Non-Uniform Rational Basis Splines (NURBS)

Quick recapitulation: The SDF level sets form the coordinates of what is termed the *progenitor* curve. This is a curve that will be offset to produce an approximation of the centerline path. Said approximation is termed the *inheritor* curve, as its geometric properties are determined from the progenitor. The intermediate objective of the offset operation is to generate the control curve for a grander geometric object, a so-called NURBS. The reason NURBS are brought to the picture is because they are the industry standard for communication of sophisticated parametric geometries in industry [26].

As a prerequisite for understanding NURBS, the reader should be familiar with the concept of a polynomial and that of a spline. If so, then think of NURBS as a weighted sum of splines, if not, then the reader is referred to “The NURBS Book” written by Les Piegl and Wayne Tiller for a background on both and then some. NURBS, just like with splines, are constructed from a sequence of “knots” and another sequence of “control points.” Typically, splines feature as many knots as control points, but this is not the case with NURBS as they feature more knots than control points [20]. Furthermore, NURBS require an additional sequence, that of so-called weights which act as local *attractors* or *repulsors*. NURBS are written as:

$$C(u) = \frac{\sum_{i=0}^{n} w_i P_i N_{i,p}(u)}{\sum_{i=0}^{n} w_i N_{i,p}(u)} \tag{49}$$

Where C is the NURBS curve, u is the knot parameter, P_i is a control point, w_i is the weight given to P_i, $N_{i,p}$ is a basis spline of order p associated with P_i. Figure 72 shown an example of a NURBS with a sequence of six (6) control points with the weight of the fourth point (labelled 3 as the numbering is zero-based) being changed.

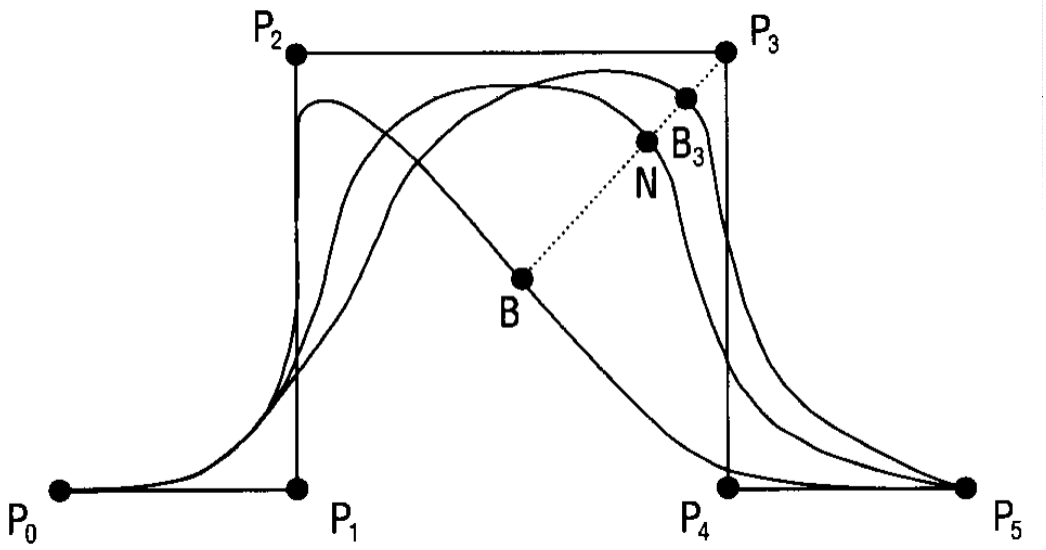


Figure 72: Effect of weights on a NURBS curve. Taken from [27]

2.8.2 Basis Splines

NURBS are characterized further by the degree p of the basis splines. The basis splines of degree p (denoted $N_{i,p}$) are defined recursively via the Cox-De Boor formula in terms of lower degree basis splines:

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{50}$$

Where the zeroth-degree basis splines are given by:

$$N_{i,0}(u) = \begin{cases} 1 & if\ u_i \le i \le u_{i+1} \\ 0, & otherwise \end{cases} \tag{51}$$

The basis splines can be constructed from the knot sequence and a given value of p using. Equation(51) and Equation (50). As an example take the following knot sequence u = [1,1,1,2,3,4,5,5,6,6,6] and p = 2, then the corresponding basis splines are shown in Figure 73.

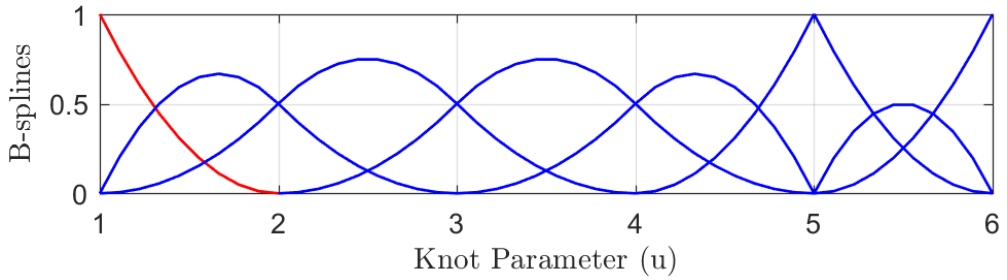


Figure 73: Example basis splines. Recreated from [26]

The knot sequence can have a few repeated entries, with the number of repetitions depending on the value of p. The only strict requirement for the knot sequence is that it be nondecreasing. Knot value repetition is discouraged as it makes the NURBS prone to forming *cusp*-like sharp turns.

2.8.3 Knot Sequence for Fiber Paths

When assembling a NURBS out of a control curve with n points, there must be $n + p + 1$ knots. The number of weights and basis splines is also n. In this work, the NURBS used to represent the centerline has a knot space that is evenly spanned. That is, the knot sequence looks like:

$$u_i = [0,0,0, \dots 0,0, a, b, \dots, z, \dots, 11,1]$$

Where the zeros and ones are consecutively repeated at the beginning and end respectively p times and a, b, …, z are nonzero. This repletion is needed because the basis splines do not partition unity at the endpoints otherwise. The latter term, “partition of unity” should be thought

of by the layman as a scheme's ability to interpolate. If the scheme (NURBS in this case) does not partition unity, well, it cannot be used as an interpolation technique. Regardless, because the knot span is evenly spaced, the knot sequences in this work are of the form:

$$u_i = \begin{cases} 0 & 0 \le i \le p+1 \\ \dfrac{i-p-1}{n-1}, & otherwise \\ 1 & n \le i \le n+p+1 \end{cases} \tag{52}$$

3. CREATION OF FEA INPUT FILE (ALGORITHM 2)

With the manufacturable fibers now available, it is time to generate a case file for FEA. Using the NURBS centerlines, a geometry-based scheme for assigning mesh elements material properties and fiber orientations is developed. The flow diagram for Algorithm 2 is shown in Figure 74. Algorithm 2 is substantially simpler than Algorithm 1, as it can be broken down into five concepts.

1) Meshing
2) NURBS evaluation
3) NURBS thickening
4) NURBS offset quadrangulation
5) Point-in-polygon

3.1 Description

The original topology optimization produced boundaries which define the shape of the optimized component. This shape is unaffected by Algorithm 1, so in Algorithm 2, the first step is to generate a spatial discretization of said boundaries. The mesh, however, lacks material and orientation information. To append this, auxiliary data to the input file, another geometry-based scheme that leverages the NURBS created in Algorithm 1 is developed. The NURBS are first thickened to produce the thickened fibers proper. The thickened fibers will consist of two NURBS offsets, called *upper* and *lower* offsets. Both offsets are evaluated at as many discrete locations as the centerline NURBS, therefore, the *upper* and the *lower* can be represented as polylines with equal number of *xy*-coordinates. The subsequent pairs of points along the *upper* and *lower* are used to create quadrilaterals that are overlayed on top of the mesh's elements. The idea is to check for all elements of the mesh are inside the quadrangles and append the fiber material ID to them, and the local NURBS orientation at the quadrangle. The quantity that determines whether an element

is inside the quadrilateral is the element's centroid. If the centroid is inside the quadrilateral, the element receives its auxiliary data from the local NURBS orientation. This process is carried out by solving a classic computational geometry problem, the point-in-polygon. Once all elements have been scanned, the data produced by Algorithm 2 can be used to customize the format of the FEA input file.

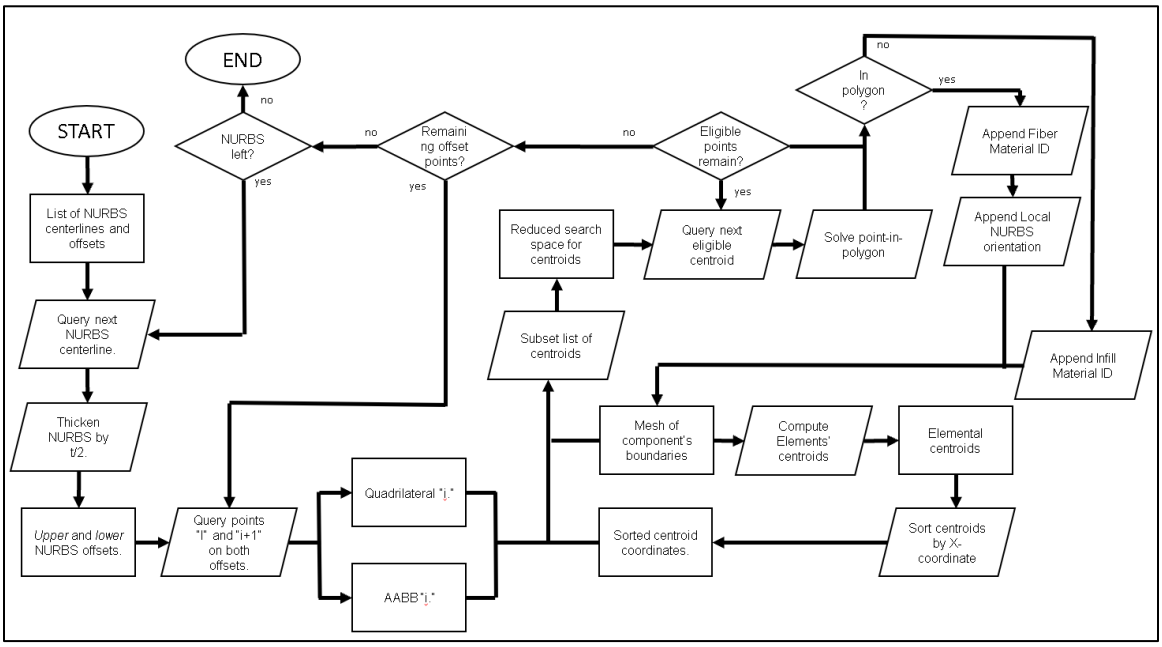


Figure 74: Flow diagram of Algorithm 2

3.2 Meshing

"Meshing" is a colloquial term used in field physical simulation to refer to the idea of discretizing a space. The idea of spatial discretization is to take a domain Ω and partition into smaller elements Ω_e such that the union of the partitions is equivalent to the original domain. This is written as:

$$\Omega = \bigcup_{e=1}^{n_{el}} \Omega_e \tag{53}$$

Where n_{el} is the number of elements that make up the spatial discretization. Recall the domain shown in Figure 11. This was meshed according to an advancing front algorithm conceptually like the SDF of Concept 1, Algorithm 1 and is shown in Figure 75. Recall Figure 14, which was used to discuss the difference between *active* vs. *inactive* boundaries. The meshing algorithm had all boundaries toggled as *active*.

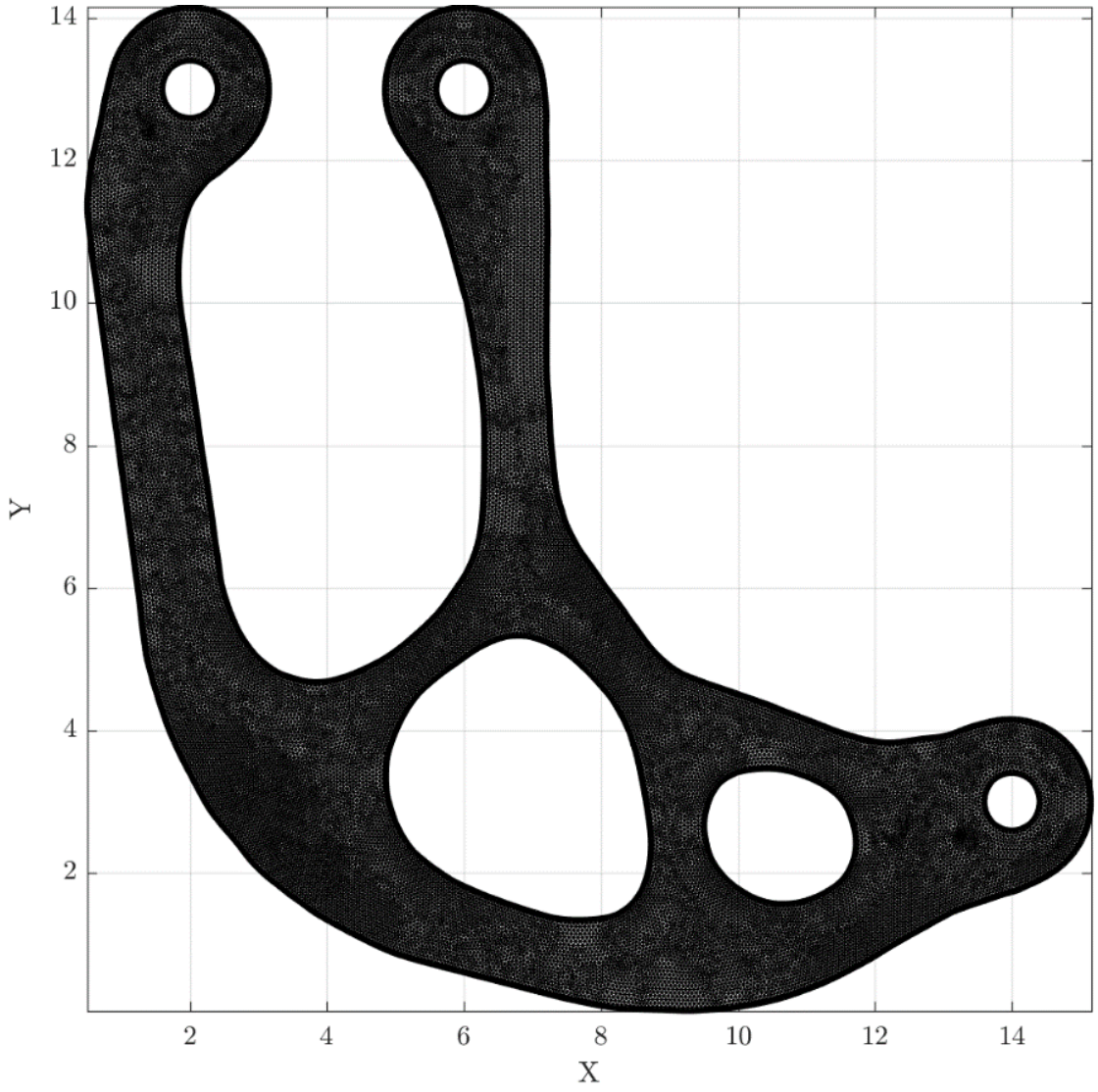


Figure 75: Mesh of the test geometry

The mesh used in Algorithm 2 consists of triangles only, that is all Ω_e in Equation (53) are triangles. The mesh generator was part MATLAB's PDE Toolbox. Pertinent to Concepts 9 and 10

of Algorithm 2, the only relevant aspect of a triangular mesh is the collection of centroids of the elements. The centroid of any triangle is simply the unweighted average of its coordinates:

$$x_{c,i} = \frac{1}{3}\left(x_{i,1} + x_{i,2} + x_{i,3}\right) \tag{54}$$

$$y_{c,i} = \frac{1}{3}\left(y_{i,1} + y_{i,2} + y_{i,3}\right) \tag{55}$$

Where *i* is an index used to denote a triangular element.

3.3 Centerline NURBS evaluation

Recall the NURBS curve introduced in Concept 7 of Algorithm 1. Defining sequence of control points, knots, and weights, produces a rational curve $C(u)$. The sequences alone define an idealized continuous shape, however, there is a matter of querying values of $C(u)$ at discrete values of u. In this work, a *granularity* parameter "*g*," defined as the number of uniformly spaced query points between knot spans is used. When *g* is zero, $C(u)$ is evaluated at the values of the knots that make up the defining sequence. Values of *g* greater than 0 will quickly refine the discrete evaluation of $C(u)$.

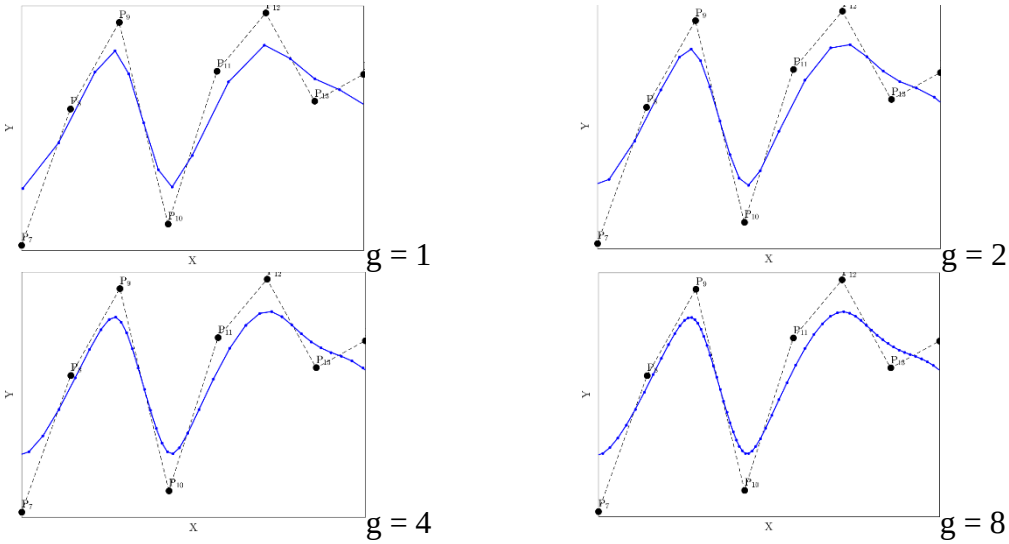


Figure 76: Effect of the granularity parameter on the fineness of a NURBS

3.4 Centerline NURBS Thickening

The NURBS centerline is deemed to be the actual path followed by the printer's nozzle. That is, the inheritor curve that was produced by Algorithm 1 was but a steppingstone. Therefore, any thickness should be offset on both sides of the centerline NURBS. In a sense, the SDF contours created in Algorithm 1 are discarded and now replaced by one of the NURBS offsets when talking about the "inside" of a fiber.

3.4.1 NURBS Thickening as Two Analytic Offsets

The NURBS *thickening* operation is conceptually like the *offset* operation discussed in Concept 3 of Algorithm 1 in the sense that progenitor curves were used to produce inheritor curves. Here, a progenitor NURBS is used to create two inheritor discrete curves, termed *upper* and *lower*. Both the *upper* and the *lower* are produced by a NURBS *offset* operation, that is, the NURBS *thickening* is simply a pair of NURBS *offsets*. In principle, a NURBS *offset* is identical to that of a discrete curve. The key difference is that the unit normal vectors are obtained from $C(u)$ and the resulting O_i points become the offset instead of the intersection of L_i with L_{i+1} (recall Figure 42). The NURBS offset is defined as:

$$C^0(u) = C(u) + dN(u) \tag{56}$$

Where $C^0(u)$ is the discrete offset curve, d is an offset distance, and N is the function for the unit normal vectors along $C(u)$. When the function for $N(u)$ known, the NURBS offset operation proceeds by evaluating Equation (56) over the entirety of the *evaluated* NURBS centerline. The intuition for the naming of the offset curves *lower* and *upper*, is the fact that Equation (56) is first evaluated with d = -t/2 and then with d = +t/2 respectively.

3.4.2 Parametric Unit Tangent and Unit Normal

The simplest way to obtain $N(u)$ is by taking the first derivative of $C(u)$ across the x and y dimensions. The intermediate result is a vector tangent to $C(u)$ that is colloquially referred to as "velocity." The magnitude of this vector is not the *unit*; therefore, it must be normalized. $N(u)$ then can be evaluated by rotating the normalized tangent vector by 90 degrees either counterclockwise or clockwise. The unit tangent vector to $C(u)$, denoted $T(u)$ is obtained via:

$$T(u) = \begin{bmatrix} T_x(u) \\ T_y(u) \end{bmatrix} = \frac{1}{\sqrt{[C_x'(u)]^2 + \left[C_y'(u)\right]^2}} \begin{bmatrix} C_x'(u) \\ C_y'(u) \end{bmatrix} \tag{57}$$

Where the primes on $C_x(u)$ and $C_y(u)$ denote derivatives with respect to the knot parameter u. Then, $N(u)$ follows from a 90-degree rotation (counterclockwise in this example):

$$N(u) = \begin{bmatrix} N_x(u) \\ N_y(u) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} T_x(u) \\ T_y(u) \end{bmatrix} = \begin{bmatrix} -T_y(u) \\ +T_x(u) \end{bmatrix} \tag{58}$$

In Figure 77 $T(u)$ and $N(u)$ are shown for a spiraling NURBS. Recall the figures that showcased Rollercoaster detection and LEE blending (Figure 68 through Figure 71), the thickened fibers there are the product of the NURBS offsetting procedure described here.

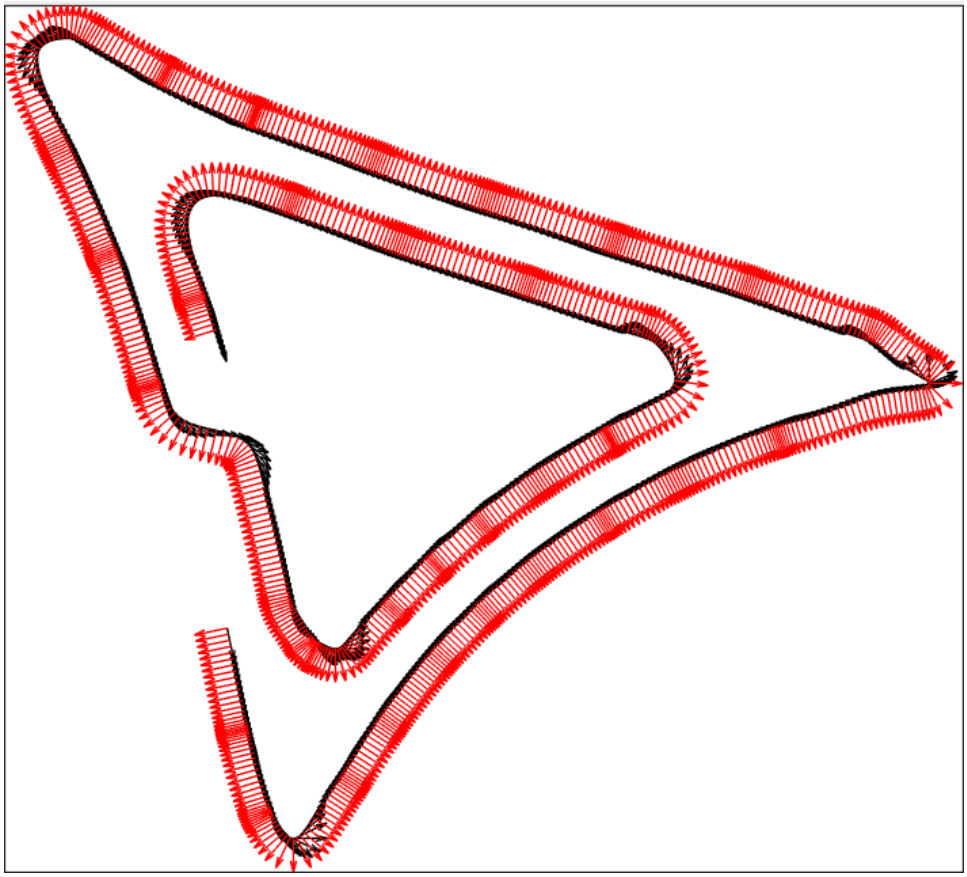


Figure 77: Illustration of N(u) in red and T(u) in black.

3.4.3 Concerning the Extension to 3D in Future Work

The unit vectors $T(u)$ and $N(u)$, when input to the vector cross-product operation produce a vector $B(u)$, called the *binormal* vector. The trio of $T(u)$, $N(u)$, and $B(u)$ form the so-called Frenet-Serret frame (named after its discoverers). Implicit in the formulation of Equation (58) is that $B(u)$ is known, namely, it was assumed to point out of the 2D plane according to a right-handed coordinate system. The reader is warned that Equation (58) does not extend to 3D and is more of a low-cost alternative to Equation (59) below, the "correct" way of finding $N(u)$.

$$N(u) = \frac{dT(u)}{du} = -\frac{C_x''(u) + C_y''(u)}{\left([C_x'(u)]^2 + [C_y'(u)]^2\right)^{\frac{3}{2}}}\begin{bmatrix} C_x''(u) \\ C_y''(u) \end{bmatrix} \tag{59}$$

However, use of this formula in 2D is discouraged because NURBS are rational expressions which make the ensuing algebra formidable to write and expensive to evaluate. Equation (59) holds for 3D space will be needed if any *curvature* or *torsion* analysis is required in future work. Equation

3.5 NURBS Quadrangulation and Overlay

With NURBS centerlines now thickened, the fibers are fully represented. Thanks to the prior. treatment of the SDF contours, the NURBS-represented fiber paths are now supposed to be flawless. However, one must now reconcile the mesh created in section 3.1 with the thickened NURBS from section 3.3. Specifically, one must use the fibers to assign material IDs and local orientations to the mesh element. To do so, a "quadrangulate and overlay" scheme is developed. The gist of it is that the points on the *upper* and *lower* are used to define quadrangles that are literally overlayed onto the mesh. The purpose of this quadrangulation is to establish the local properties of the elements based on the fiber segments that are overlayed on top of them.

The quadrangulate and overlay scheme is illustrated by Figure 78 and its exploded views (Figure 79through Figure 86). The exploded views coincide with those shown in Figure 29 through Figure 36. The element centroids computed using Equation (54) and Equation (55) are painted red.

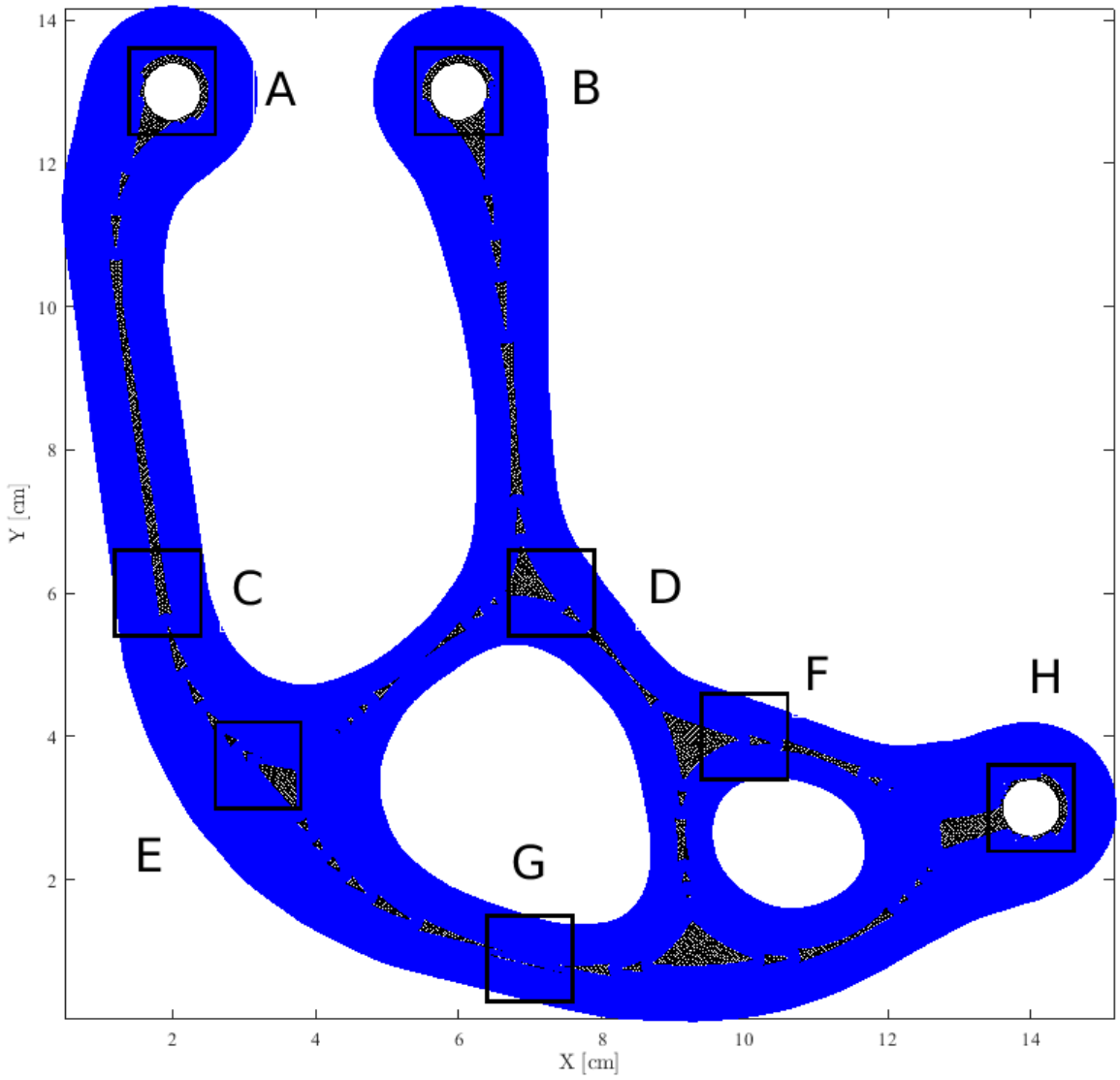


Figure 78: Quadrangulated fibers overlayed on top of mesh

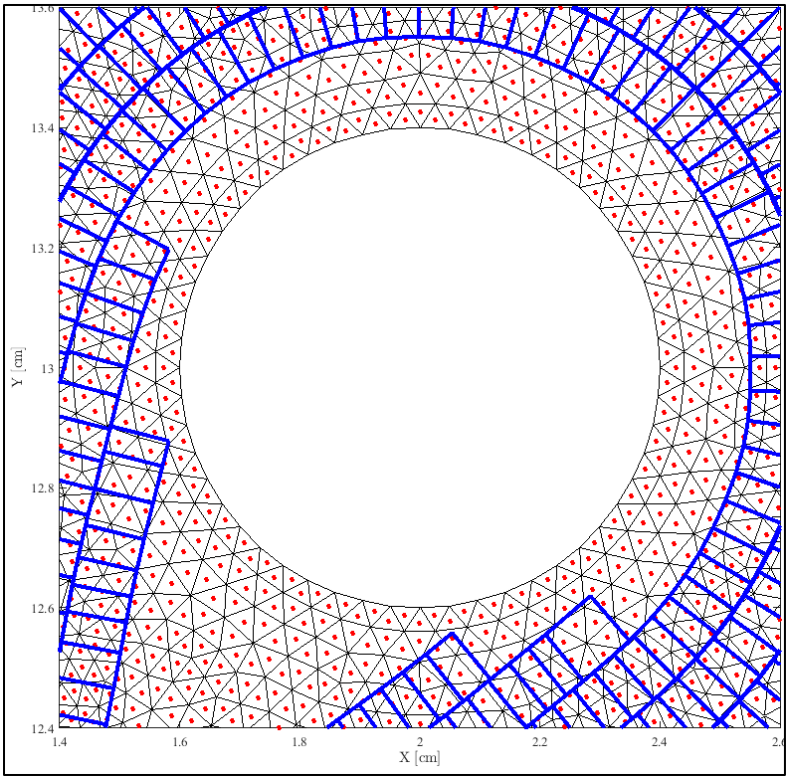


Figure 79: Zoom "A" into Figure 78

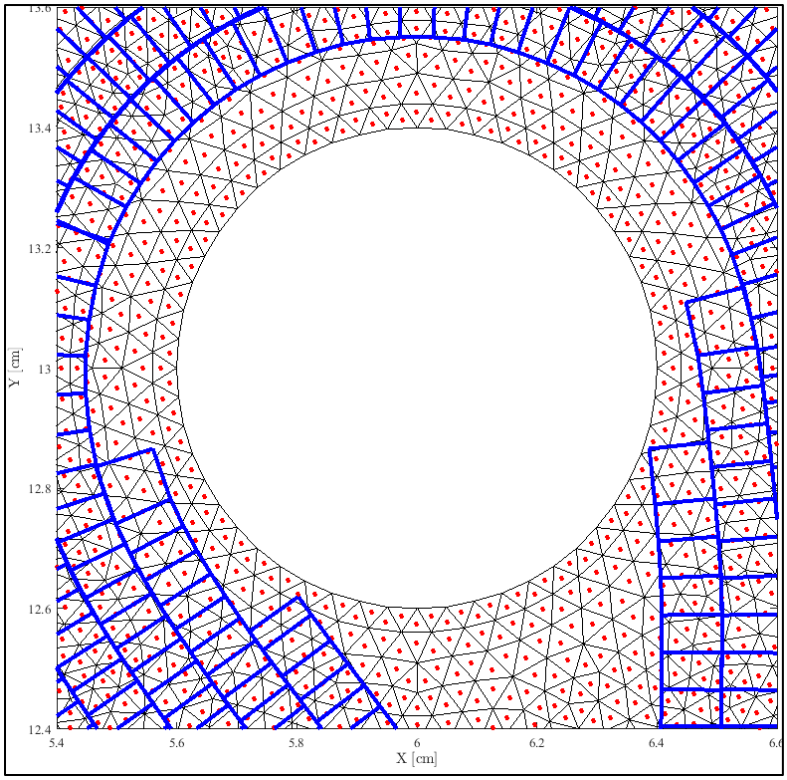


Figure 80: Zoom "B" into Figure 78

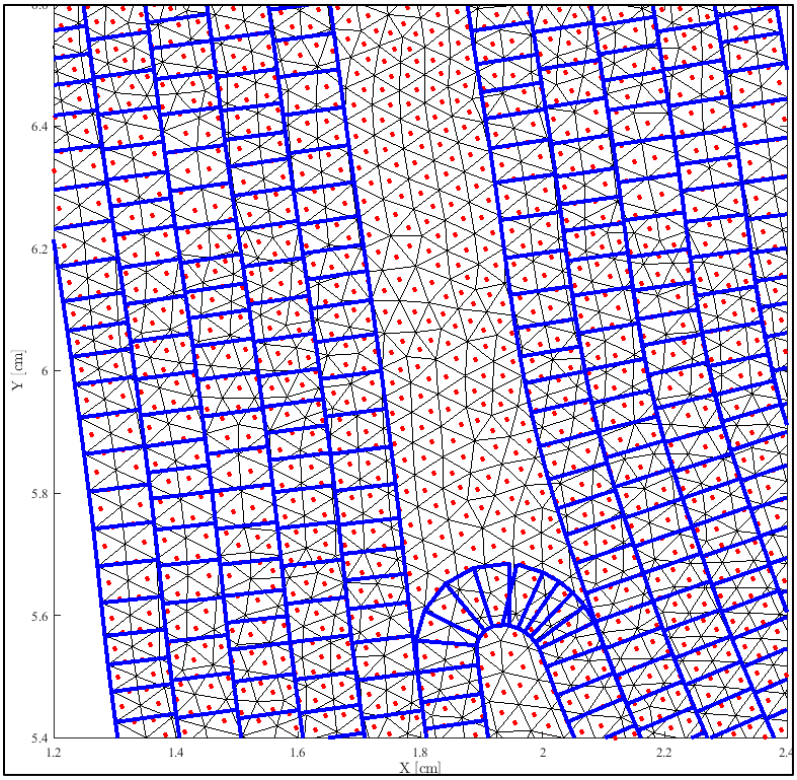


Figure 81: Zoom "C" into Figure 78

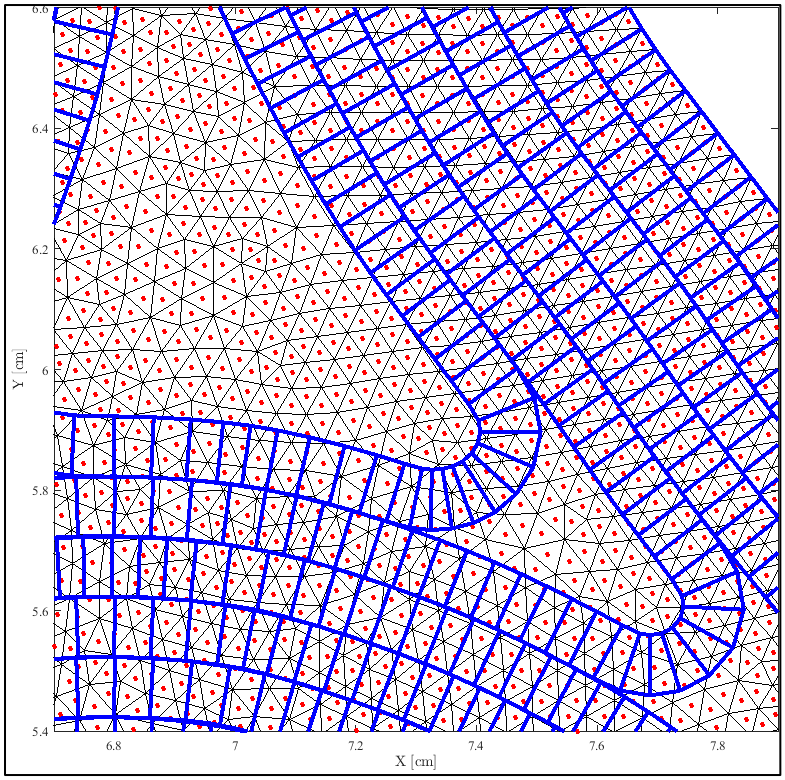


Figure 82: Zoom "D" into Figure 78

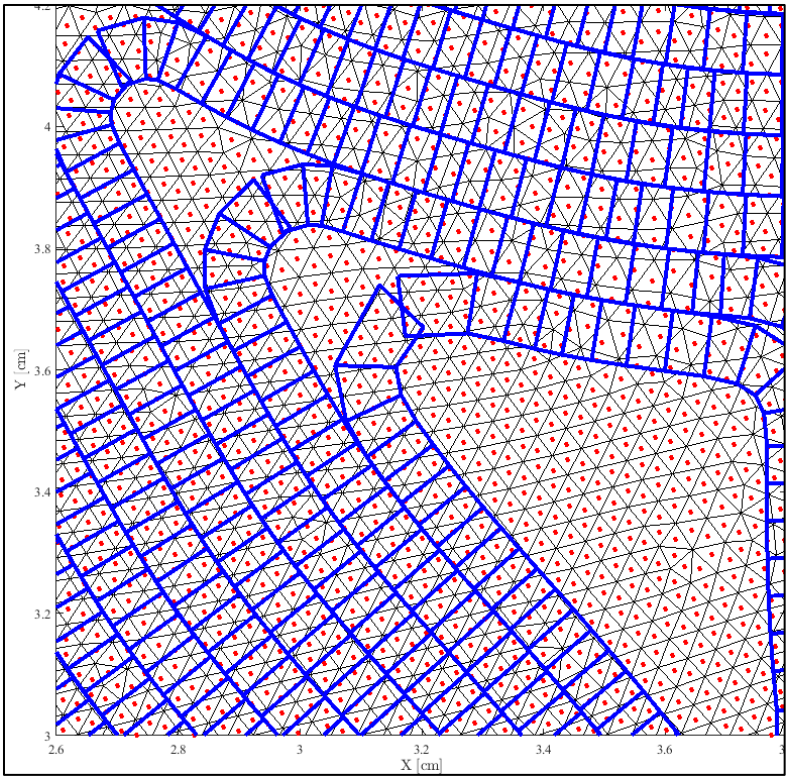


Figure 83: Zoom "E" into Figure 78

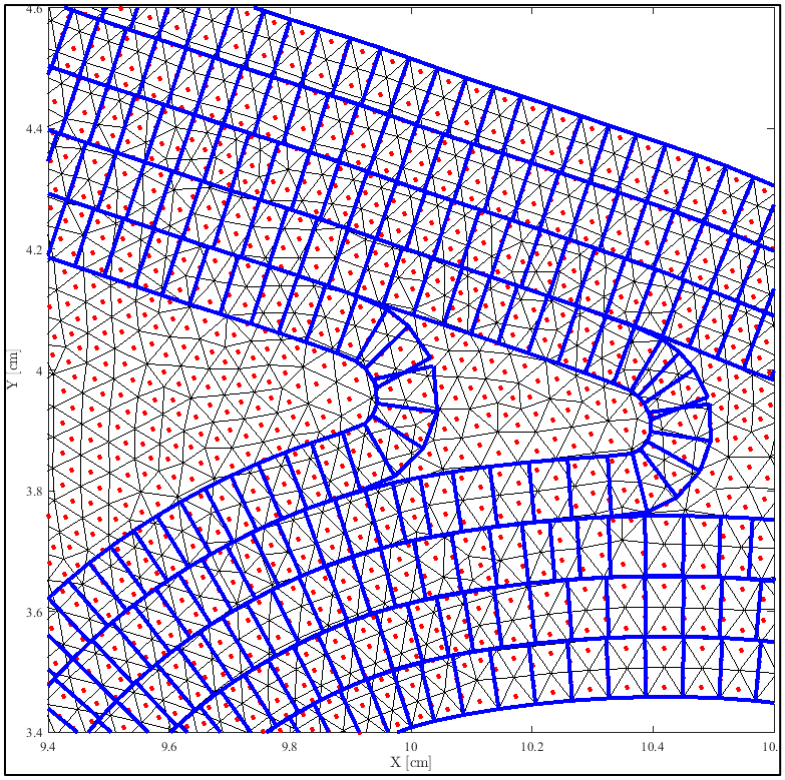


Figure 84: Zoom "F" into Figure 78

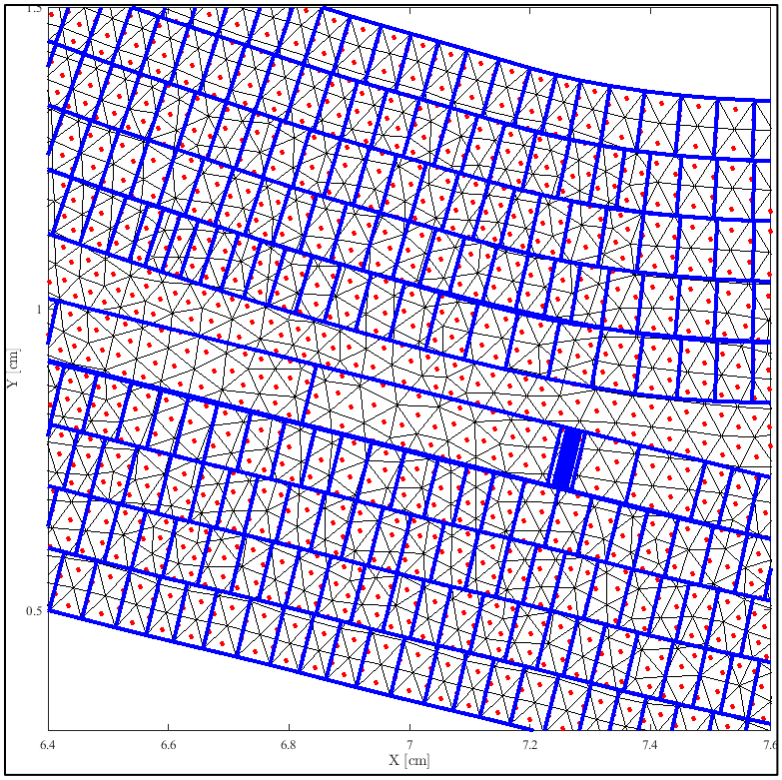


Figure 85: Zoom "G" into Figure 78

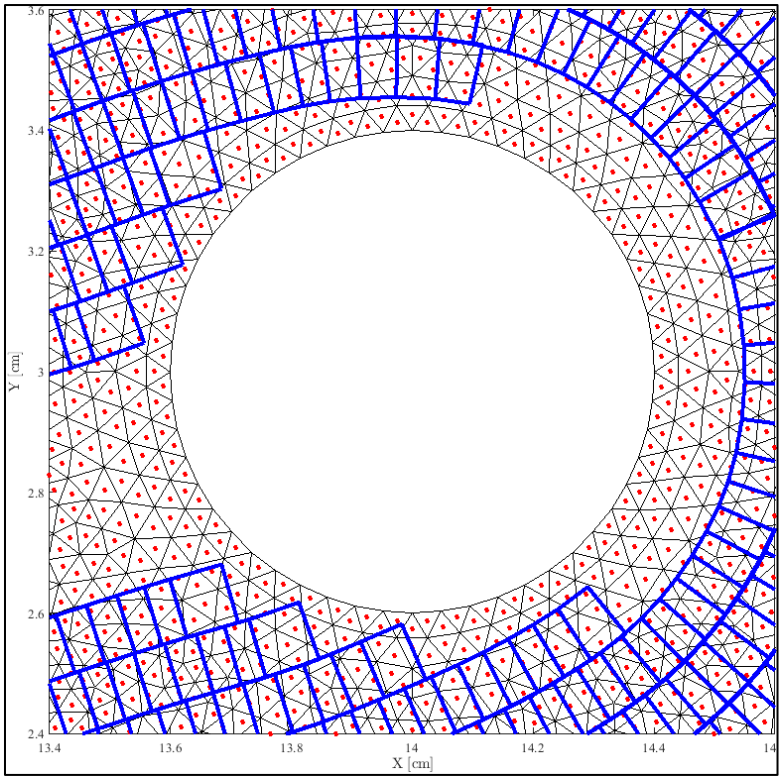


Figure 86: Zoom "H" into Figure 78t

3.5.1 Description of Fiber Segments

To describe the quadrangles, take four (4) points and call them *A*, *B*, *C*, *D* denoted by the coordinated pairs x_A, y_A, and x_B, y_B, and x_C, y_C, and x_D, y_D respectively. Points *A* and *B* are every two adjacent points on the *upper* offset, and points *C* and *D* are any to adjacent points on the *lower* offset. More explicitly, points *A* and *C* are labelled "i" on their respective curves, and points *B* and *D* are labelled "i+1" on their respective curves. The idea is to form quadrilateral *ABCD* for every pair of "i" and "i+1" points across the NURBS offsets. Then one can find the elements whose centroids fall within the quadrilateral. Said elements receive a material flag corresponding to the fiber's material, and an angle. See Figure 87 for an illustration.

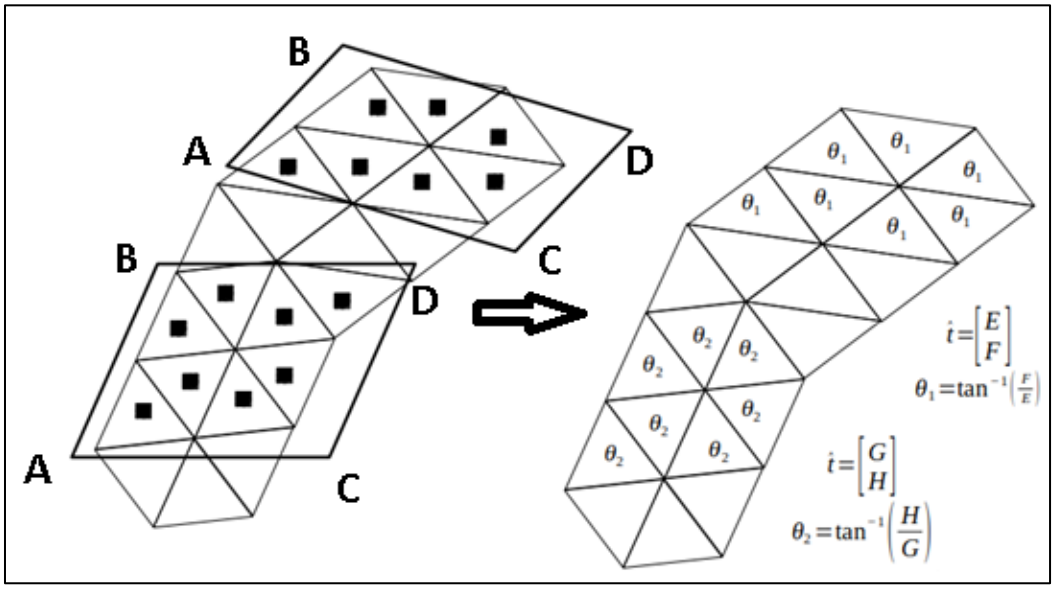


Figure 87: Assigning angles to elements from NURBS quadrangulation

3.6 Point-In-Polygon

Now, say one has point *P*, the centroid of a finite element computed using Equation (57) and Equation (58). One must determine whether the centroid is within a particular quadrilateral *ABCD* as shown in Figure 87. There are multiple ways to determine this. One way is through *ray casting*, which entails assuming a direction (say, *E*) such that one forms the line *PE*. One uses this

imaginary line to perform intersection checks with the sides of *ABCD*. That is *PE* is checked for intersection against segments *AB*, *BC*, *CD*, and *DA*. If the number of intersections is odd, then the point is inside *ABCD*, else, the point is outside [16]. Take Figure 88 as an example. There, the point on the inside has been given three possible rays, all of which intersect one side (odd number of intersections). The points on the outside can be given a ray direction but it will either not intersect any sides or intersect two sides (i.e., not an odd number of intersections.

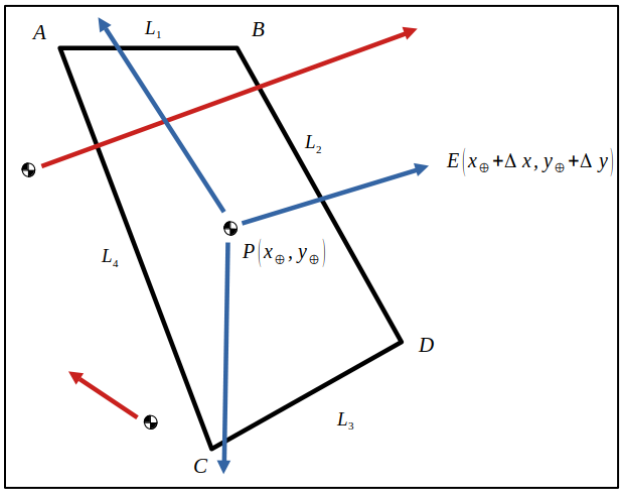


Figure 88: Ray casting on a quadrangle

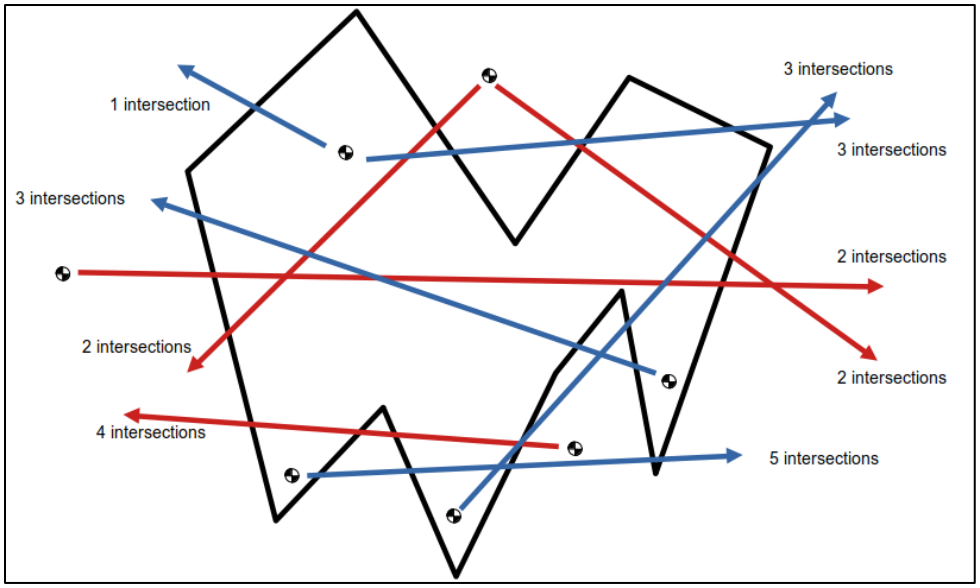


Figure 89: Ray casting on an arbitrary polygon

The problem that is being discussed here is known in computer science as the *point in polygon* problem and has been documented for a long a time. It remains an active field of research as polygons of arbitrarily complex shapes are being used an investigated in industry. [28]

3.6.1 Cost of Ray Cast for Point-In-Polygon

The ray casting technique is robust and will generalize to any concave polygon (see Figure 89), however, it is expensive because of the intersection checks. One intersection check essentially constitutes the evaluation of Equation (10) through Equation (14). Each intersection check entails 12 addition operations, 5 multiplication operations, 2 division operations, and 4 logical comparisons (because one must ensure that both z_i and z_{i+1} are less than or equal to 1 and greater than or equal to 0). However, that is one intersection check, one must check all four sides, thus, the total effort for ray casting a single elemental centroid against a NURBS quadrangle is 48 addition operations,20 multiplication operations, 8 division operations, and 16 logical comparisons.

3.6.2 A Lower Cost Alternative to Ray Casting

Since the inclusion of point *P* is assessed against *convex quadrilaterals*, the following simpler approach can be used: Form segments *AP*, *BP*, *CP*, and *DP*. And compute the following dot products: *AB*·*AP*, *AD*·*AP*, *BA*·*BP*, *BC*·*BP*, *CB*·*CP*, *CD*·*CP*, *DC*·*DP*, and *DA*·*DP*. If all eight (8) of the dot products are positive, then point *P* is inside, if any of the eight dot products is negative, *P* is outside. The intuition for this logic is that the vectors defined in the dot products all point from the vertices of *ABCD* to the point. Since the NURBS quadrangles are convex, the two vectors that any one vertex *A*, or *B*, or *C*, or *D* makes with the quadrilateral *ABCD* are both mostly in the direction of the vector towards an interior point. This approach works only for nearly rectangular polygons and does not generalize to *convex* or *concave* polygons like ray casting does. The cost of

this approach is 8 additions, 16 multiplications, 0 divisions, and 8 logical comparisons. Across the four different operations, this is a 66% reduction in addition operations, 60% reduction in multiplications, 100% reduction in division operations and 50% reduction in logical comparisons when compared against ray casting.

3.6.3 Sort-Assisted Search with Axis-Aligned Bounding Boxes

Even though a simple point vs. quadrilateral inclusion test is available there is a time complexity issue at hand that stems from the fact that one (and a computer) cannot distinguish overall shape or proximity of the points *P* to some quadrilateral from simply looking at a sequence of *xy* coordinates. Without knowing some aspect of the shape, one's only recourse is to perform a crippling O(nm) task, where *n* is the number of elements, and *m* is the number of quadrilaterals. If the element size is approximately equal to the fiber thickness, then $m \approx n$.

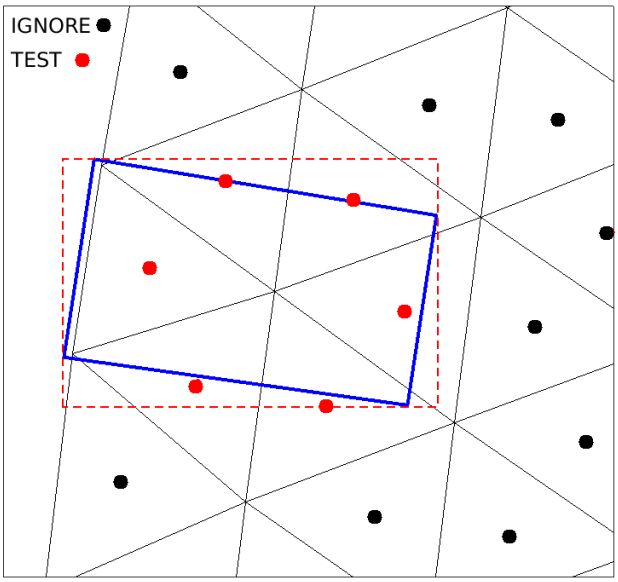


Figure 90: AABB (in red) used to reduce number of point-in-polygon tests.

To speed up this task, the search is sort-assisted in the same way that the rollercoaster algorithm was. However, in the rollercoaster one was dealing with a simple circle. The radius and center of

the circle were used to filter out the points that were obviously nowhere near the circle, thus, a reduced number of inclusion checks were performed. Said exclusion technique was powered by an implicit Axis-Aligned Bounding Box (AABB). Here, the same concept will be re-leveraged as it is more important than before. The AABB is instead constructed by finding the maximum and minimum coordinates of the quadrilateral.

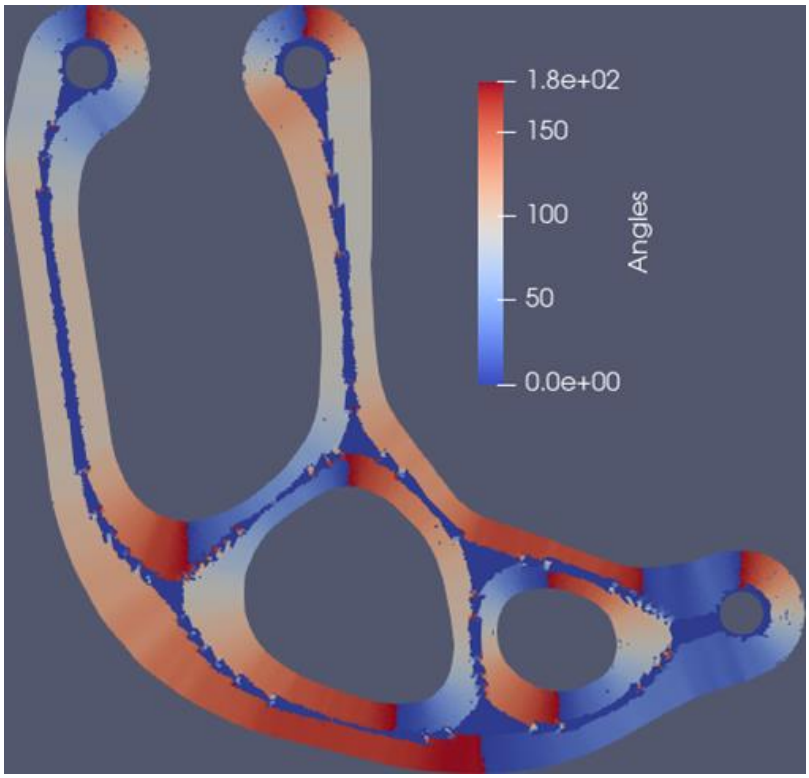


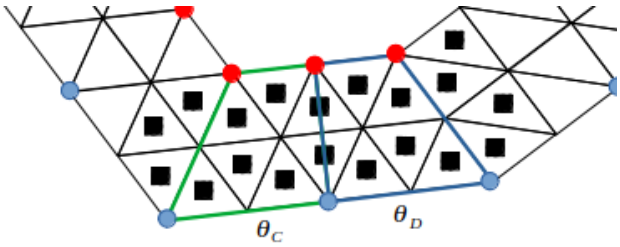
Figure 91: Algorithm 2 deployed on the test mesh.

The centroids of the triangular elements are first sorted according to their *x*-coordinates. The *x*-bounds of the AABB are then used to determine a substantially narrow (when compared to the entirety of the mesh) range of centroids that are candidates for the point-in-polygon test. Within this subset of the centroids, the *y*-bounds of the AABB are then used to throw out any centroids that are not within the AABB. The remainder of the centroids, will, be inside the AABB and are thus reasonable to have the point-in-polygon check done on them. Applying Algorithm 2 to the

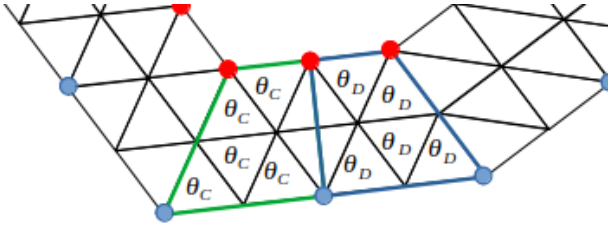
test mesh results in Figure 91, where the negative angles were added to +180 degrees to ensure that the Euler angles are nonnegative.

3.6.4 A Caveat with the Simplified Point-In-Polygon Approach

The point-in-polygon procedure described in section 3.5.2 is not robust, as it is prone to fail when the point lies arbitrarily close to one of the quadrangle's edges. This is because implied to the algorithm's success is that the quadrangle be in fact a *rectangle*, meaning that all interior angles are of ninety degrees. The NURBS quadrangles are not perfect rectangles, so, the fringe case whereby the centroid of an element lies essentially on one of the sides of the quadrangle makes the simplified point-in-polygon procedure failure-prone. The fringe case is illustrated in Figure 92. Despite the lack of robustness, it successfully flagged most elements with the fiber material ID and the local NURBS orientation. The fringe case occurs mostly when the NURBS quadrangles are formed near a turn. See Figure 93 as an example where a quadrangulated curve makes weaving turns. The quadrangles are mostly orthogonal except at the turns where they become trapezoidal. There, the low-cost point-in-polygon test from section 3.5.2 is not valid.



(a)



(b)

Figure 92: Fringe case for simplified point-in-polygon method

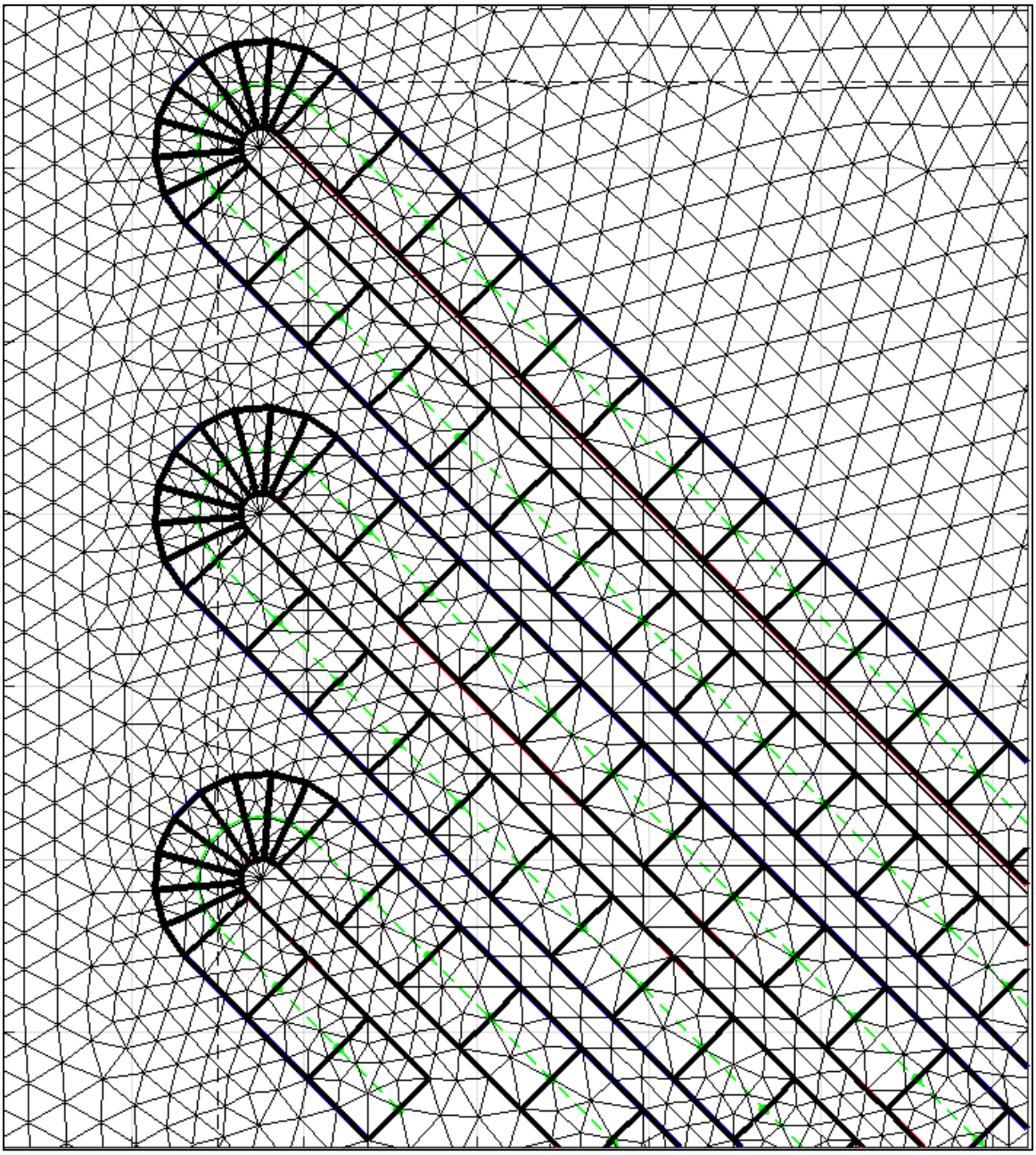


Figure 93: A quadrangulated, thickened, polyline.

4. THE PARALLEL FEA SCRIPT

Algorithm 2 produces a mesh with *auxiliary data*, namely the Euler angles and the material tags of the elements. This information must be used to construct analysis that will reveal the performance of the TO design after imposing manufacturability. The code used to do this is conceptually discussed mostly in terms of the mathematics that it is based on and the relevant PETSc abstractions.

4.1 A Primer on Continuum Mechanics

For analysis purposes, the CFRC components are modelled as a *continuum*, which is a space with no voids. To begin the analysis, one needs to enforce the relevant laws of physics inside the domain(Ω) and its boundary (Γ). Said laws are Newton's 2nd Law of motion (N2L), the law of conservation of momentum (Cauchy's 2nd Law of motion).

4.1.1 The Elasticity Equation

N2L states that the time rate of change of momentum equals the sum of the forces. In structural analysis, the two most common forces are the external tractions (T) and the body forces (B). If one denotes the mass with m and the velocity with v, then by N2L one writes:

$$T + B = \frac{\partial(mv)}{\partial t} \tag{60}$$

However, the finite element method in structural analysis is based on continuum mechanics, so the above statement must be rewritten in terms of tensors. This is done as:

$$B = \int_{\Omega} \rho f_i d\Omega \tag{61}$$

$$T = \int_{\Gamma} \sigma_{ij} \hat{n}_j d\Gamma \tag{62}$$

$$\frac{\partial(mv)}{\partial t} = \frac{\partial}{\partial t}\int_{\Omega} \rho v_i d\Omega \tag{63}$$

Where σ_{ij} is the rank-2 Cauchy stress tensor, ρ is the medium's density (a rank-0 tensor), f are the body forces (rank-1 tensor), and n is the outward unit normal to the boundary Γ. Using the divergence theorem, one can rewrite the traction force which is currently written as a boundary integral as a domain integral instead:

$$\int_{\Gamma} \sigma_{ij}\hat{n}_j d\Gamma = \int_{\Omega} \nabla \cdot \sigma_{ij} d\Omega \tag{64}$$

This allows for all terms in the original N2L statement to be integrals over the domain, allowing one to drop the integrals and write:

$$\nabla \cdot \sigma_{ij} + \rho f_i = \overline{\dot{\rho}\dot{v}} \tag{65}$$

This is the so-called elasticity equation, and it is a Partial Differential Equation (PDE) valued over a rank-1 tensor field, that of force. Typically, the body forces are known, and the linear momentum is prescribed as a set of boundary or initial conditions. However, no meaningful a priori knowledge of the stress tensor is available. Furthermore, because the equation involves a rank-2 tensor, the system is underdetermined. In 3D, this entails 9 unknowns but only 3 equations.

4.1.2 Constitutive Modelling

To close the system, the stress is assumed to be a function of one of the other known quantities. In structural analysis, this is done by making the stress tensor a function of the rank-1 displacement field (whose derivatives relate to the velocity tensor). This is done implicitly via the Generalized Hooke's Law:

$$\sigma_{ij} = C_{ijk\ell}E_{k\ell} \tag{66}$$

Where C_{ijkl} is the rank-4 stiffness tensor (linear), and E_{kl} is the rank-2 Lagrange strain tensor. The latter is a function of the displacement field:

$$E_{ij} = \frac{1}{2}(\mathrm{u_{i,j}} + \mathrm{u_{j,i}} + \mathrm{u_{i,k}u_{k,j}}) \tag{67}$$

This tensor is nonlinear and for simplicity is often linearized:

$$\varepsilon_{ij} = \frac{1}{2}(\mathrm{u_{i,j}} + \mathrm{u_{j,i}}) \tag{68}$$

Where ε_{kl} is the small strain tensor. Regardless, the stiffness tensor is assumed to be constant and is readily generated by using engineering constants peculiar to the material that composes the medium.

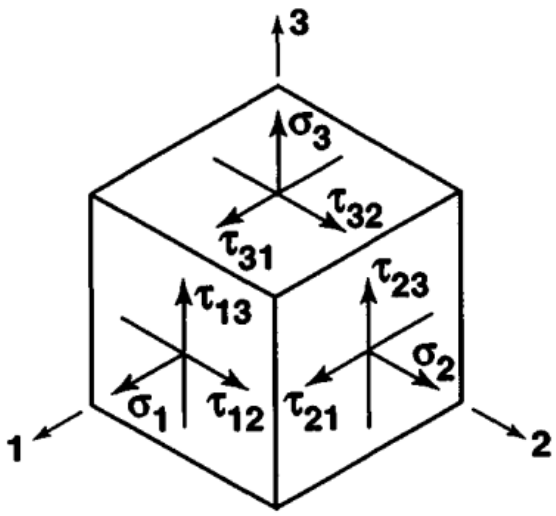


Figure 94: Typical illustration of the stress tensor. Taken from [2]

4.1.3 Symmetry of Stress, Strain, and Stiffness Tensors

Three more aspects of these equations are mentioned to wrap-up the discussion on physical modelling. Firstly, due to angular momentum considerations, the Cauchy stress tensor is

symmetric. Secondly, the definition of the Lagrange strain tensor makes it also symmetric (and that remains true for the small strain tensor). Due to these symmetries, the stiffness tensor is symmetric in two indices ($C_{ijkl} = C_{ijlk} = C_{jikl} = C_{jilk}$).

4.1.4 Beware the Infinitesimal Strain Assumption

Thirdly, the derivation of strain involves the so-called Eulerian and Lagrangian coordinate systems. All linear structural analysis assume that deformations are small, thus justifying taking both coordinate systems to be identical. The reader is warned that the equations presented thus far are "as is." In fact, there are multiple measures of strain and even nonlinear Hooke's laws. For more information on tensors and continuum mechanics, the reader is referred to the book by Thomas Mase, Ronald Smelser, and George Mase [29].

4.1.5 Compliance Tensor

The special case of C_{ijkl} from the generalized Hooke's Law is a constant tensor defined in terms of the so-called *engineering constants*. These are macroscopic parameters that characterize the mechanical response of a medium to static loading. They are catalogued in literature and measured in laboratory tests. The values of the stiffness tensor are not readily measurable though. The standard procedure to construct it is to first assemble its inverse, the *compliance* tensor from the engineering constants. Due to the two-fold symmetry of $C_{ijkl,}$ the compliance tensor is written as a 6x6 matrix in 3D:

$$S_{ijkl} = \begin{bmatrix} +\frac{1}{E_1} & -\frac{\nu_{21}}{E_2} & -\frac{\nu_{31}}{E_3} & +\frac{\eta_{23,1}}{G_{23}} & +\frac{\eta_{13,1}}{G_{13}} & +\frac{\eta_{12,1}}{G_{12}} \\ -\frac{\nu_{12}}{E_1} & +\frac{1}{E_2} & -\frac{\nu_{32}}{E_3} & +\frac{\eta_{23,2}}{G_{23}} & +\frac{\eta_{13,2}}{G_{13}} & +\frac{\eta_{12,2}}{G_{12}} \\ -\frac{\nu_{13}}{E_1} & -\frac{\nu_{23}}{E_2} & +\frac{1}{E_3} & +\frac{\eta_{23,3}}{G_{23}} & +\frac{\eta_{13,3}}{G_{13}} & +\frac{\eta_{12,3}}{G_{12}} \\ +\frac{\eta_{1,23}}{E_1} & +\frac{\eta_{2,23}}{E_2} & +\frac{\eta_{3,23}}{E_3} & +\frac{1}{G_{23}} & +\frac{\mu_{23,13}}{G_{13}} & +\frac{\mu_{23,12}}{G_{12}} \\ +\frac{\eta_{1,13}}{E_1} & +\frac{\eta_{2,13}}{E_2} & +\frac{\eta_{3,23}}{E_3} & +\frac{\mu_{13,23}}{G_{23}} & +\frac{1}{G_{13}} & +\frac{\mu_{13,12}}{G_{12}} \\ +\frac{\eta_{1,12}}{E_1} & +\frac{\eta_{2,12}}{E_2} & +\frac{\eta_{3,12}}{E_3} & +\frac{\mu_{12,23}}{G_{23}} & +\frac{\mu_{12,13}}{G_{13}} & +\frac{1}{G_{12}} \end{bmatrix} \tag{69}$$

Where the quantities E_i are the *Young's Moduli*, ν_{ij} are the *Poisson's ratios*, G_{ij} are the *Shear Moduli*, $\eta_{ij,k}$ and $\eta_{k,ji}$ are the *Lekhnitskii ratios*, and $\mu_{ij,kl}$ or $\mu_{kl,ij}$ are the *Chentsov ratios* [2]. These are the so-called engineering constants for the case of the linear elastic solid and must be fetched from a materials database for each material that is to be simulated. The engineering constants characterize certain behaviors in the elastic response for varying degrees of anisotropy in the linearly elastic medium (see Figure 95).

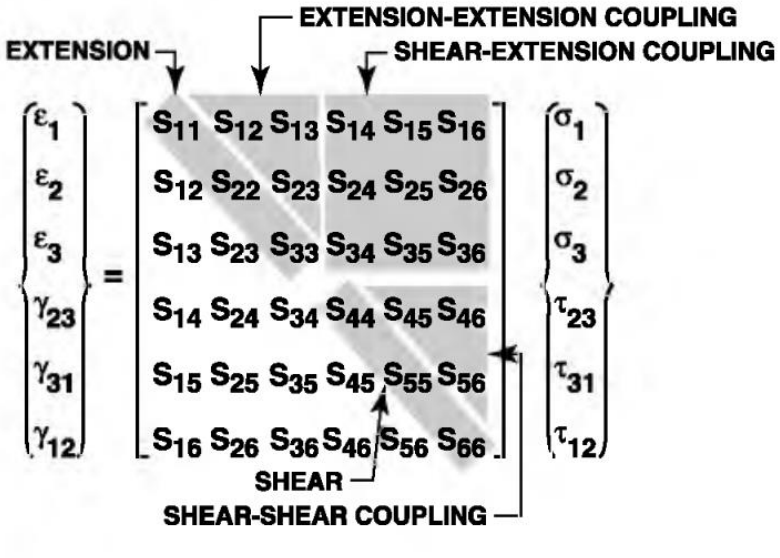


Figure 95: Compliance components and their effect on anisotropy. Taken from [2]

The linearly elastic solids can be classified according to their degree of anisotropy. Labels ranging from *isotropic*, *transversely orthotropic*, *orthotropic*, *monoclinic*, and *fully anisotropic* are used to refer to special cases of the compliance tensor S_{ijkl}, where the former most is least anisotropic and latter most is most anisotropic [29] [3]. In this work, only the *orthotropic* case is considered, which is characterized by $\mu_{kl,ij} = \mu_{ij,kl} = \eta_{ij,k} = \eta_{k,ji} = 0$, that is no shear-shear or shear-extension coupling is considered [2].

4.1.6 Plane Stress

The *compliance* tensor was introduced in its 3D but will be reduced to a meaningful 2D form as per the scope of this work. When reducing the dimensionality of a structural analysis from 3D to 2D, one of two customary approaches is taken, either that of *plane stress* or *plane strain* [30]. In this work the former is used, which entails $\sigma_3 = \tau_{13} = \tau_{23} = 0$. Thus, all stresses are contained within the 1-2 plane. This allows one to reduce the compliance tensor from a 6x6 matrix to a 3x3 one:

$$S_{ijkl} = \begin{bmatrix} +\frac{1}{E_1} & -\frac{\nu_{21}}{E_2} & +\frac{\eta_{12,1}}{G_{12}} \\ -\frac{\nu_{12}}{E_1} & +\frac{1}{E_2} & +\frac{\eta_{12,2}}{G_{12}} \\ +\frac{\eta_{1,12}}{E_1} & +\frac{\eta_{2,12}}{E_2} & +\frac{1}{G_{12}} \end{bmatrix} \tag{70}$$

In the orthotropic case, the stiffness tensor becomes (after setting the Lekhnitskii ratios to zero):

$$C_{ijk\ell} = \begin{bmatrix} \frac{E_1}{1-\nu_{12}\nu_{21}} & \frac{\nu_{12}E_2}{1-\nu_{12}\nu_{21}} & 0 \\ \frac{\nu_{12}E_2}{1-\nu_{12}\nu_{21}} & \frac{E_2}{1-\nu_{12}\nu_{21}} & 0 \\ 0 & 0 & G_{12} \end{bmatrix} = C \tag{71}$$

The engineering constants are measured in a reference coordinate system that is in general offset from the global axis used to enforce the elasticity equation. For this reason, it is necessary to write the equivalent stiffness tensor in the global frame. This is the underlying reason for requiring the elements of the mesh to get a local angle from the NURBS centerline in Algorithm 2. Euler Angles, such as the ones appended to the finite elements from the NURBS centerlines.

4.1.7 Tensor Coordinate Transforms

The stiffness tensor C_{ijkl} is of rank 4, so as per tensor coordinate transformation rules, it must be multiplied by four direction cosine tensors of rank 2 to change coordinate frames:

$$C_{pqrt} = Q_{ip}Q_{jq}Q_{kr}Q_{\ell t}C_{ijk\ell} \tag{72}$$

Raw conversion of a rank-4 stiffness tensor is cumbersome. A more "digestible" way of doing so is to use the Generalized Hooke's Law:

$$\sigma_{ij} = C_{ijk\ell}\varepsilon_{k\ell} \tag{73}$$

Transforming the rank -2 strain and stress tensors:

$$Q_{ix}Q_{jy}\sigma_{xy} = C_{ijk\ell}Q_{kx}Q_{\ell y}\varepsilon_{xy} \tag{74}$$

That is, the stress and strain tensors in some arbitrary i,j basis can be written in the global x,y counterpart. Rearranging Equation (86) one gets:

$$\begin{aligned} \sigma_{xy} &= Q_{ix}^{-1}Q_{jy}^{-1}C_{ijk\ell}Q_{kx}Q_{\ell y}\varepsilon_{x'y'} \\ \sigma_{xy} &= C_{xx'yy'}\varepsilon_{x'y'} \end{aligned} \tag{75}$$

The question now is, what are the direction cosine tensors Q_{ij}? For a rotation in 2D encoded by a single Euler angle, Q_{ij} can be written as a 2x2 matrix as:

$$Q_{ij} = \begin{bmatrix} +\mathrm{c}(\theta) & +\mathrm{s}(\theta) \\ -\mathrm{s}(\theta) & +\mathrm{c}(\theta) \end{bmatrix}, \qquad c(\theta) = \cos(\theta)\,, \qquad s(\theta) = \sin(\theta) \tag{76}$$

However, the tensor product of two direction cosines $Q_{ij}Q_{pq}$ is not the same as a matrix multiplication, rather, it must be produces via tensor enumeration. Doing so produces a rank-4 tensor which may be written as a 4x4 matrix [31]:

$$Q_{ij}Q_{pq} = \begin{bmatrix} +\mathrm{c}^2(\theta) & +c(\theta)s(\theta) & -c(\theta)s(\theta) & +\mathrm{c}^2(\theta) \\ +s(\theta)c(\theta) & +s^2(\theta) & -s^2(\theta) & +s(\theta)c(\theta) \\ -s(\theta)c(\theta) & -s^2(\theta) & +s^2(\theta) & -s(\theta)c(\theta) \\ +\mathrm{c}^2(\theta) & +c(\theta)s(\theta) & -s(\theta)c(\theta) & +\mathrm{c}^2(\theta) \end{bmatrix} \tag{77}$$

Because of the symmetries of the stiffness, stress, and strain tensor, a simplified version of $Q_{ij}Q_{pq}$ (which can partake in matrix multiplication) can be used [2] [31]:

$$Q_{ij}Q_{pq} = \begin{bmatrix} +\mathrm{c}^2(\theta) & +s^2(\theta) & -2\,\mathrm{s}(\theta)\,c(\theta) \\ +s^2(\theta) & +\mathrm{c}^2(\theta) & +2\,\mathrm{s}(\theta)\,c(\theta) \\ \mathrm{s}(\theta)\,c(\theta) & -\mathrm{s}(\theta)\,c(\theta) & \mathrm{c}^2(\theta) - \mathrm{s}^2(\theta) \end{bmatrix} = B(\theta) \tag{78}$$

Thus, the stress-strain relations in the global frame are written in matrix form as:

$$\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} = B(\theta) \begin{bmatrix} \dfrac{E_1}{1-\nu_{12}\nu_{21}} & \dfrac{\nu_{12}E_2}{1-\nu_{12}\nu_{21}} & 0 \\ \dfrac{\nu_{12}E_2}{1-\nu_{12}\nu_{21}} & \dfrac{E_2}{1-\nu_{12}\nu_{21}} & 0 \\ 0 & 0 & G_{12} \end{bmatrix} RB^{-1}(\theta)R^{-1} \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{bmatrix} \tag{79}$$

Where R is named *Reuter's* matrix and is needed to account for the often-given *engineering strain* which is different from the *tensor* strain ($\gamma_{xy} = 2\varepsilon_{xy}$) [2].

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \qquad R^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \tag{80}$$

4.2 A Primer on the Finite Element Method (FEM)

The FEM is a special case of a grander family of numeric schemes called *variational methods*. In formulating the FEM, a standard variational procedure is followed whereby the so-

called *weak* form of the model PDE is found and the field variables that one is solving for are written in terms of interpolation schemes. The interpolation schemes are deployed over a spatial discretization of the domain (such as the mesh shown in Figure 75), such that a minimum *continuity* requirement is satisfied. This entire process culminates in the construction of a linear system of equations that returns the solution field over the discretized domain [30].

4.2.1 Interpolation

The staple of the FEM is to write the field variables as weighted sums of interpolating polynomials where the weighting factors are observations of the field variable itself:

$$u_i = \sum_{j=1}^{n} u_j^h \phi_j \tag{81}$$

Where u is some field variable (i.e., the displacement field), u^h is some known value of the field variable at some discrete location, and ϕ_i is an interpolating function (almost exclusively a polynomial basis). The polynomial basis must be an interpolator, meaning that it must satisfy the so-called *partition of unity* [19]:

$$\sum_{i=1}^{n} \phi_i = 1 \tag{82}$$

Furthermore, the interpolating polynomial basis must satisfy certain differentiability requirements that are set by the *weak* form. If the *weak* form is a first order PDE, then the interpolation basis must have at least one nonzero derivative. If the *weak* form is a second order PDE, then the interpolation basis must have at least two nonzero derivatives [30].

4.2.2 Weak Form of the Elasticity Equation

The so-called *weak* form is an equivalent equation to the so-called *strong* form of Equation (65), as its solution field also satisfies the boundary conditions of the latter. The benefit of working with the *weak* form (and the reason it is called *weak*) is that it is of lower order of differentiability. To derive the weak form, it is standard to multiply both sides of any strong form (for any physical modelling) by some *test* tensor (w) and integrating over the domain:

$$\int_{\Omega} \nabla_{\mathrm{j}} \cdot (\sigma_{ij}) w_i d\Omega \; + \int_{\Omega} \rho f_i w_i d\Omega = \int_{\Omega} (\dot{\overline{\rho v}})_i w_i \; d\Omega \tag{83}$$

The test tensor is some arbitrary field whose only constraint is to vanish at the boundary Γ. It is otherwise arbitrary. Using integration by parts, the stress term can be rewritten as:

$$\left[w_i \sigma_{ij}\right]_{\Gamma} - \int_{\Omega} \sigma_{ij} w_{i,j} d\Omega \; + \int_{\Omega} \rho f_i w_i d\Omega = \int_{\Omega} (\dot{\overline{\rho v}})_i w_i \; d\Omega \tag{84}$$

Using the constitutive modelling, the physical laws, and the small strain tensor, the elasticity equation is rewritten as:

$$[w_i T]_{\Gamma} - \frac{1}{2} \int_{\Omega} C_{ijkl} \varepsilon_{ij} w_{k,\ell} d\Omega \; + \int_{\Omega} \rho f_i w_i d\Omega = \int_{\Omega} (\dot{\overline{\rho v}})_i w_i \; d\Omega \tag{85}$$

Where in the boundary term, the stress tensor is replaced by some known traction applied on the boundary. The question now is, what is the test tensor w? The answer is that the FEM user gets to decide. Of the many choices the most widespread is*:*

$$w_i = \sum_{j=1}^{n} w_j^h \phi_j \tag{86}$$

4.2.3 Spatial Discretization and Polynomial Basis

There is now a question about the space over which the interpolation takes place. In this work the domain is two-dimensional, and the spatial discretization consists exclusively of three-node triangles. Also, so far no intuition about the polynomial basis has been provided. This is because the interpolation scheme depends on the shape of the spatial discretization. For three-node triangles the so-called barycentric coordinates are used as the basis functions:

$$\phi_i = \begin{bmatrix}\phi_1\\ \phi_2\\ \phi_3\end{bmatrix} = \begin{bmatrix}L_1\\ L_2\\ L_3\end{bmatrix} = \begin{bmatrix}L_1\\ L_2\\ 1-L_1-L_2\end{bmatrix} = \begin{bmatrix}\xi\\ \eta\\ 1-\xi-\eta\end{bmatrix} \tag{87}$$

Where L_1, L_2, L_3 are the opposing areas that correspond to nodes 1 through three in a triangle. Evidently, the basis ϕ_i for the 3-node triangle is linear (as the *weak* form of the elasticity equation is a first order PDE) and adds up to 1. The reader is advised that other spatial discretizations exist for 2D such as quadrilaterals, or in 3D, tetrahedrons, hexahedrons, and triangular prisms. The choice of discretization changes the form of the ϕ_i.

4.2.4 Interpolation Over Three-Node Triangles

The interpolation scheme must be reconciled with the weak form by rewriting of the small strain tensor ε_{ij} in terms of the interpolating polynomial basis:

$$\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{bmatrix} = \begin{bmatrix}\dfrac{\partial u_x}{\partial x}\\ \dfrac{\partial u_y}{\partial y}\\ \dfrac{1}{2}\left(\dfrac{\partial u_x}{\partial y}+\dfrac{\partial u_y}{\partial x}\right)\end{bmatrix} \approx \begin{bmatrix}\displaystyle\sum_{i=1}^{N}\frac{d\phi_{\mathrm{i}}}{dx}u_x\\ \displaystyle\sum_{i=1}^{N}\frac{d\phi_{\mathrm{i}}}{dy}u_y\\ \dfrac{1}{2}\displaystyle\sum_{i=1}^{N}\left(\frac{d\phi_{\mathrm{i}}}{dy}u_x+\frac{d\phi_{\mathrm{i}}}{dx}v_y\right)\end{bmatrix} \tag{88}$$

This can be written in matrix form for 3-node triangles as:

$$\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{bmatrix} = Du = \begin{bmatrix} \frac{d\phi_1}{dx} & 0 & \frac{d\phi_2}{dx} & 0 & \frac{d\phi_3}{dx} & 0 \\ 0 & \frac{d\phi_1}{dy} & 0 & \frac{d\phi_2}{dy} & 0 & \frac{d\phi_3}{dy} \\ \frac{1}{2}\frac{d\phi_1}{dy} & \frac{1}{2}\frac{d\phi_1}{dx} & \frac{1}{2}\frac{d\phi_2}{dy} & \frac{1}{2}\frac{d\phi_2}{dx} & \frac{1}{2}\frac{d\phi_3}{dy} & \frac{1}{2}\frac{d\phi_3}{dx} \end{bmatrix} \begin{bmatrix} u_x^1 \\ u_y^1 \\ u_x^2 \\ u_y^2 \\ u_x^3 \\ u_y^3 \end{bmatrix} \tag{89}$$

Where the tensor index $i = 1,2$ has been renumbered as $i = x, y$. This expression is substituted into Equation (85) for ε_{ij} and its transpose for $w_{i,j}$.

4.2.5 Integration of the Weighted Integral

Equation (85) is a discretized version of the weak form of the elasticity equation written as weighted integrals. The integrals themselves must be evaluated, however, there is a problem: The basis polynomials are written in a coordinate system (the barycentric one) that is not the global *xy*. A transformation from the barycentric system to the *xy* is needed. The derivatives of the shape functions *w.r.t* to *x* and *y* can be written in terms of L_1 and L_2 via the chain rule:

$$\nabla_x \phi = \begin{bmatrix} \frac{d\phi_i}{dx} \\ \frac{d\phi_i}{dy} \end{bmatrix} = \begin{bmatrix} \frac{d\phi_i}{dL_1}\frac{dL_1}{dx} + \frac{d\phi_i}{dL_2}\frac{dL_2}{dx} \\ \frac{d\phi_i}{dL_1}\frac{dL_1}{dy} + \frac{d\phi_i}{dL_2}\frac{dL_2}{dy} \end{bmatrix}, \qquad i = 1,2,3, \tag{90}$$

Derivatives of ϕ_i *w.r.t* to L_1 and L_2 are readily available from Equation (87) for 3-node triangles:

$$\begin{aligned} \frac{d\phi_1}{dL_1} &= \frac{d\phi_2}{dL_2} = 1 \\ \frac{d\phi_1}{dL_2} &= \frac{d\phi_2}{dL_1} = 0 \\ \frac{d\phi_3}{dL_1} &= \frac{d\phi_3}{dL_2} = -1 \end{aligned} \tag{91}$$

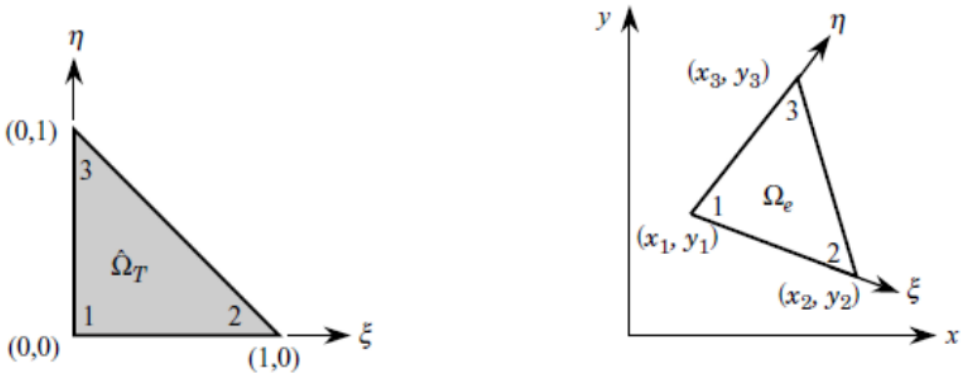


Figure 96: Mapping from reference geometry to arbitrary one. Taken from [30]

Because the basis polynomials partition unity, they and the triangle's nodes can be used to form a linearly independent basis for that spans the 2D space:

$$x_i = \begin{bmatrix} x \\ y \end{bmatrix} = \sum_{j=1}^{3} \begin{bmatrix} x_j^e \phi_j^e \\ y_j^e \phi_j^e \end{bmatrix} \tag{92}$$

The derivatives of the barycentric coordinates *w.r.t* to *x* and *y* (dL_1/dx, dL_1/dy, dL_2/dx, and dL_2/dy) can be found by taking derivatives *w.r.t* L_1 and L_2 from *x* and *y* from the above expression:

$$x_{i,j} = \nabla \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{dx}{dL_1} & \frac{dy}{dL_1} \\ \frac{dx}{dL_2} & \frac{dy}{dL_2} \end{bmatrix} = \sum_{j=1}^{3} \begin{bmatrix} x_j^e \frac{d\phi_j^e}{dL_1} & y_j^e \frac{d\phi_j^e}{dL_1} \\ x_j^e \frac{d\phi_j^e}{dL_2} & y_j^e \frac{d\phi_j^e}{dL_2} \end{bmatrix} = J_e \tag{93}$$

Where the matrix *J* is called the *Jacobian*, which is a linear map from the barycentric coordinate system to the *x*-*y* one. Inversion of this system yields the derivatives needed to assemble the rectangular matrix needed for the strains.

$$J_e^{-1} = \begin{bmatrix} \frac{dL_1}{dx} & \frac{dL_2}{dx} \\ \frac{dL_1}{dy} & \frac{dL_2}{dy} \end{bmatrix} \tag{94}$$

4.3 PETSc

With the lengthy theoretical discussion now concluded, it is time to discuss how PETSc is used to implement ALL the math discussed so far in this last, third chapter. The acronym PETSc stands for "Portable Extensible Toolkit for Scientific computing." It is an Automatic Program Interface (API) written for the C and Fortran programming languages by the PETSc development team based of the Argonne National Laboratory. As of writing, PETSc is on version 3.18 and remains under active development with the most current version available for public inspection in GitLab®. PETSc is open-source software released under the 3-clause BSD license [32].

The objective of PETSc is to provide abstractions for mathematical objects used in all field of Science, Technology, Engineering, and Mathematics (STEM). Examples of such abstractions include (but are not limited to) vectors, matrices, linear and nonlinear solvers, finite elements, meshes, graph representations and partitioners, and optimization solvers, etc. Therefore, PETSc is a scientific computing library. However, its much grander selling point is that *parallelism* is seamlessly built into the abstractions through the Message Passing Interface (MPI). That is, PETSc is meant to become an all-encompassing high-performance computing library to further STEM. See FIGURE to gauge the ambition.

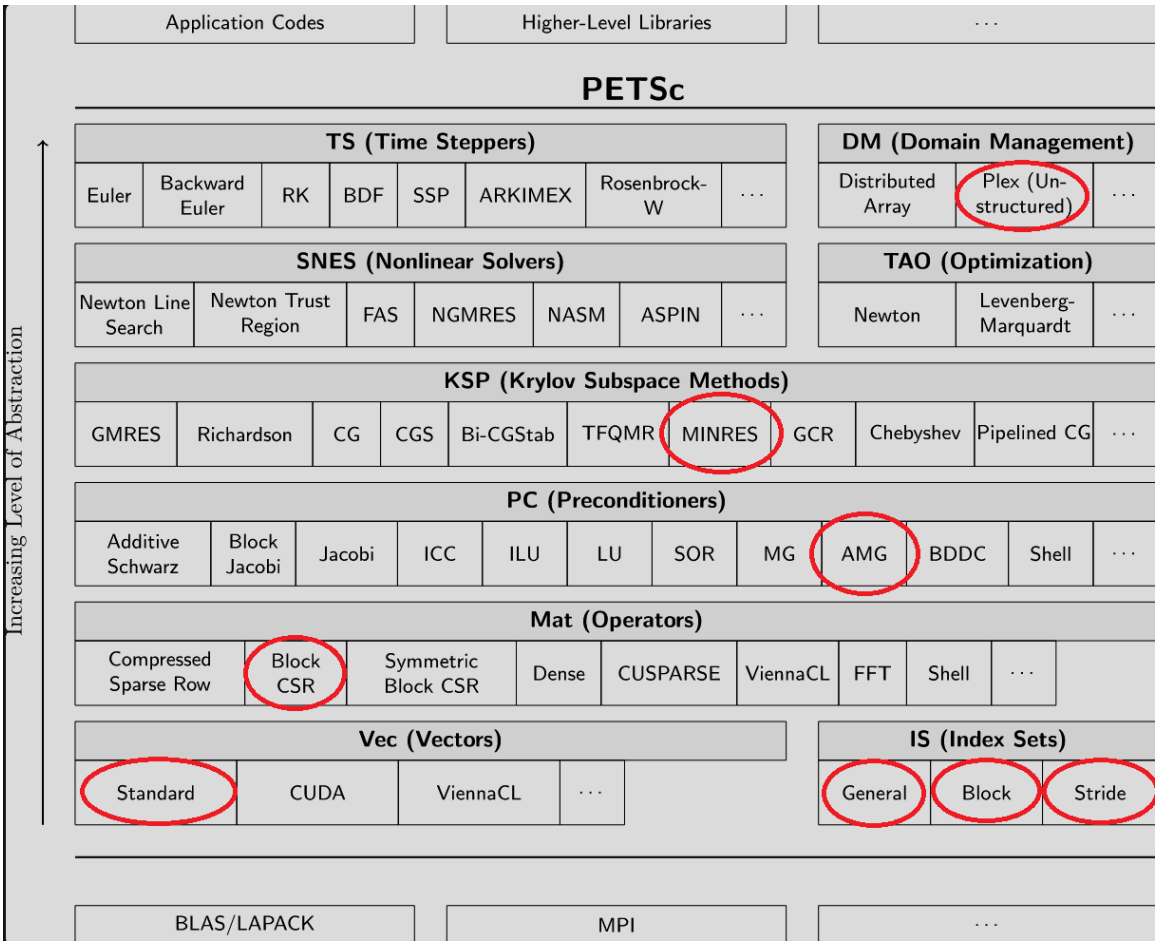


Figure 97: PETSc's arsenal of abstractions. Taken from [32]

Of the abstractions shown above, the ones circled in red are the ones used in this work. Describing them in detail is beyond the scope of this work as that falls in the realms of computational science applied to numeric linear algebra and general computer programming. One abstraction and one of its specializations though, the Domain Management (DM) in "Plex" mode is worthy of discussion as it pertains to the parallel assembly of the global finite element stiffness matrix. This is discussed next.

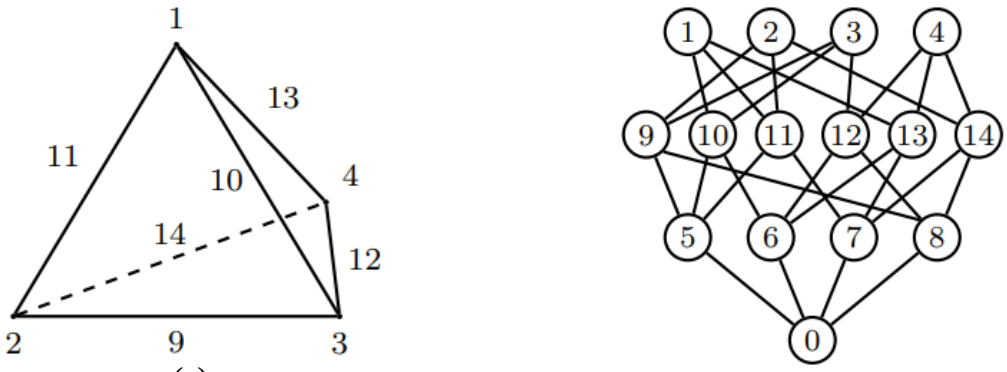
4.3.1 DM-Plex

DM is an abstraction meant to manage spatial discretizations (i.e., meshes). Meshes have a two-fold classification: *structured* vs. *unstructured*. In the former, spatial elements have an

implied, highly regular ordering scheme and typically consist of quadrilaterals in 2D or hexahedra in 3D. *Unstructured* meshes on the other hand, have a completely irregular order as is the case in the mesh shown in Figure 75.

DM-Plex is used to encode the order of the unstructured mesh in the form of a *graph*, which is a collection of *interrelated nodes*. The *nodes* themselves are a mathematical abstraction meant to represent anything. Relations between the nodes are denoted by arrows. DM-Plex implements a very specific graph, one that is *directed* and *acyclic*, meaning that the arrows cannot be traveled backwards, and that it is impossible to traverse nodes from some arbitrary starting node according to the arrows and end up back at the same node.

DM-Plex organizes its *Directed, Acyclic Graph* (DAG) into either three (3) levels for 2D domains or four (4) levels for 3D domains. One level corresponds to nodes that represent an element's vertices (such as the three corners in the triangular finite elements). Another level corresponds to the edges that make up element. Yet another level is reserved for surfaces in the 3D case. The last possible level is reserved for the elements themselves. See Figure 98 for an example



(a) (b)

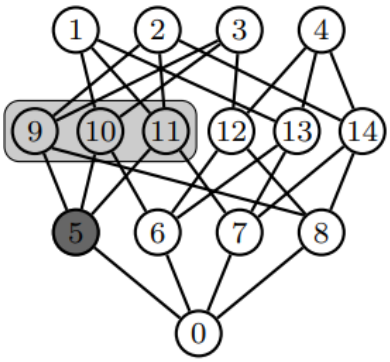
Figure 98: A tetrahedron (a) and its DAG (b) in DM-Plex. Taken from [33]

The relationship between all constituents of a spatial discretization is thus encoded compactly in a DAG. DM-Plex offers its users query routines that return the *nodes* to which any one particular *node* relates to.

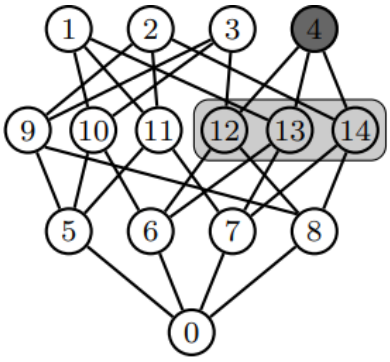
4.3.2 DAG Query Routines

There are two types of query routines which the layman can think of as “explicit” and “implicit.” Those are (in PETSc terminology) the *cone*, the *support*, the *star*, and the *closure*. The *cone* of a node is the set of all its connected points one level above. The *support* of a node is the set of all its connected points one level below. The *star* of a node is the set of all nodes across all levels below it that recursively or implicitly trace back to the node. Finally, the *closure* of a node is the set of all nodes across all levels above that recursively or implicitly relate back to the node. See Figure 99 for details

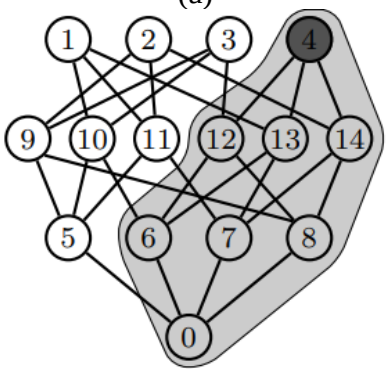
These query operations are critical in the assembly of the global stiffness matrix. Elemental stiffness matrices rely heavily on the *closure* of a node that represents an element. This is because the spatial coordinates of the nodes are required to assemble the *Jacobian* tensors Equation (94). The *support* operation is critical for the imposition of boundary conditions because those are applied on the boundary edges for 2D. An integration scheme over the edges is carried out to obtain nodal forces that must be distributed in the linear system over the rows that correspond to the affected nodes.



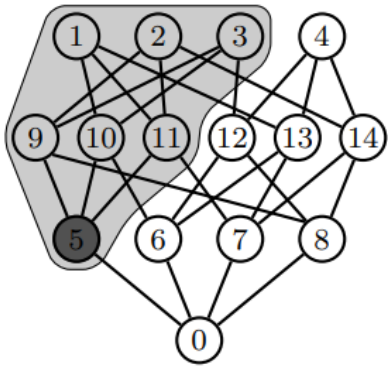
Cone of node 5
(a)



Support of node 4
(b)



Star of node 4
(c)



Closure of node 5
(d)

Figure 99: Illustration of DM-Plex's query routines. Adapted from [33]

In this work, the *cone* operation is used as part of a memory allocation scheme. The *star* operation is not explicitly used in this work; however, it is critical in the back end of the graph partitioning step.

4.3.3 Considerations of MPI Parallelism

The unstructured mesh is thus represented in DM-Plex via the DAG. However, there remains a question of how to make the overall FEA process faster. This is the stage of this work were PETSc's MPI *parallelism* is leveraged. In FEA, there are three critical stages that can be partitioned across computer processors to speed up the task: The assembly of the stiffness matrix, to iterative inversion to the global stiffness matrix, and the post-processing [33].

By far, the most expensive of the three is the assembly of the global stiffness matrix. Addition of MPI processes leads to linear decreases in the time required to assemble the matrix. This, however, is at the expense of having to partition the memory for the global stiffness matrix and right-hand-side vector. This is illustrated in Figure 100, were the green and blue chunks are memory that is local to a process, but the blue chunks need to be communicated to other processes. The memory partitions are visible to their respective MPI processes, so additional communication must take place at the solver stage [34].

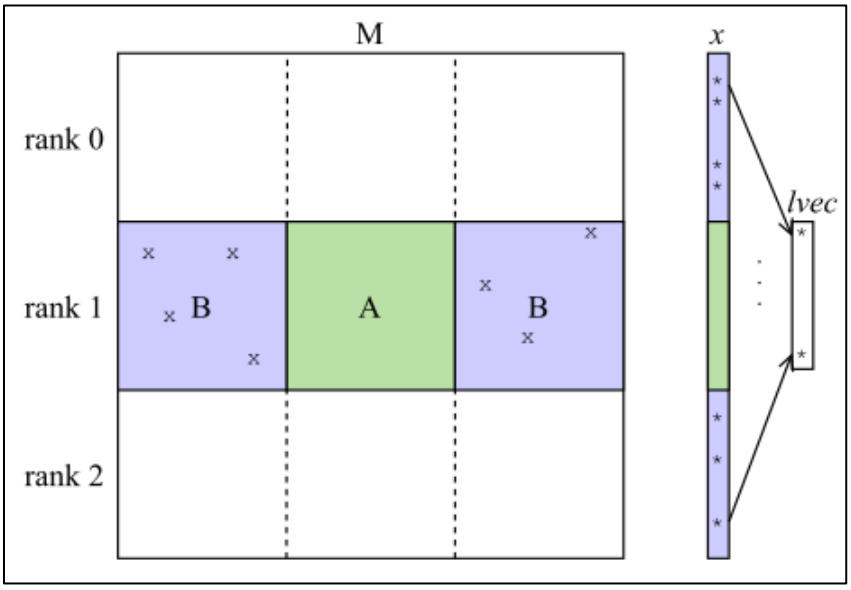


Figure 100: Illustration of PETSc's parallel partition of matrix. Taken from [34]

This inter-processor communication was not needed in the *serial* case and consequently slows down the speed of the solver. However, the losses at the solver are offset by the gains in the assembly phase up to a point. Both the losses and gains at the solver and assembly phases respectively are increased by adding MPI processes [34]. Therefore, there is an optimal number of processors to be used for a given mesh [33].

4.3.4 Graph Partitioning

DM-Plex first collects the DAG and corresponding geometric information to a single MPI process in the Random-Access Memory (RAM). For additional processes to assist in the task of FEA, they must be able to see the contents of the DAG and mesh. MPI, however, does not allow for one process to access the memory allocated for another process. MPI, however, allows for processes to communicate information between them. To partition the FEA task, therefore, the original MPI process must communicate information about the mesh and DAG to other processes so that they can make copies [34].

In the interim, a parallel task in MPI has the prospect of increasing memory usage over the *serial* counterpart. A memory deallocation is in place to avoid unnecessary copying and thus make the *parallel* case consume almost as much memory as the *serial* one. This, however, is not truly possible because the mesh and DAG cannot be evenly distributed across all MPI processes in any respect. Therefore, copies of parts of the mesh and DAG, so-called *ghost* data is created and kept with the MPI processes so that together (with their short-sighted memory access) they can reconstruct the entire FEA problem [35].

The task of partitioning the mesh and DAG (particularly for unstructured meshes) is not straightforward. For this reason, *graph partitioners* have been developed to parallelize physical simulations [35]. PETSc's DM-Plex interfaces with graph partitioners such as PT-SCOTCH, ParMetis, and Chaco to enable the partition of the DAG over an arbitrary number of MPI processes [32]. To illustrate graph partitioning, consider the simple mesh and its DAG shown in Figure 101:

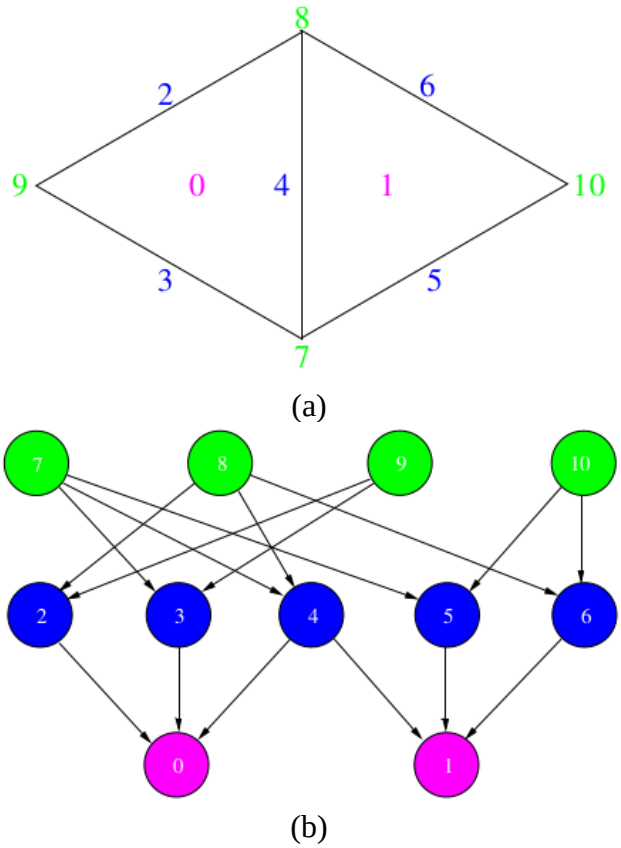


Figure 101: A simple 2D mesh and its DAG before partitioning. Taken from [35]

Together, the two three-node triangles form a simple mesh and DAG, which upon partitioning over two MPI processes could look like FIGURE. There, the two processes are labelled “0” and “1”. Vertices 8,7 on process 0 and vertices 3 and 2 on process 1 are copies of the same nodes respectively. Therefore, two of them are stored as *ghosts*. The same goes for process 0’s edge “4” and process 1’s edge “7.” A more pertinent example of graph partitioning is Figure 103.

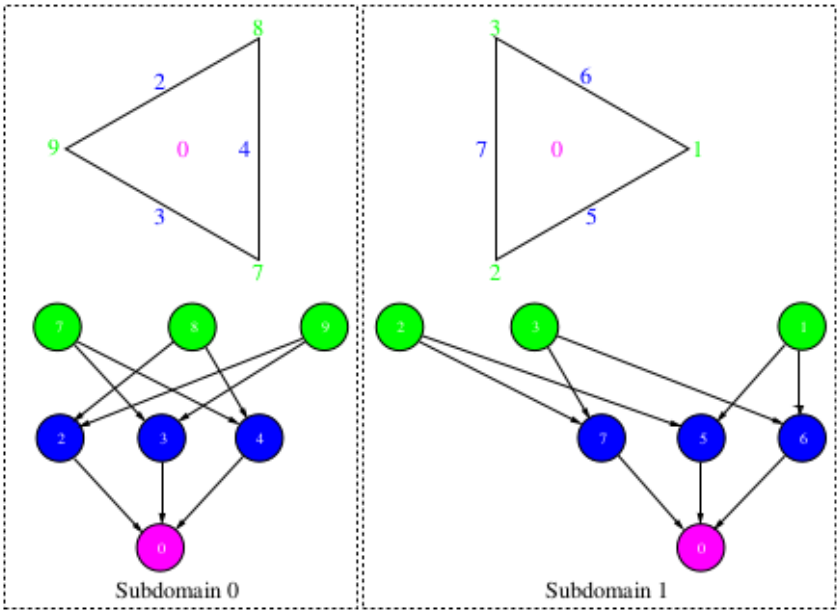


Figure 102: The partitioned version of the mesh and DAG in Figure 101. Taken from [35]

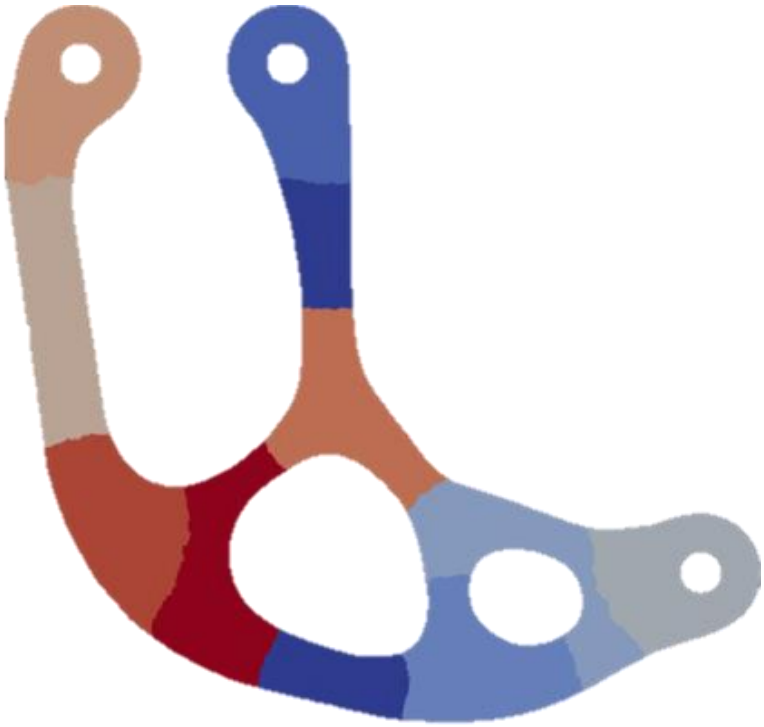


Figure 103: DM-Plex graph partitioning on the test mesh (10 MPI processes)

4.3.5 Characteristics of the PETSc Script

The discussion on DM-Plex is concluded. PETSc is a vast library and further discussion on its constituents deviates from the scope of this research excessively. For more information, the interested reader is referred to the PETSc website [32]. This subsection is concluded by highlighting some of the noteworthy inner workings of the script used in this work.

- Uses the PT-SCOTCH graph partitioner using a “balancing” strategy meaning that the partitioner attempts to give each process as close the same number of elements as possible.
- Computes the Jacobian and its inverse for all elements in each process.
    - For static FEA cases, these are not stored as they are needed only once.
    - In topology optimization, the same mesh is reused, so provisions to remember them are included for use in grander analysis schemes.
- Computes one stiffness matrix per material.
- Does not store the *transformed* stiffness matrices. Doing so would require the storage of a dense 3x3 matrix for each element.
    - Rebuilds them as needed inside an assembly loop.
- Computes the element stiffness matrices using PETSc’s specialized MatRARt() routine.
    - Uses PETSc’s Index Sets (IS) to insert them into the global stiffness matrix.
- Allocation of the global stiffness matrix is entrusted to PETSc’s DMCreateMatrix() routine which allocates the matrix from the DAG.
- Uses the KSPMINRES solver with the PCGAMG preconditioner.

4.3.6 Validation Against MSC NASTRAN

Chapter 4 concludes with sample results of a test load case (Figure 104) on the custom mesh shown in Algorithm 2 after appending the Euler angles. Results for displacement and stress fields are visualized with the open-source application ParaView®. The same case was run with in MSC NASTRAN. Several fields (displacement, strain, and stress) were evaluated and both software came within at most 2% of each other. These (except strain) are shown in Figure 105 through Figure 109. The results of this benchmark test are summarized in Table 15.

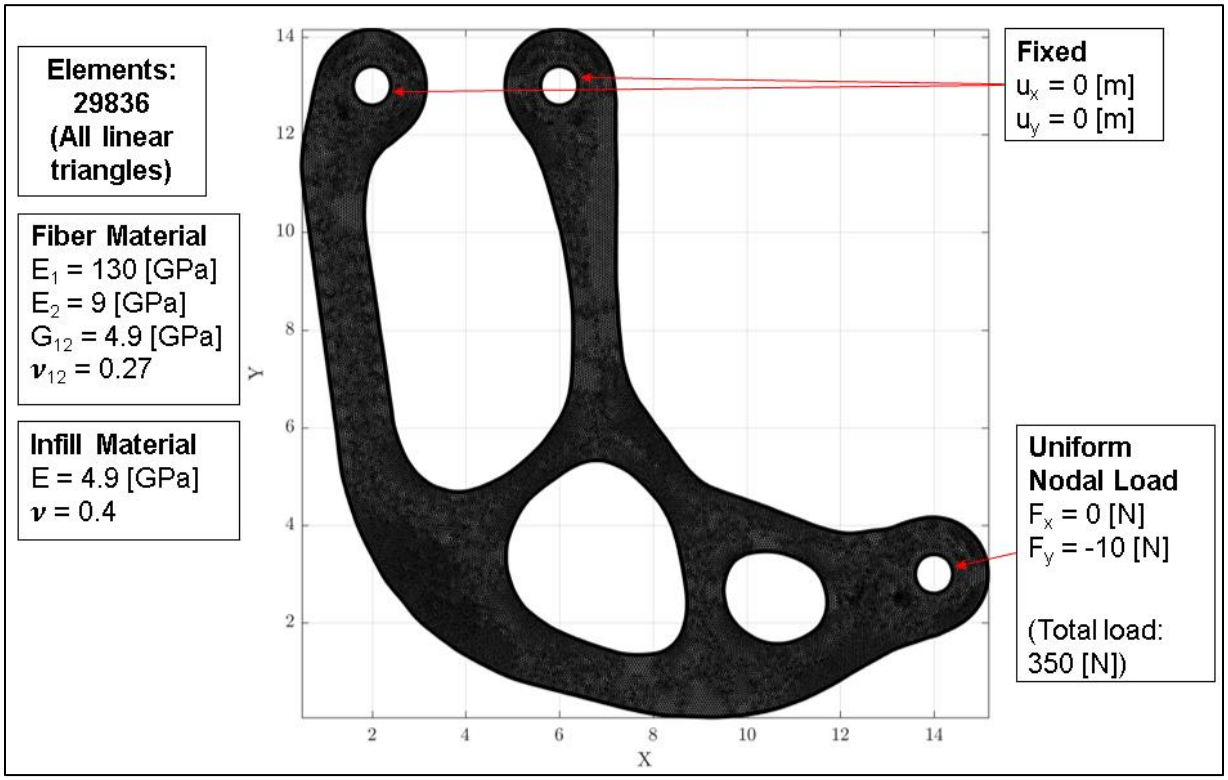


Figure 104: Load case and material properties

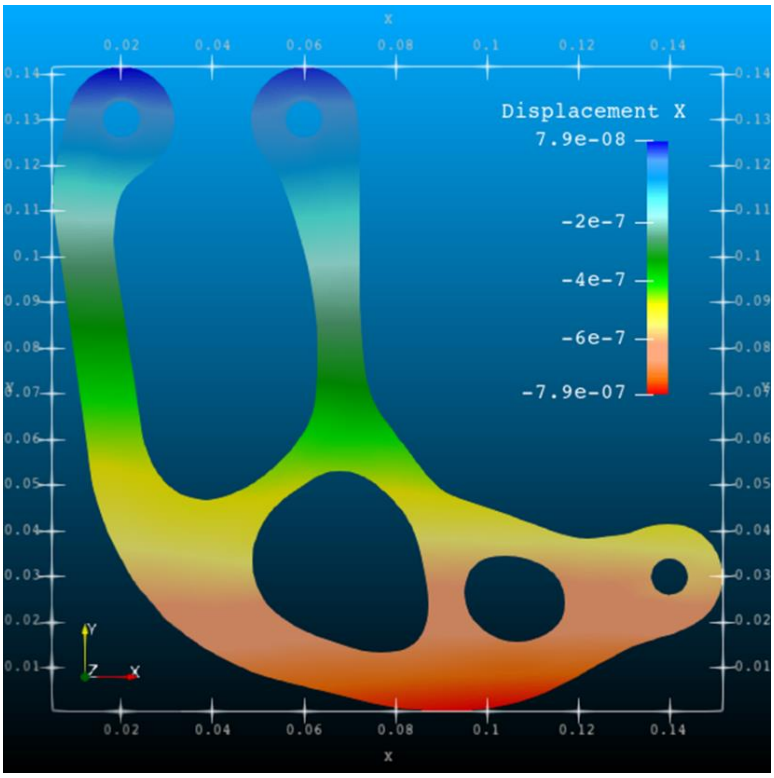


Figure 105: X-displacement (u_x) field (PETSc)

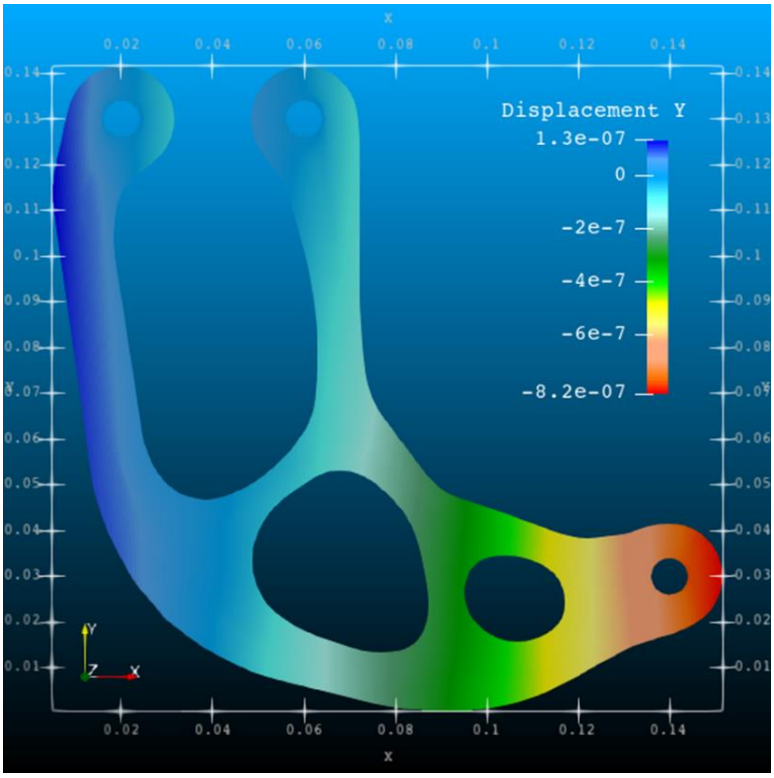


Figure 106: Y-displacement (u_y) field. (PETSc)

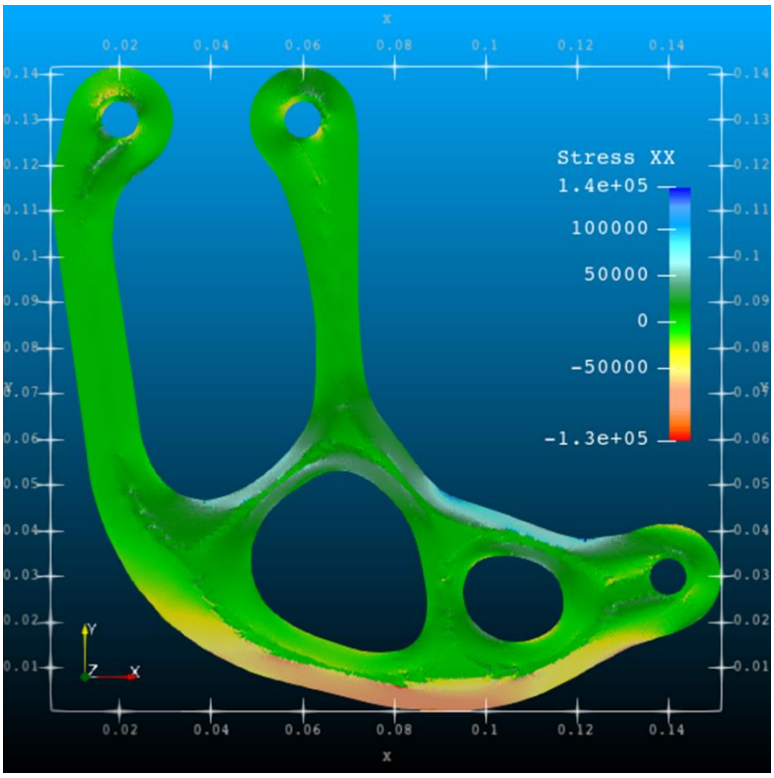


Figure 107: XX Stress tensor component (σ_{xx}) field. (PETSc)

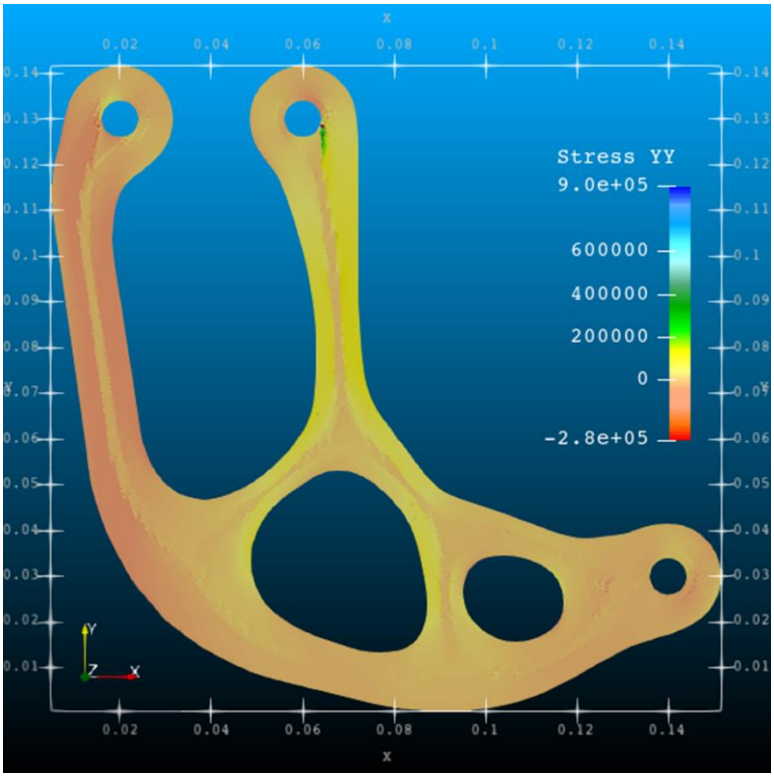


Figure 108: YY-Stress tensor component (σ_{yy}) field (PETSc)

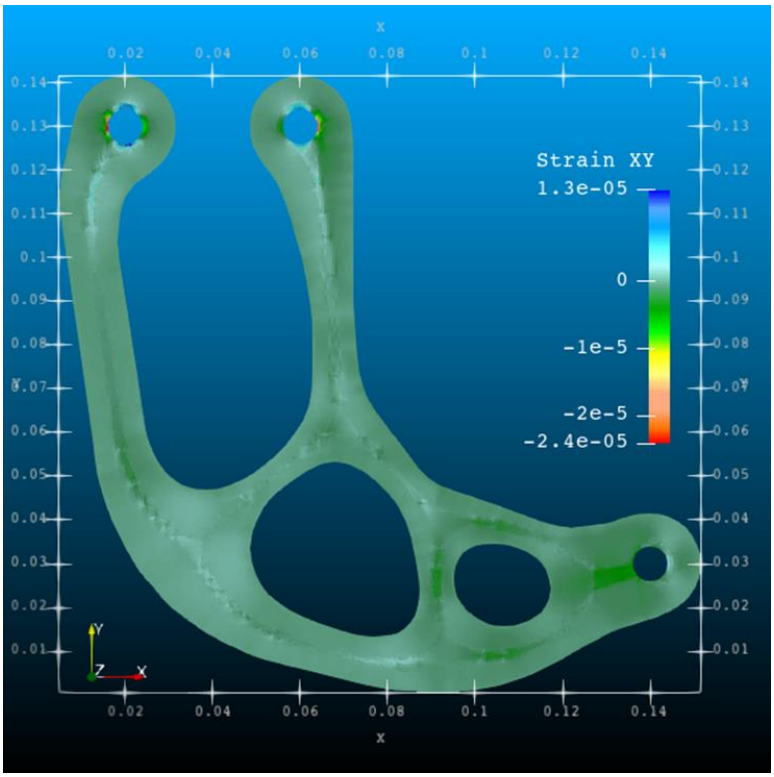


Figure 109: XY-Stress tensor component (σ_{xy}) field (PETSc)

Table 15: Load case results

Quantity	Symbol	Units	PETSc	MSC NASTRAN	Percent Difference
x-displacement	u_x	[m]	-7.90e-7 – +7.87e-8	-7.78e-7 – +7.72e-8	1.61%– 1.94%
y-displacement	u_y	[m]	-8.24e-7– +1.31e-7	-8.16e-7– +1.29e-7	1.00%– 1.81%
xx-strain tensor	ε_{xx}	N/A	-1.37e-5– +1.34e-5	-1.35e-5– +1.33e-5	1.20%– 1.36%
yy-strain tensor	ε_{yy}	N/A	-1.50e-5– +1.75e-5	-1.48e-5– +1.72e-5	1.38%– 1.50%
xy-strain tensor	ε_{xy}	N/A	-2.42e-5– +1.35e-5	-2.38e-5– +1.33e-5	1.51%– 1.50%
xx-stress tensor	σ_{xx}	[Pa]	-1.27e+5– +1.44e+5	-1.26e+5– +1.42e+5	0.96%– 1.49%
yy-stress tensor	σ_{yy}	[Pa]	-2.80e+5– +9.03e+5	-2.77e+5– +8.93e+5	0.95%– 1.15%
xy-stress tensor	σ_{xy}	[Pa]	-2.12e+5– +8.73e+4	-2.09e+5– +8.59e+4	1.21%– 1.63%

The results presented in Table 15 are given as "minimum/maximum" values. That is, the extremal values in the tensor fields are presented. The percentage differences were computed relative to the MSC NASTRAN values, that is:

$$\Delta = 100\% \left(\frac{z_{\text{MSC NASTRAN}} - z_{\text{PETSc}}}{z_{\text{MSC NASTRAN}}} \right) \tag{95}$$

Where z is any of the tensor field components listed in the second column in Table 15. The extremal values of the tensor fields presented so far are within 2% of each for both software, but a more comprehensive comparison is also showcasing the tensor fields output by MSC NASTRAN. This is done next to conclude chapter 3.

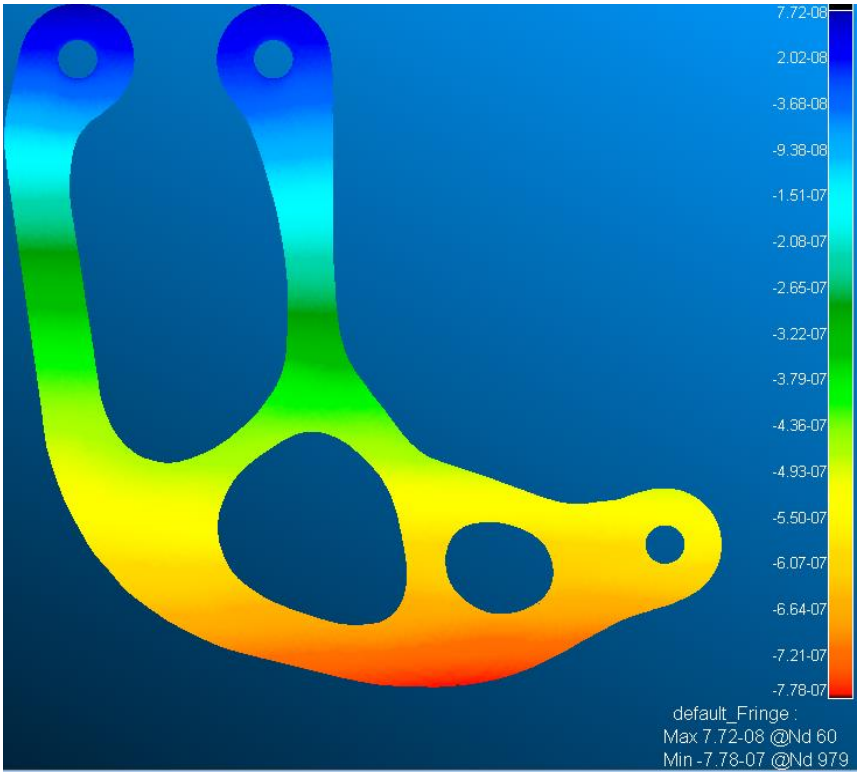


Figure 110: X-displacement (u_x) field (MSC NASTRAN)

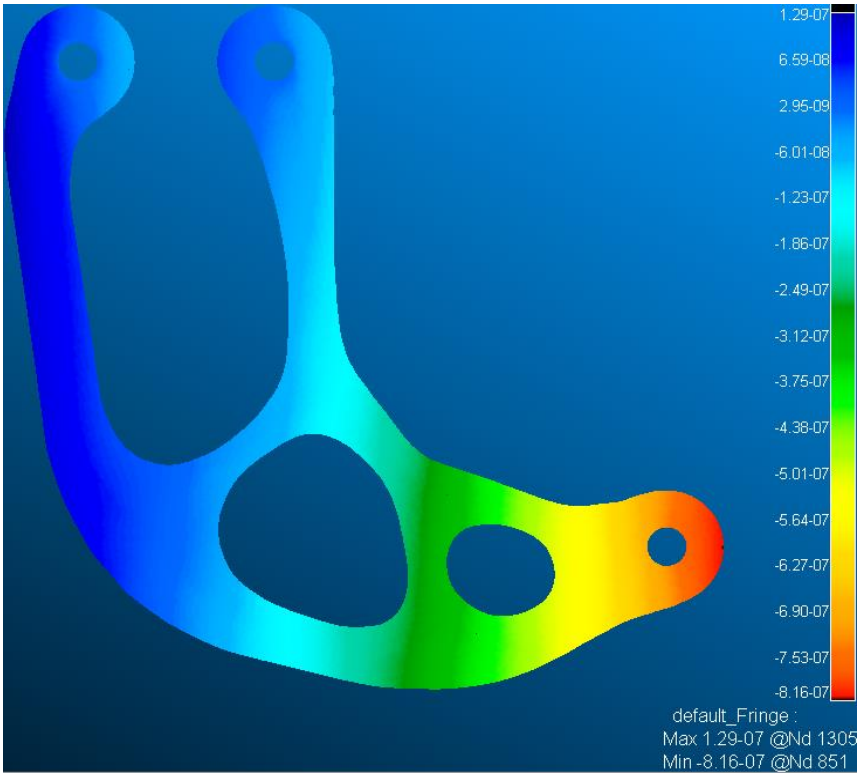


Figure 111: Y-displacement (ux) field (MSC NASTRAN)

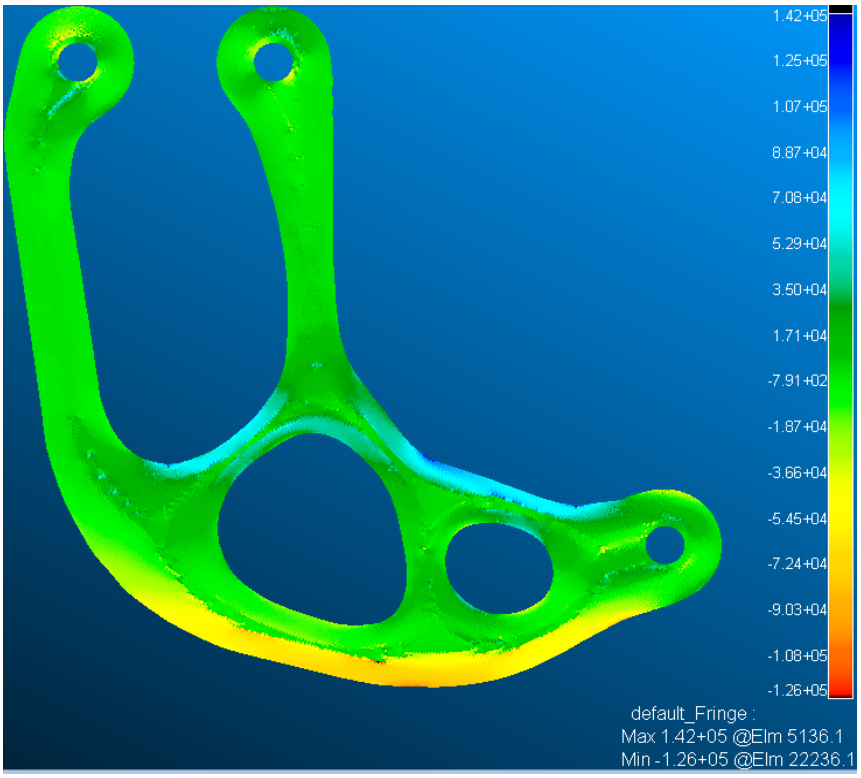


Figure 112: XX-Stress tensor component (σ_{xx}) field (MSC NASTRAN)

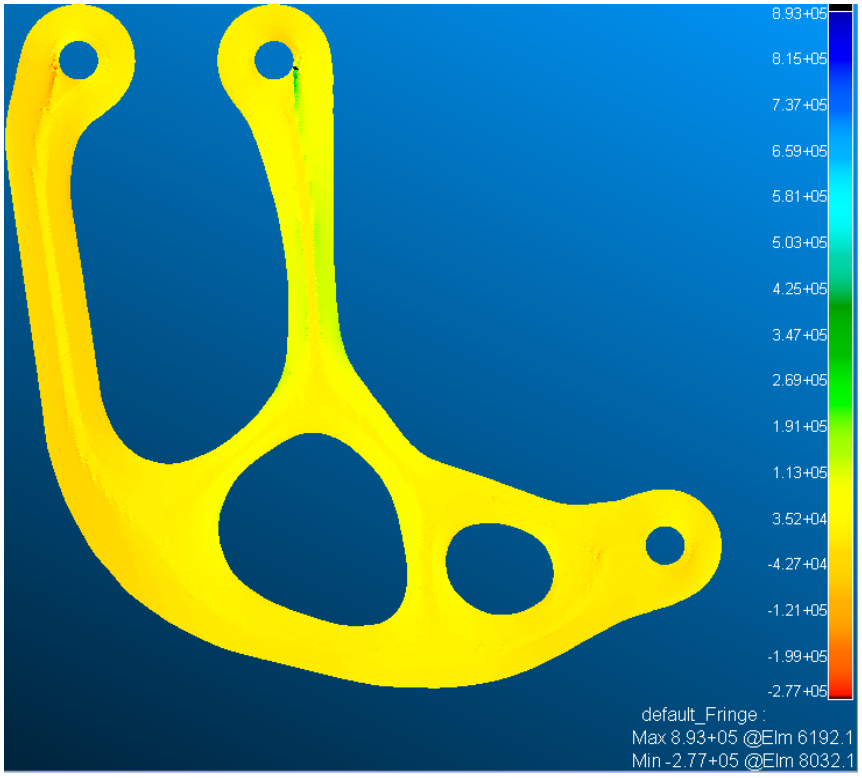


Figure 113: YY-Stress tensor component (σ_{yy}) field (MSC NASTRAN)

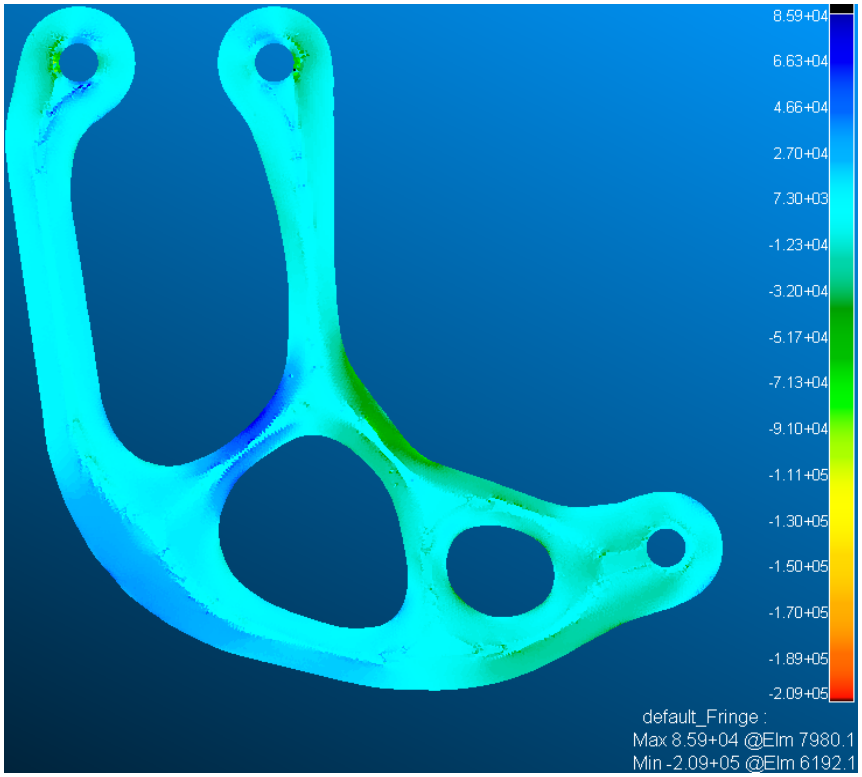


Figure 114: XY-Stress tensor component (σ_{xy}) field (MSC NASTRAN)

5. CONCLUDING REMARKS

In this work, two algorithms meant to guarantee the manufacturability of topologically optimized designs were developed and presented. Algorithm 1 enforces the manufacturability of the design solely from the TO boundary, desired fiber thickness, and turning tolerance of the printer. Algorithm 1 can best be described as a constrained space-filling algorithm based on the SDF. Algorithm 2 is a routine that generates input for FEA routines. It takes as input the NURBS generated by Algorithm 1, and the boundaries generated by TO to accurately assign *auxiliary* data to a mesh. Said data consists of Euler angles derived from the local NURBS orientation which are then assigned in bulk to the elements whose centroids are within the NURBS offset quadrangles. Finally, a parallel PETSc script that implements the Finite Element Method applied to linear elasticity as derived from continuum mechanics was showcased. This script is custom to take as input the product of Algorithm 2. The latter was used to generate FEA input files for both MSC NASTRAN and the PETSc script. Both software ran a load case and were within 2% of each other over various queried tensor fields.

5.1 Regarding Algorithm 1's SDF Level Sets

The SDF contours are essentially the backbone for this work. However, the performance of both algorithms has a substantial dependency on the quality of the interpolated SDF contours. Issues stemming from SDF coarseness were diagnosed early on by the "islands" shown in Figure 29, and Figure 31. These "islands" should not be confused with short fibers, which are otherwise well-defined contours produced from successful interpolation of the SDF. The existence of the "islands" is attributed to data coarseness. Other issues pertaining to data coarseness manifested during the early stages of Algorithm 2, when the NURBS curves were offset to produce the fibers. The most obvious case was shown in Figure 85 where some of the quadrangles were unnaturally

larger or smaller than their more uniform counterparts. This was due to irregularities in the SDF contour as it unevenly distributed the control points. Data coarseness should be mitigated by generating the SDF contours with a finer stencil. This, however, makes the SDF's generation more expensive. A future iteration of this work will not use the SDF at all and will instead offset the active contours directly. This will avoid the data coarseness problem but will require implementation of priority queues and polygon vs. polygon intersection detection and decomposition.

5.2 Regarding Algorithm 1's Laplacian Smoothing

The Laplacian smoothing technique was investigated as a potential fix to the data coarseness problem exhibited by the SDF contours. It was effective at eradicating the "wiggle" type noise which made the offset operation on polylines possible. Despite the myriad of problems associated with it (shrinkage, feature loss, warping, and numerical instability), Laplacian smoothing was a valuable tool. The proposed advancing front scheme in Section 4.1 should eliminate the need for Laplacian smoothing altogether. The Laplacian smoothing technique won't be immediately discarded though, as it can potentially be a competitor to the LEE as a blending technique if *localized* smoothing is used as opposed to *globalized* smoothing. The topic of Laplacian smoothing is too broad (it is an active field of research) and not enough time could be allocated to fully understand it for purposes of fiber placement. The use of the *Explicit Laplacian* scheme has one open-ended question: The choice for λ has to seemingly be arbitrarily low. In future work, other Laplacian schemes, such as the *Implicit Laplacian* scheme could be explored. Also, a method whereby the shrinkage and translation changes are undone should be explored.

5.3 Regarding Algorithm 1's Rollercoaster

The rollercoaster's initial iteration featured readily understandable flaws that were mostly addressed. The failure to fully flag a tight turn was effectively addressed by prescribing the "sink" distance. The flagging of regions of a concave curve was not really addressed but a corrective procedure was implemented to detect the false positives in the flag buffer. There is one element of open-endedness and that is the sink distance. The sink distance was prescribed heuristically as a scaling factor when offsetting the "rail tracks". During testing, a sink factor of 0.95 substantially improved results over the naïve base offset distance. That, however, was not perfect, as bands consisting of two -1's were still encountered. A more deterministic approach would attempt to leverage the curvature of the contour.

Curvature as a metric was neglected because it does not have a universally accepted definition for polylines. It can, however, be estimated according to simple schemes such as the one based on angle presented by [36], the one based on area by [37] or via the reciprocal of the radius of subsequent three-point circles among many other proposed schema. The point being, that the successor to the rollercoaster algorithm or its replacement should incorporate curvature as a metric to avoid the false flagging altogether and to flag the entirety of the tight turn more comprehensibly without needing such "jury-rig" measures as the sink distance or the dot product check.

5.4 Regarding Algorithm 2's Quadrangulation and Overlay

The quadrangulation scheme does its job, but there is a more efficient way to do it. The need for the overlay scheme stems for a technical limitation within MATLAB's PDE toolbox, and that is that the NURBS quadrangles overwhelm the toolbox's composite domain routine. There are simply too many! Even though quadrangles are a simple shape, Composition of a unified domain

via Boolean addition of many shapes that overlap and form voids between them is computationally nightmarish. During preliminary testing, a quadrangulated fiber was fed to MATLAB's PDE toolbox to extract a mesh that locally conformed to the fiber. The entirety of the overlay and search procedure was avoided, as the quadrangles were meshed as if they were a subdomain. This created a relational mechanism whereby the elements inside the quadrangle were readily queried and had their properties assigned immediately. There was no need for the sort-assisted search.

Because of the technical limitations with overly complicated union domains, the quadrangulation step coupled with the advancing fronts proposed in Section 4.1 should be augmented to produce a mesh concurrently. That is, the creation of a custom meshing routine for the placement of the fibers is in order. This must offset the active domains by the fiber thickness, quadrangulate, and mesh the quadrangles before proceeding with further offsets.

5.5 Regarding Algorithm 2's Point-In-Polygon Tests

The scheme must switch to one of the classical point-in-polygon schemes such as area-based, angle-based, or triangle- based, as the assumption that the NURBS quadrangles are orthogonal is not valid. This was elucidated with the so-called "fringe" case.

REFERENCES

[1] E. van de Ven, C. L. M. M. R. Ayas and F. van Keulen, "Accessibility of support structures in topology optimization for additive manufacturing.," *International Journal for Numerical Methods in Engineering,* vol. 122, no. 8, pp. 2038-2056, 2021.

[2] R. M. Jones, Mechanics of Composite Materials, 2nd ed., New York: Taylor & Francis Group, LLC, 1999.

[3] D. R. Askeland, P. P. Fulay and W. J. Wright, The Science and Engineering of Materials, 6th ed., H. Gowans, Ed., Cengage Learning, 2010.

[4] R. R. Fernandes, A. Y. Tamijani and M. Al-Haik, "Mechanical characterization of additively manufactured fiber-reinforced composites," *Aerospace Science and Technology*, vol. 113, 2021.

[5] A. Seifans, S. Ayyagari and M. Al-Haik, "Elastic/viscoplastic characterization of additively manufactured," *Aerospace Science and Technology,* vol. 111, 2021.

[6] M. Monti, M. Palenzona, F. Fiorino, F. Baudach and A. Onnis, "Design, manufacturing and FEA prediction of the mechanical behavior of a hybrid-molded polycarbonate / continuous fiber reinforced composite component," *Composites Part B,* vol. 238, p. 109891, 2022.

[7] N. van de Werken, J. Hurley, P. Khanbolouki, A. N. Sarvestani, A. Y. Tamijani and M. Tehrani, "Design Considerations and modeling of fiber reinforced 3D printed parts," *Composites Part B: Engineering,* vol. 160, pp. 684-692, 1 March 2019.

[8] H. Huang and R. Talreja, "Effects of void geometry on elastic properties of unidirectional fiber reinforced composites," *Composites Science and Technology,* vol. 65, no. 13, pp. 1964-1981, 2005.

[9] J. Jarvinen, V. Matilainen, X. P. H. S. A. Li, I. Makela and O. Nyrhila, "Characterization of Effect of Support Structures in Laser Additive Manufacturing of Stainless Steel,," *Physics Procedia,* vol. 56, pp. 72-81, 2014.

[10] M. P. Bendsoe and N. Kikuchi, "Generating Optimal Topologies in Structural Design Using A Homogenization Methods," *Computer Methods in Applied Mechanics and Engineering,* vol. 71, no. 2, pp. 197-224, 1988.

[11] A. M. Mirzendehdel and K. Suresh, "Support structure constrained topology optimization for additive manufacturing,," *Computer-Aided Design,* vol. 81, pp. 1-13, 2016.

[12] L. Alacoque, R. T. Watkins and A. Y. Tamijani, "Stress-based and robust topology optimization for thermoelastic multi-material periodic microsctructures.," *Computer Methods in Applied Mechanics and Engineering,* vol. 379, 2021.

[13] F. Fernandez, W. S. Compel, J. P. Lewicki and D. A. Tortorelli, "Optimal design of fiber reinforced composite structures and their," *Computer methods in applied mechanics and engineering,* vol. 353, pp. 277-307, 2019.

[14] Y. Zhou, W. Zhang, J. Zhu and Z. Xu, "Feature-driven topology optimization method with signed distance function," *Scie,* vol. 310, pp. 1-32, 2016.

[15] MSC. Software Corporation, MSC. Nastran, Santa Ana CA, 2004.

[16] F. Feito, J. Torres and A. Ureña, "Orientation, simplicity, and inclusion test for planar polygons," *Computers & Graphics,* vol. 19, no. 4, pp. 595-600, 1995.

[17] M. Wei, S. Wuyao, J. Qin, J. Wu and T.-T. Wong, "Feature-preserving optimization for noisy mesh using joint bilateral," *Optics and Lasers in Engineering,* vol. 51, no. 11, pp. 1223-1234, 2013.

[18] B. Barroqueiro, A. Adrade-Campos, Dias-de-Oliveira and R. Valente A.F., "Bridging Between Topology Optimization and Additive Manufacturing via Laplacian Smoothing," *Journal of Mechanical Design,* vol. 143, no. 9, pp. 143-151, 2021.

[19] J. D. Hoffman, Numerical Methods for Engineers and Scientists, 2nd ed., Basel, New York: Marcel Dekker Inc., 2001.

[20] W. Cheney and D. Kincaid, Numerical Mathematics and Computing, 6th ed., B. Willete, Ed., Bob Pirtle, 2008.

[21] T. Fukuchi, "Algorithm for deriving multidimensional space finite difference schemes using interpolation polynomials," January 2022. [Online]. Available: https://www.researchgate.net/publication/357511041_Algorithm_for_deriving_multidimensional_space_finite_difference_schemes_using_interpolation_polynomials?channel=doi&linkId=61d1590bd45006081683dc6f&showFulltext=true.

[22] T. Kim, J. Sewall, A. Sud and L. Ming C., "Fast Simulation of Laplacian Growth," *IEEE Computer Graphics and Applications,* vol. 27, no. 2, pp. 68-76, 2007.

[23] J. K. Seong, G. Elber and M. Kim, "Trimming local and global self-intersections in offset curves/surfaces using distance maps," *Computer-Aided Design,* vol. 38, no. 3, pp. 183-193, 2006.

[24] J. C. Fermiani, C.-Y. Chuang and A. Razdan, "Least eccentric ellipses for geometric Hermite interpolation," *Computer Aided Geometric Design,* vol. 29, no. 2, pp. 141-149, 2012.

[25] M. R. Spiegel, S. Lipschutz and J. Liu, Mathematical Handbook of Formulas and Tables, 4th ed., McGraw Hill, 2013.

[26] L. Piegl and W. Tiller, The NURBS Book, 2nd ed., Berlin: Springer, 1995.

[27] L. A. Piegl and W. Tiller, "Computing offsets of NURBS curves and surfaces," *Computer Aided Design,* vol. 31, no. 2, pp. 147-156, 1999.

[28] H. Chong-Wei and S. Tian-Yuan, "On the Complexity of Point-in-Polygon Algorithms," *Computers & Geosciences,* vol. 23, no. 1, pp. 109-118, 1997.

[29] T. G. Mase, R. E. Smelser and G. E. Mase, Continuum Mechanics for Engineers, 3rd ed., J. Reddy, Ed., Boca Raton, Florida: CRC Press, 2010.

[30] J. Reddy, An Introduction to the Finite Element Method, 3rd ed., New York City, New York: McGraw-Hill, 2006.

[31] S. W. Tsai and E. M. Wu, "A General Theory Of Strength for Anisotropic Materials," U.S. Air Force Materials Laboratory, 1972.

[32] S. Balay, S. Abhyankar, M. Adams, S. Benson, P. Brune, K. Buschelman and E. M. Constantinescu, "PETSc Web page," 2022. [Online]. Available: https://petsc.org/release/.

[33] M. Lange, L. Mitchell, M. G. Knepley and G. J. Gorman, "Efficient Mesh Management in Firedrake Using PETSC-DMPLEX," *SIAM Journal on Scientific Computing,* vol. 38, no. 5, pp. 143-155, 2016.

[34] J. Zhang, J. Brown, S. Balay, J. Faibussowitsch, M. Knepley, O. Marin, R. Trans Mills, T. Munson, B. F. Smith and S. Zampini, "The PetscSF Scalable Communication Layer," arXiv, 2021.

[35] M. G. Knepley and D. A. Karpeev, "Mesh Algorithms for PDE with Sieve I: Mesh Distribution," *Scientific Programming,* vol. 17, no. 3, pp. 215-230, 2009.

[36] J. Cufi, C. Rodriguez and A. Reventos, "Curvature for Polygons," *The American Mathematical Montly,* vol. 122, no. 4, pp. 332-337, 2015.

[37] K. Park, "Discrete Curvature based on Area," *Honam Mathematical*, vol. 32, pp. 53-60, 2010.