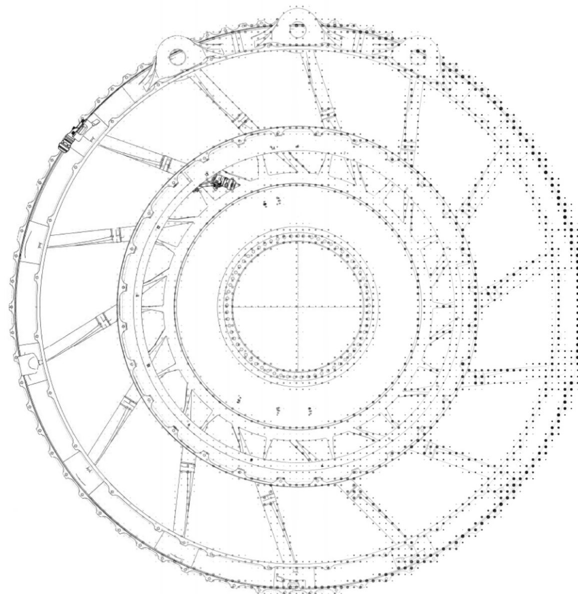
University of South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2021
Industrial IT and Automation IM

General Approaches for 3D Point-Cloud Evaluation, Classification and Material Thickness



Ruben Austefjord

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway
www.usn.no

Course: FMH606 Master's Thesis 2021
Title: *General Approaches for 3D Point-Cloud Evaluation, Classification and Material Thickness*
Pages: *58*
Keywords: *Point Clouds, Point Cloud Evaluation, Point Cloud Classification, and Material Thickness Evaluation.*

Author: *Ruben Austefjord*
Supervisors: *Håkon Viumdal, Ola Marius Lysaker*
External partners: *Håvard Norum GKN Aerospace*

Summary:
Analysis of geometric point cloud measurements pose very difficult challenges if removed from prior knowledge regarding the measurement process or measured object. The goal is to investigate how collections of 3D coordinates produced from 3D scan measurement systems can be structured, evaluated and analyzed solely from the information available in the data. The key difficulty is to create frameworks and methods able to be generally applicable to any unstructured point cloud.

The problems and dependencies regarding handling of unstructured geometric point clouds are presented alongside methods to resolve them. A method for utilizing the commercially available software ATOS Professional used by the aerospace manufacturer GKN Aerospace to find material thickness of welding junctions is discussed. A general method for determining material thickness for 3D scanned data with low curvature is proposed and implemented through Python programming.

The methods are continuously tested throughout the implementation of the presented material thickness method using fabricated 2D and 3D geometrical point cloud shapes with random noise to mimic real measurement variance. Large grounds for further work has been uncovered where parts of the presented methods have room for improvement as well as the discovery of other solutions and analytical methods.

Preface

This master's thesis is written as the individual finale of the Industrial Master study programme of Industrial IT and Automation at the Faculty of Technology, Natural Sciences and Maritime Sciences at the University of South-Eastern Norway, Porsgrunn. The work follows the requirement of the *FMH606 Master's Thesis* subject and is a continuation of the work presented in the preliminary *FM4017 Project* subject. It is throughout the thesis assumed that details about the TRF manufacturing process alongside it's hardware and software components are known from the previous work in *Adaptive Welding Automation* (AWA) by Austefjord, R. 2020 [1] and have thus not been explained twice.

The intent throughout the thesis is to infer useful insight and knowledge from information available within 3D point cloud datasets by utilizing studies on structuring, clustering and classifying algorithms. Ultimately presenting frameworks and methods for general solutions. The thesis has had a focus on method development and implementation, resulting in an equal priority on documenting the textual thesis as well as developing good code.

I would like to extend my gratitude towards my supervisors, *Håkon Viumdal*, *Ola Marius Lysaker* and *Håvard Norum* for their professional expertise, guidance and feedback aiding me through the presented work. I also owe much to the coworkers at GKN that have enthusiastically included and involved me to work on equal footing throughout the three years spanning the IM Master's study. Finally I would like to thank my family and friends for the unconditional support and guidance making the years studying as enjoyable as they have been.

The cover image is a reference to the GKN Aerospace produced TRF that initially sparked the interest for finding methods and ways to determine material thickness in new means. The style is a reference to the shifting focus of the real part to it's digital representation as a point cloud, the image is further discussed in (A.4).

Porsgrunn, 19th May 2021.

Ruben Austefjord

Contents

Preface 3

Contents 5
List of Figures . . . 7

1 Introduction 9
1.1 3D Scan as a Measurement System . . . 10
1.1.1 Image or Projected View Representation of 3D Data . . . 11
1.1.2 Volumetric Representation of 3D Data . . . 12
1.1.3 Mesh Representation of 3D Data . . . 12
1.1.4 Point Cloud Representation of 3D Data . . . 13
1.2 Problem Description . . . 14
1.2.1 General Point Cloud Methods . . . 14
1.2.2 General Material Thickness Evaluation . . . 17
1.2.3 Scripting ATOS Professional Software . . . 17

2 Methods 18
2.1 General Evaluation Framework . . . 18
2.1.1 Graph Theory as a Framework . . . 18
2.1.2 Neighbourhood Parameterization . . . 20
2.2 Material Thickness . . . 23
2.2.1 Euclidean Distance . . . 23
2.2.2 Parallel 2D Planes . . . 24
2.2.3 Best-Fit Planes . . . 26
2.2.4 Neighbourhood Optimized Planes . . . 27
2.2.5 Material Thickness Optimized Plane Search . . . 30
2.3 ATOS Professional . . . 34
2.3.1 ATOS Professional Scripts . . . 34

3 Results 35
3.1 Investigational Results . . . 35
3.2 Development Results . . . 36

4 Conclusion 37

5 Discussion **38**
5.1 Further Work 39

Bibliography **41**

A Software Source Code **43**
A.1 GOM Inspect Suite Script 43
A.2 GOM Inspect Suite Automation Script 44
A.3 Methods 46
A.4 Cover Image 54

List of Figures

1.1 Common Representations Of 3D Data, Image In Courtesy Of Qi, C.R. et al., 2017 [3]. 10
1.2 2D Representation Of 3D Object, Image In Courtesy Of Su, H. et al., 2015 [4]. 11
1.3 Volumetric Representation Of 3D Object, Image In courtesy Of Qi, C. R. et al., 2016 [6]. 12
1.4 Mesh Representation Of The Stanford Bunny [2] In Courtesy Of Novaković P. 13
1.5 Point Cloud Representation Of The Stanford Bunny [2] In Courtesy Of Qi, C. R. et al., 2017 [3]. 13
1.6 Heat Mapped Point Density Distribution On Disks For Random Points. . . 14
1.7 2D Material Thickness Example Plate With Point-wise Material Thickness Pairing. 16

2.1 Structure Of Nodes And Edges In Graphs. 18
2.2 4-Node Graph And It's Corresponding Adjacency Matrix. 19
2.3 4-Node Graph And It's Corresponding Adjacency List. 19
2.4 3D Sample Dataset With Two Distance Vectors. 23
2.5 3D Sample Dataset With Two Ideal Planes Across The Bottom And Top Surface. 24
2.6 3D Sample Dataset Where Material Thickness Is Evaluated At Random Points. 25
2.7 Example Of Least Squares Plane Fitting With 50 Randomly Distributed Points. 26
2.8 Least Squares Plane Fitting On 3D Sample Dataset With Varying p_0 And k Neighbouring Points. 27
2.9 Normalized Eigenvalues, Linearity, Planarity And Scattering Of A Linear And Planar Dataset. 28
2.10 SVD Plane Fitting On 3D Sample Dataset With Varying $p0$ And k Neighbouring Points. 30
2.11 2D Material Thickness From Point B By The Shortest Distance Plane Normal $\vec{b}$ Search. 31
2.12 Dataset A and B Comprised of Randomly Distributed 3D Points Along Two Opposing Disks. 32

2.13 Material Thickness Two Way Search Demonstrated On Dataset ***A***. 33
2.14 Material Thickness Two Way Search Demonstrated On Dataset ***B***. 33
2.15 CAD Model Of A Mount-Strut With Welding Junctions Indicated In Red, Model In Courtesy Of GKN Trollhättan [18]. 34

A.1 Comparison Of Gradients From Left To Right / Right To Left. 54
A.2 Gradient Used In Blend Function. 57

Nomenclature

List of abbreviations and symbols used throughout the thesis.

The *italic* font style is used to emphasise important phrases and names.

Symbol	**Explanation**
AWA	Adaptive Welding Automation
CAD	Computer-Aided Design
LIDAR	Laser Imaging, Detection And Ranging
ML	Machine Learning
PCA	Principle Component Analysis
SAR	Synthetic Aperture Radar
SLS	Structured Light Sensor
STL	Standard Triangle Language
SVD	Singular Value Decomposition
TRF	Turbine Rear Frame
λ_{1D}	Linear Dimensionality
λ_{2D}	Planar Dimensionality

1 Introduction

3D scanning has in recent years become more and more prevalent as the means to measure and digitize objects, surfaces and scenes in both scientific and commercial related applications. The reduction in both price and size alongside a gradual increase in accuracy of the measurement systems has made them widely available and applicable to a vast amount of use-cases. Typical examples are terrain and ground surveys by *Laser Imaging, Detection, and Ranging* (LIDAR) or *Synthetic Aperture Radar* (SAR). Which recently has taken the step from stationary ground surveys to aerial surveys with airplanes, drones and satellites providing data for previously untouched or unreachable terrain. Similar examples are 3D modelling for digital assets used in cinematics, videos or games. Where instead of building 3D assets from scratch, real objects are scanned and imported directly. Another example is it's use in archaeological surveys where excavation is either harmful or unfeasible.

At the manufacturing site GKN Aerospace Norway (GAN) this technology is currently used in production of aerospace engine components. As introduced in the previous master's project study AWA by Austefjord, R. 2020 [1], the key difficulties of manufacturing a Turbine Rear Frame (TRF) lay within the welding of forged and cold pressed metal-alloy components of varying thickness. For visual reference this is the part depicted on the front page of the thesis. To solve the key difficulties an automatized method of determining the material thickness in the welding junctions had to be developed. The suggested solution was presented as a system architecture containing three main steps. Firstly the raw material components were digitized through an optical 3D scanner where the resulting 3D point mesh was evaluated against an ideal CAD model to determine the material thickness at any point. Next the data was partitioned to only relate information regarding the welding junctions and transferred to a database for storage and accessibility. Lastly the data was accessed from the database and used to adjust the manufacturing parameters of the welding robot.

From the results of the master's project study it was concluded that the presented system architecture lacked the initial material thickness information from the ATOS Professional due to an outdated script. Upon further investigation and changes in GAN's production scheme it was decided that the master's thesis should continue the study of 3D scanned objects of similar nature the TRF. With a goal of achieving general material thickness evaluation either through the ATOS Professional software or directly from the point clouds produced by 3D measurement systems.

Throughout this master's thesis problems and solutions regarding 3D scanned objects will be discussed. Straying from the highly implementation based master's project, the goal is now to present a more generalized theoretical study of the the 3D scanned objects.

1.1 3D Scan as a Measurement System

As 3D scanning units and modules has reached satisfactory levels of accuracy, price and size, they have gradually been introduced as measurement units replacing traditional methods. An example of this is how traditional "ground scaling" in large has been replaced by physical on site LIDAR or aerial LIDAR and SAR. The reasoning behind the transition is largely due to the reduction in measurement time and cost as well as the ease of access in certain use-cases. Being only restricted by the measurement systems field of view, 3D scanning offers superior usability when its accuracy is sufficient for the task.

3D measurements are mainly achieved through two basic principles, reflection timing or observational analysis. Methods such as LIDAR and SAR determine the distance by timing the return of their respective emitted wave signals. While other methods such as *Structured Light Sensor* (SLS) and *Photogrammetry* work by analyzing the observed "images" captured by specialised cameras. Similar for both methods are that they produce datasets containing the best approximations of distances that have been observed as a collection of points referenced by their distance in 3D space to the measurement system.

Today the most notable ways of representing these datasets are through *Image-*, *Volumetric-*, *Mesh-* or *Point cloud-* representations. This data describes the structure and geometry of the scanned objects, surfaces or scenes. Depending on the use-case the different forms of representations may have different merits, ranging from visualization to analysis. A handy visual guide for the different representations using the *Stanford Bunny* [2] as render object can be seen in Figure 1.1.

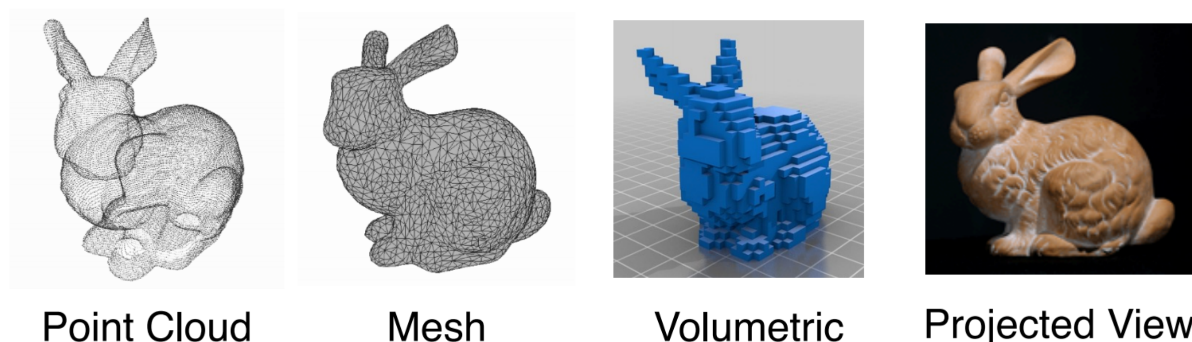


Figure 1.1: Common Representations Of 3D Data, Image In Courtesy Of Qi, C.R. et al., 2017 [3].

1.1.1 Image or Projected View Representation of 3D Data

The Image-based or "2D" representation of a 3D object is achieved through capturing images of the object at different view-points as demonstrated in Figure 1.2. This is particularly beneficial due to the vast amount of work that has been done surrounding analysis of 2D images. Notably methods such as image feature recognition, edge detection, classification and colour scheme analysis may be used directly in tandem with this form of representation.

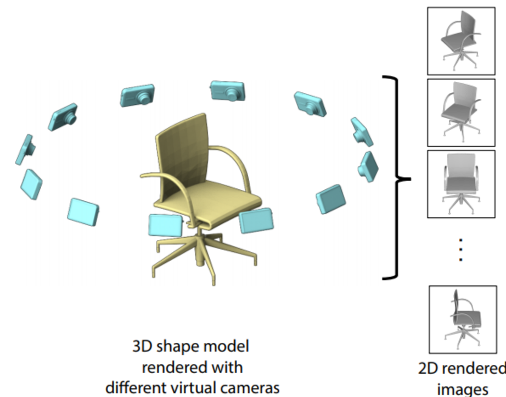


Figure 1.2: 2D Representation Of 3D Object, Image In Courtesy Of Su, H. et al., 2015 [4].

This representation however lacks the true "3D" aspect required for other methods of analysis such as volume, density, depth and other features that requires more than two dimensions. Notably there has been done work on reconstructing 3D objects from 2D images, Su, H. et al., 2015 [4] and also neural networks capable of mapping depth in images, Zhou, T. et al., 2017 [5]. This may in the future make the image representation very valuable for both 2D and 3D analysis.

This representation is currently best suited for simple 2D visualisation and in depth 2D analysis.

1.1.2 Volumetric Representation of 3D Data

The volumetric representation of a 3D object describes the dataset in voxels otherwise referred to as ”3D pixels”. These are size variable cubes defined by a length, height and depth together with information regarding their connected neighbours. They are most commonly used in medical environments to determine volumes and provide quick 3D visualization. An example of this representation is provided in Figure 1.3. This representation is best suited for quick 3D visualization and simple 3D analysis.

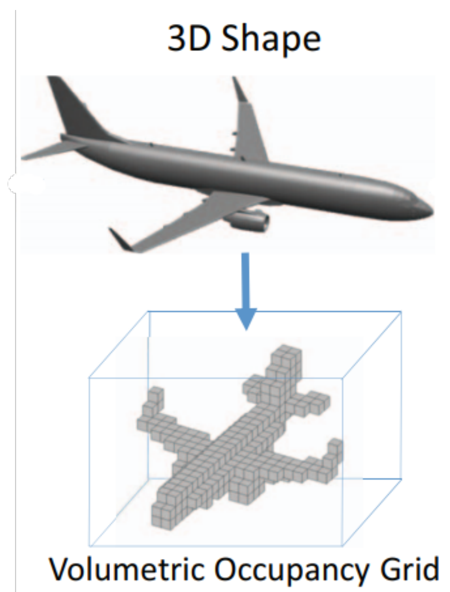


Figure 1.3: Volumetric Representation Of 3D Object, Image In courtesy Of Qi, C. R. et al., 2016 [6].

1.1.3 Mesh Representation of 3D Data

The surface-based mesh representation of 3D objects describe the triangulated external perimeter of the scanned object. Often produced from point clouds, this representation contains both information regarding the coordinates of the points as well as which other points make up the perimeter-triangles connected to that point. Displayed in Figure 1.4 meshes attempt to optimize the grid such that it is fully connected and closed. This can be further scaled up and down which means that there exist cases where the mesh has to interpolate across ”incomplete data”. Similarly holes or missing data has to be enclosed such that the mesh often is extrapolated beyond the original dataset in order to ensure a fully connected and closed surface.

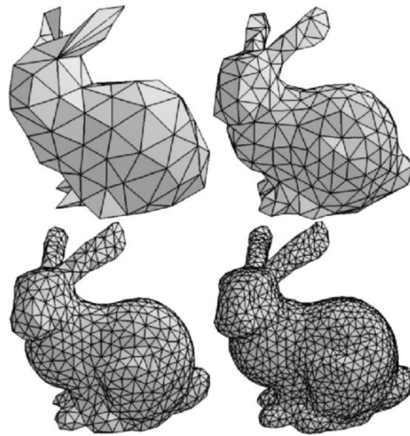


Figure 1.4: Mesh Representation Of The Stanford Bunny [2] In Courtesy Of Novaković P.

This representation has close ties to graphs and is very well suited for creating digital aspects or replications of physical objects. It provides good visualization and possibilities for in depth 3D analysis.

1.1.4 Point Cloud Representation of 3D Data

The Point-based representation of 3D objects describe the 3D coordinates of all measured points in a dataset. Commonly referred to as a Point cloud, the representation can be considered as the unedited raw output of a 3D scan as displayed in Figure 1.5. Depending on the accuracy and limits of the measurement system, a point cloud may be highly varied in regards to point density. Together with the coordinates, the point can have descriptors such as color and temperature if such data also is available.

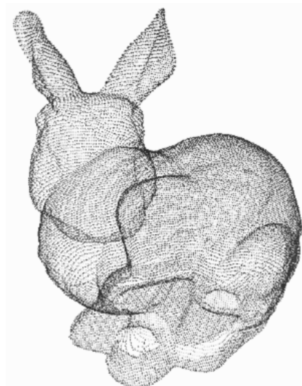


Figure 1.5: Point Cloud Representation Of The Stanford Bunny [2] In Courtesy Of Qi, C. R. et al., 2017 [3].

This representation is best suited for in depth 3D analysis without compromising any information.

For the work in this master's thesis the methods will be developed from the point cloud representation as it most closely resembles the general raw data-output of any 3D scanning system and retains the maximum amount of information.

1.2 Problem Description

The intent of this section is to introduce the problems surrounding developing *General Point Cloud Methods*, *General Material Thickness Evaluation* and of *Scripting the ATOS Professional Software*.

1.2.1 General Point Cloud Methods

Datasets produced by the means of 3D scanning varies almost indefinitely from object to object. Moving, rotating, altering or exchanging the scanned object will result in a completely unique dataset. Measurement uncertainties, accuracy and resolution also ensures that any two scans would be unique though similar. Temperatures, pollutants, obstructions, air currents and all other conceivable sources of noise likewise has altering effects on the final dataset. Much of this variability bears little significance because it often contributes orders of magnitude less to the total deviation from the theoretical ideal when compared to the accuracy of the measurement system.

Out of the sources of variability there are mainly three that pose a challenge when attempting to generalize methods across a large manifold of datasets. The first of which is the *point density* in the dataset. Example illustration shown in Figure 1.6.

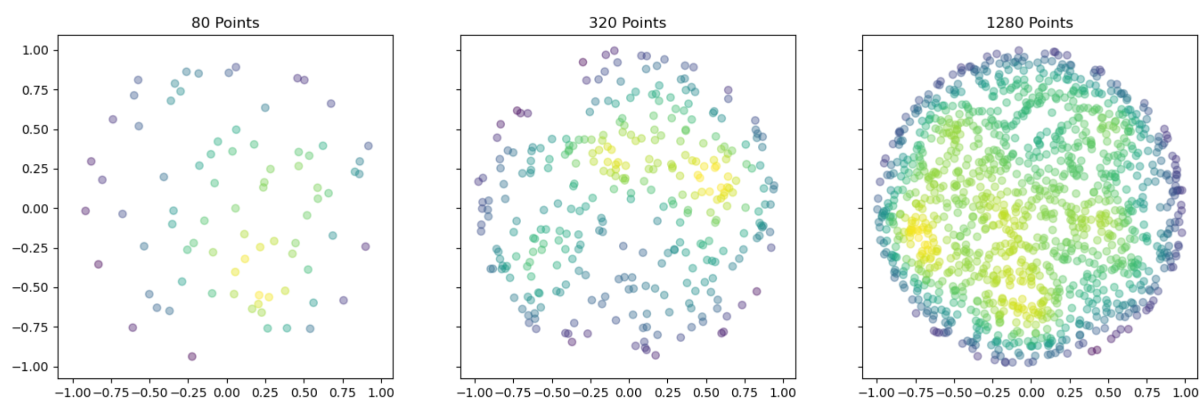


Figure 1.6: Heat Mapped Point Density Distribution On Disks For Random Points.

Measuring a perfectly flat and rectangular 2D surface (plane) with a flawless measurement system, the surface could in theory be represented by the four edges of the plane. Such a scan would produce a dataset of only four points while perfectly representing the real object. Reintroducing the notion that the surface might not be perfectly flat and that there certainly are measurement variances and uncertainties that compound across the surface. The dataset would include far more points to represent the scanned object. This can be further complicated by scanning a more complex surface such as a sphere. Considering a perfect sphere, the ideal point cloud representation would be infinitely dense as there would be no edges in the object that could simplify the representation. Comparing the two ideal representations there is both a limit to how sparse and dense the point density can be within a dataset. Where it is practically impossible to reach either end of the ideals for a real measurement system. The resulting point density in the dataset is then tightly connected to the capabilities of the measurement system and the type/shape of object scanned. Resulting in an inherently infinite variance in datasets depending on what is being measured and what measured them. This means that methods applied to such datasets have to be invariant to point density, e.g. if the goal is to classify a plane, it should be able to classify the plane regardless of the dataset consists of four points or one million.

Another challenge when working with point clouds is the topic of *orientation*. The notion of up/down, left/right, front/back does not simply apply to objects that have been 3D-scanned. The objects can be placed in any position/orientation within the measurement systems viewpoint. Likewise, the measurement system can be placed in any position/orientation surrounding the object. This fact decouples any notion of orientation in the dataset and imposes the frame of reference from the measurement system to the measured object. Resulting in any point within the dataset soley being a coordinate point based on the measured distance from the measurement system. This means that methods applied to such datasets have to be invariant to orientation and reference systems, e.g. ”Up” can’t simply be defined as increasing ”z” values.

Another general challenge is that a dataset does not posses any prior heuristic or empirical knowledge regarding itself as the only information available in such a dataset is the individual coordinates of the points. Importantly this means that any single point has no information regarding its neighbouring points or if that point is part of feature ”X or Y”. At first glance the lack of this knowledge within a dataset might not be thought of as particularly important. A given object does after all not need to know what or where it is in order to exist. Which is to say that for the object, only it’s information is important, not the knowledge.

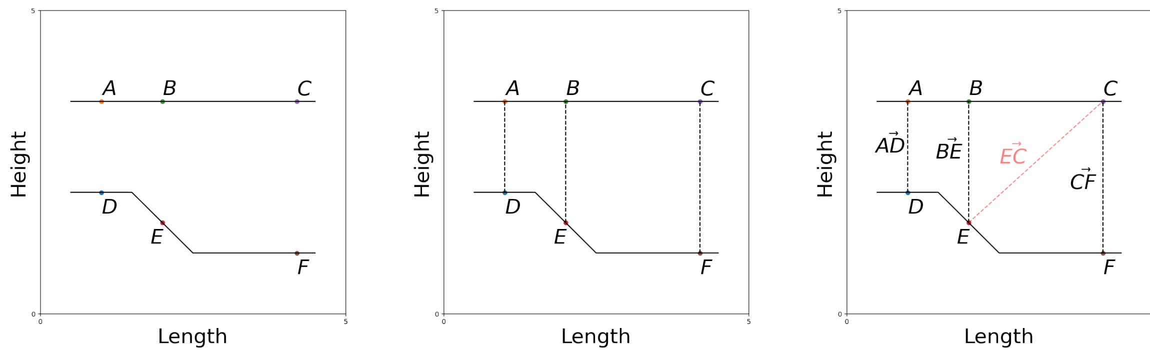


Figure 1.7: 2D Material Thickness Example Plate With Point-wise Material Thickness Pairing.

To illustrate these three general challenges with one example, Figure 1.7 depicts a 2D cross section of a metal plate. The top surface is defined by the 3 points (A, B and C) which is distributed linearly along the length of the plate. The bottom surface is comprised of the points (D, E and F) where the height of the surface varies along the length of the plate. The goal is to find the material thickness, defined in this case as the distance between the top and bottom surface in the height direction. In the way that this example is constructed, finding the material thickness is not very difficult. The solution is simply to calculate the *Euclidean Distance* between the point pairs (A, D), (B, E) and (C, F). However, consider a change in *point density*, let the bottom surface now be defined by 9 points along the surface. Point pairs can no longer be formed directly in the vertical height direction for every point, which means that some sub-optimal point pairing has to be made for at least 6 of the points on the bottom surface. Here the same calculation would not result in the correct material thickness, take Figure 1.7 vector $\vec{EC}$ as an example. The next, maybe obvious problem is that if the plate was rotated even slightly, e.g. it's *orientation* changed, the height direction would no longer equate to the material thickness. Similarly if the shape changed from a plate to a curved object, the material thickness might not even be defined in the same direction across the entire surface. Lastly, none of the points inhabit some *knowledge* regarding whether it is located on the top or bottom surface. Nor any information regarding what point it is supposed to be paired with in order to calculate the distance.

From these challenges it can be deduced that general methods has to be invariant to variations in point density, orientations, shapes and heuristic and empirical knowledge.

1.2.2 General Material Thickness Evaluation

Material thickness can generally be thought of as the distance between two opposing sides of the same fully-connected surface. For simple shapes such as cubes this usually is the shortest of the three dimensions with regards to height, width and length. The material thickness can then be defined by that direction across the entire object. However for more complex shapes with curvatures, holes and extrusions, the material thickness can't be defined solely in one direction as the material thickness now depends directly on the point of measurement. Material thickness should then be defined as the distance from the point of measurement to the opposing side of the same fully-connected surface. Or in general as the shortest distance to another opposing surface.

From a measurement standpoint, material thickness is a relatively simple property to measure as long as the direction of the measurement is defined. This definition is however often made from prior experience or knowledge. The problem in defining the direction is often to *know* where one surface ends and the next begins, while also *understanding* that one surface opposes the other in a ”back/front”, ”top/bottom” or ”left/right” pairing.

Proposing a method that may be generally applicable thus has to rely on the same prior knowledge or be able to incite it by itself. There is some return of value presented by the increase in accuracy from material thickness measurements being done on a costly but accurate 3D scan apposed to a manual measurement. However the greatest return would be if the measurement could be achieved autonomously without relying on any prior knowledge or experience.

1.2.3 Scripting ATOS Professional Software

To summarize the problem description regarding the scripting of the ATOS Professional software from Austefjord, R. 2020 [1] we have:

A script capable of autonomously detecting and evaluating the material thickness within the welding junctions has to be developed. Using built-in functions of the software such as script-ability and material thickness evaluation compared to CAD files.

2 Methods

In this chapter methods for generalizing and classifying the 3D point cloud datasets are presented through the subsection *General Evaluation Framework*. Followed by methods for evaluating material thickness in *Material Thickness* and resolving the method for scripting *ATOS Professional*.

2.1 General Evaluation Framework

To tackle the unending quarrel of density variance, lack of orientation and knowledge, some known theory and mathematical theorems can be considered in order to form a basic ”reference” frame for the unstructured 3D point cloud measurements. Making it possible for more sophisticated, general and specific methods to be applied to the datasets without having to rely on specialised heuristic or empirical knowledge.

The intent and goal for this section is to introduce the methods that extrapolate upon the information available in the datasets.

2.1.1 Graph Theory as a Framework

When the data is presented in the form of either *volumetric*, *point cloud* or *mesh* structures, it is simple to formulate many of the questions as graph theory problems. In graph theory the data is generally presented in terms of nodes and edges as shown in Figure 2.1. For *volumetric* representation the voxels would represent the nodes while the connections to the nearby voxels define their edges. In a point cloud the points directly correspond to nodes whereas the edges are the distances to all other points. A mesh structure is inertly already a graph and thus require no new formulation. Common for all of these representations is that the graphs are undirected, meaning there is no preferred direction to the edges.

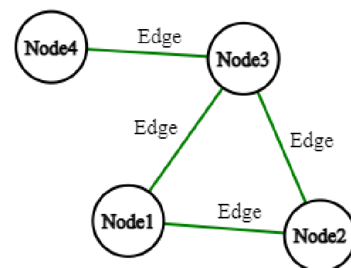


Figure 2.1: Structure Of Nodes And Edges In Graphs.

There exist a couple ways to represent graphs, most commonly they are represented through either an adjacency matrix, ref. Figure 2.2 or an adjacency list, ref. Figure 2.3. Considering that an adjacency matrix lists all possible edges between nodes even if there are none, the size of the matrix grows proportionally with the size of the dataset squared. An adjacency list however only denotes which edges exist and thus only grows proportionally with the amount of connections/edges in the dataset. Datasets produced from 3D scanning will be very large thus it only makes sense to continue with the representation in form of an adjacency list.

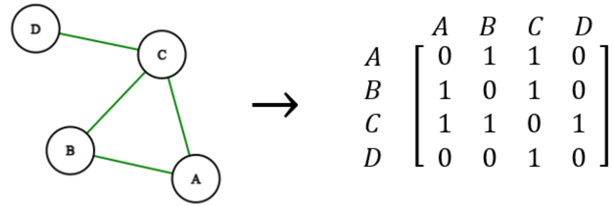


Figure 2.2: 4-Node Graph And It's Corresponding Adjacency Matrix.

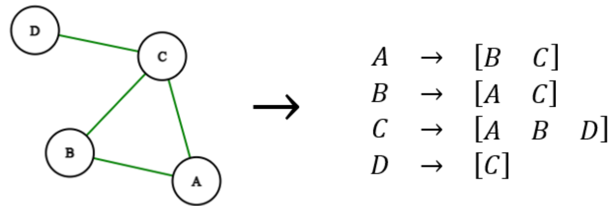


Figure 2.3: 4-Node Graph And It's Corresponding Adjacency List.

Depending on the format of the input, the adjacency lists may be either sparse or dense depending on the amount of edges coupled to a given node. For both the *volumetric* and *mesh* representation, there only exist a handful of edges per node, making them naturally sparse. This is great for low memory consumption and computational power. Point clouds are however naturally dense since all nodes are connected to each other, effectively making the representation an adjacency matrix filled with ones. Working with point clouds as graphs is therefore cumbersome if computational power and memory is limited. Luckily most of the edges are abundant and provide no particularly insightful information. The adjacency list may therefore be shorted to only contain a certain specified number k edges of interest.

KD-Trees

As is the case with both the *volumetric* and *mesh* representations, the general interest in *point clouds* are the closest neighbours of the points/nodes. Evaluating the k nearest neighbours of a graph is fortunately a widely covered problem with many solutions. One of which is the "K-Nearest-Neighbour" (KNN) search introduced by Friedman, J. H. et al., 1977 [7]. This function works by structuring the data in a binary tree known as a kd-tree. This structure separates the dataset based upon a defined delimiter which in this case has the goal of grouping similar data. From Friedman, J. H. et al., 1977 [7] the tree delimiter is the point in the dataset where the greatest spread occurs, e.g. the difference between the minimum and maximum values. This equates to be the median of the dataset and is known as the *Standard Split*. The delimiter used for constructing the kd-trees has further been improved through several iterations where today the most commonly applied is the *Canonical Sliding-Midpoint Split* introduced by Maneewongvatana, S. et al., 1999 [8]. Once a kd-tree is constructed, any point n can be queried for their nearest k neighbours without having to calculate the distances to all possible points in the dataset. This is made possible by only evaluating the points in the same tree cluster/branch, greatly reducing the computational effort.

Using this method, point clouds can be formulated as graphs without sacrificing memory or computation power and can be represented by an adjacency list equal to the ones used for both *volumetric* and *mesh* structures.

2.1.2 Neighbourhood Parameterization

For the sake of analyzing the dataset, there is often a desire to find similarities or features within the data in order to classify points or areas depending on the grouping. Such data clusters can be evaluated with different tools, ranging from shape recognition (Barr, A. H., 1981 [9]) to modern day neural network classifiers (Zhu L. et al., 2019 [10]). Direct examples are separating the bare ground from vegetation (Jiang, Y. et al., 2019 [11]) and from urban features (Weinmann, M. et al., 2015 [12]). The list of possible applications are not short, indicating that finding such data clusters within the point cloud could provide highly insightful and valuable information.

There are several methods for finding such clusters, however many of them are very specialised towards which objects, surfaces or scenes that are expected as inputs, or from which measurement system the data was collected. The problem is in general that if given an ordinary coffee-mug, two different measurement systems could provide vastly different point resolutions of the mug once scanned. Say one is a point cloud of 30 000 points A, while the other is 5 000 points B. If an algorithm should classify clusters that are either flat or have a curvature above a certain percentage. The same algorithm might evaluate the scanned mug A to be flat in a region given by a point on the mug and the $k = 10$

closest points. Whereas for B, due to the lower point density, the same ten closest points would be much further removed in distance, revealing that there was a curve. Even worse would be to use the same algorithm on an entirely different object. It is pretty obvious that the number of k nearby neighbours has to have some form of variability with the point density as well as the shape of the object, surface or scene.

In Weinmann, M. et al., 2014 [13] it is said that the best selection of the number k nearest neighbours can be determined by selecting the value of k which yields the minimum *Shannon Entropy*. This ensures that the point selection includes enough points that a local descriptive cluster can be formed while also ensuring that the greatest local information divide has been reached.

Shannon Entropy for 3D Structures

Presented in *A Mathematical Theory of Communication* by Claude Elwood Shannon [14], the Shannon Entropy is used to evaluate the information entropy of a given probability distribution. Denoted by $H(x)$ the Shannon Entropy is a measure of the uncertainty of an event "x", ranging from $0 \rightarrow \inf$. If the probability distribution only has one outcome, there exist no uncertainty in the distribution and the Shannon Entropy equates to 0. Mathematically the function is represented in (2.1)

$$H(x) = -\sum_{x} P(x) \log P(x) \tag{2.1}$$

were $P(x)$ is the probability distribution of events "x" and log is the base-2 logarithm. An unlikely outcome will score a high Shannon Entropy value, whereas the most likely outcome will score the lowest value of Shannon Entropy.

Considering a point p_0 in the dataset and it's nearest k neighbours. A covariance matrix C can be calculated for the collection of points from $p_0 \rightarrow p_k$ as shown in (2.2) and (2.3)

$$\bar{p} = \frac{1}{1+k} \sum_{i=0}^{k} p_i \tag{2.2}$$

$$C = \frac{1}{1+k} \sum_{i=0}^{k} (p_i - \bar{p})(p_i - \bar{p})^T \tag{2.3}$$

where from the covariance matrix, the three eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ can be calculated. Such that λ_1 explains the greatest variance in the point collection, followed by λ_2 and λ_3. Since the dimensionality of the dataset is 3D, 100% of the variation can be explained by the three eigenvalues. Following Weinmann, M. et al., 2014 [13], the eigenvalues can be normalized giving the eigenvalues e_1, e_2 and e_3 as defined in (2.4), (2.5) and (2.6)

distributed from $0 \rightarrow 1$

$$e_1 = \frac{|\lambda_1|}{|\lambda_1| + |\lambda_2| + |\lambda_3|} \tag{2.4}$$

$$e_2 = \frac{|\lambda_2|}{|\lambda_1| + |\lambda_2| + |\lambda_3|} \tag{2.5}$$

$$e_3 = \frac{|\lambda_3|}{|\lambda_1| + |\lambda_2| + |\lambda_3|} \tag{2.6}$$

where $|\lambda_n|$ in this thesis is the abs(Re(λ_n)) of eigenvalue n.[1] It is further shown in Weinmann, M. et al., 2014 [13] that the eigenvalues can be exploited to describe certain dimensionality features such as *linearity* L_λ, *planarity* P_λ and *scattering* S_λ described in (2.7), (2.8) and (2.9)

$$L_\lambda = \frac{e_1 - e_2}{e_1} \tag{2.7}$$

$$P_\lambda = \frac{e_2 - e_3}{e_1} \tag{2.8}$$

$$S_\lambda = \frac{e_3}{e_1}. \tag{2.9}$$

As the eigenvalues are normalized and range from $0 \rightarrow 1$ and importantly sum up to one for both $e_1 + e_2 + e_3 = 1$ and $L_\lambda + P_\lambda + S_\lambda = 1$, the values can be thought of as "pseudo" probabilities. The value of e_1 will for instance indicate the "probability" of the dataset being described solely by e_1. Likewise the "probability" that the dataset is linear or one-dimensional can be evaluated by L_λ. The values can then carefully be applied to the Shannon Entropy function, ensuring that no eigenvalue equates to 0 by adding an infinitesimally small ε. From this the *eigenentropy* e_λ and *Shannon Entropy* E_λ are defined according to (2.1) as follows in (2.10) and (2.11)

$$e_\lambda = -e_1 \log e_1 - e_2 \log e_2 - e_3 \log e_3 \tag{2.10}$$

$$E_\lambda = -L_\lambda \log L_\lambda - P_\lambda \log P_\lambda - S_\lambda \log S_\lambda. \tag{2.11}$$

[1] The $|\lambda_n|$ operation is theoretically unnecessary since the covariance matrix C is said to be positive definite. This claim is beyond the scope of this thesis, however the interested reader is referred to Weinmann, M. et al., 2014 [13] page 5.

2.2 Material Thickness

In this section the approach of creating a general applicable method for evaluating *material thickness* is described. The methods are developed to match the underlying human intuition towards material thickness while gradually utilizing the general methods presented in *General Evaluation Framework* (subsection 2.1). The methods are presented in order from least to most general.

2.2.1 Euclidean Distance

Given the simple rectangular dataset in Figure 2.4 with respects to material thickness. Much alike the 2D example in *General Point Cloud Methods* (subsection 1.2.1) this 3D dataset has points distributed such that the height direction can be defined as the direction of the material thickness. The points are also distributed evenly across a grid such that the direction-lines in the grid intersecting the points are ⊥ to each other.

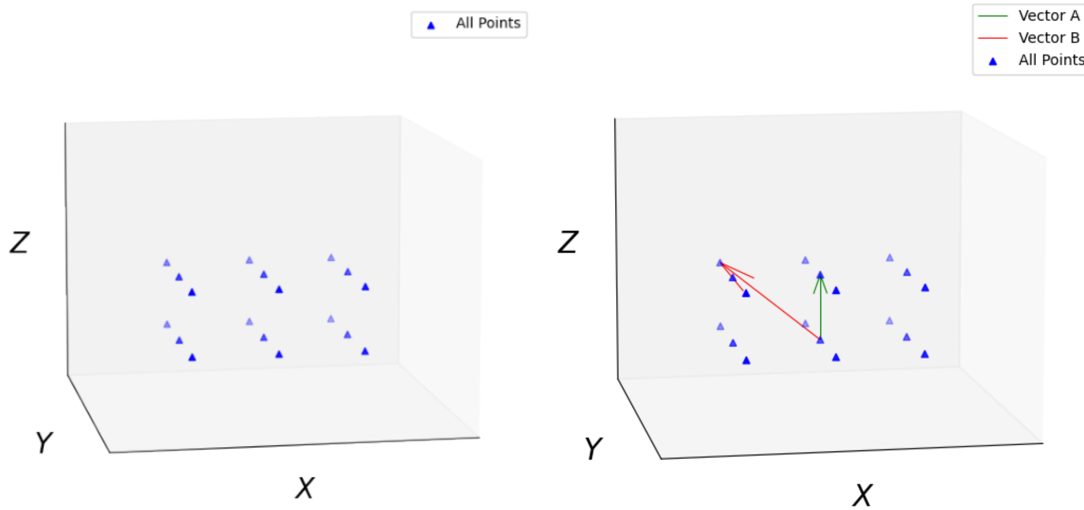


Figure 2.4: 3D Sample Dataset With Two Distance Vectors.

The distance between any point pair p and q can be calculated as the *Euclidean Distance* "$dist(p,q)$" between them as seen in (2.12)

$$dist(p,q) = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2 + (q_z - p_z)^2}. \tag{2.12}$$

Knowing that "Z" is the direction of the material thickness *Vector A* from Figure 2.4 equates to the correct point pairing to determine the material thickness.

This method is comparable to manual thickness measurements with e.g. calipers. The direction of the measurement is known and point p and q is selected to adhere to this knowledge. This makes the method not generally applicable as it relies solely on heuristic and empirical knowledge.

2.2.2 Parallel 2D Planes

Given a box, book or simply a sheet of paper, our perception of depth within an object allows us to determine the direction of the material thickness almost immediately. We identify the surfaces on the objects and choose the ones with the smallest distance between as the depth. Consider the same simple rectangular dataset in Figure 2.5 with respects to material thickness. Expressing our intuition of depth, we have that the distance between the two ideal and parallel planes equates to the material thickness.

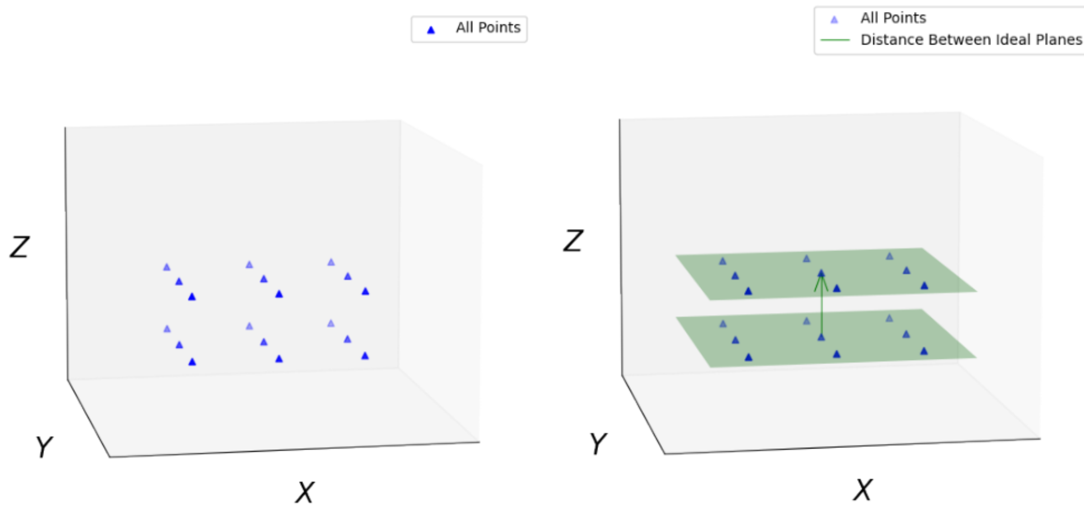


Figure 2.5: 3D Sample Dataset With Two Ideal Planes Across The Bottom And Top Surface.

Expressing this mathematically the method can be divided into three steps: a), b) and c) each explaining the process taken to determine the material thickness.

a) An initial point p_0 has to be selected from where the material thickness should be evaluated. This may be done through the end-user's specific choice, a random selection or through an incremental loop if all points are to be evaluated. The only constraint is that the point p_0 has to exist in the dataset.

b) A 2D plane has to be created through the point p_0, such that the plane resembles the surface of the object. Generally this can be achieved with the function for a plane, described in (2.13)

$$ax + by + cz + d = 0 \tag{2.13}$$

where (x, y, z) is a point located in the plane, while $[a, b, c]$ is a direction vector perpendicular to the plane. Having selected a point p_0 in *a*), this only leaves the direction vector as unknown. Recalling that the direction of the material thickness is defined as the height or "Z" direction in the dataset, the normal vector for the plane is known. Expressing the equation for a plane with respect to a point $A = (x_0, y_0, z_0)$ and the plane's normal vector $\vec{n} = [a, b, c]$ we have in (2.14)

$$a(x - x_0) + b(y - y_0) + c(z - z_0) = 0. \tag{2.14}$$

c) Steps *a*) and *b*) are repeated for a point on the opposing surface such that two parallel planes are formed. The material thickness can then be found from calculating the distance between the planes given by (2.15)

$$dist = \frac{|d_1 - d_2|}{\sqrt{a^2 + b^2 + c^2}}. \tag{2.15}$$

This method more closely resembles our thought process when finding the material thickness of an object. Though relying on the prior knowledge regarding the direction of the material thickness. If the point pairing is selected at random, a distance will be found, however not necessarily in the direction of the material thickness as shown in Figure 2.6.

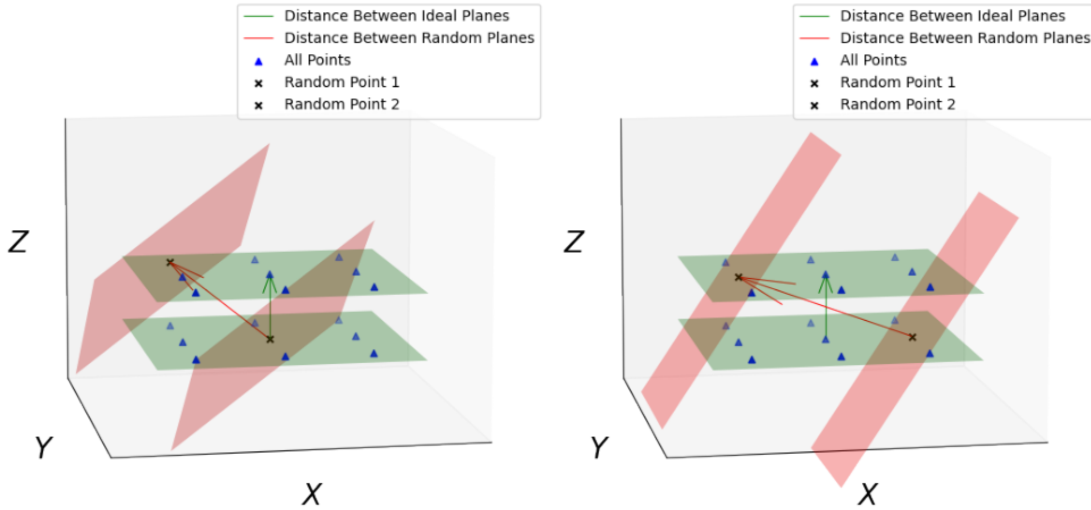


Figure 2.6: 3D Sample Dataset Where Material Thickness Is Evaluated At Random Points.

2.2.3 Best-Fit Planes

Evaluating the flaws of the *Parallel Planes* method, the intention is not to find "a" plane, but a plane that represents the surface the selected point lies within. This can be achieved by evaluating the nearest neighbours of the initial selected point through a KD-tree search. In this way the closest k neighbouring points can be evaluated as a local cluster explaining the surface surrounding the initial point. From this cluster a 2D plane can be fitted to match the points through a least squares optimization problem. Implementing the least squares method in accordance with the matrix formulation described under the *Ordinary Least Squares* page [15]. The cluster can be expressed as an overdetermined system on the form $\sum_{j=1}^{p} X_{ij}\beta_j = y_i$, $(i = 1,2,\ldots,k)$, $(j = 1,2,\ldots,p)$ where p is the number of equation variables. Equation (2.13) can be solved for z and expressed as $y = X\beta$ where

$$X = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_k & y_k & 1 \end{bmatrix}, \ \beta = \begin{bmatrix} a \\ b \\ d \end{bmatrix} \text{ and } y = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{bmatrix} \text{ for } k \text{ points in the cluster.} \tag{2.16}$$

A plane can then be fitted to match the points through a least squares optimization problem as formulated in (2.17), (2.18) and (2.19)

$$\textit{arg min:} \ S(\beta) = \sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 = (y - X\beta)^2 \text{ where it follows that:} \tag{2.17}$$

$$X^T X\beta = X^T y \tag{2.18}$$

$$\beta = (X^T X)^{-1} X^T y \ . \tag{2.19}$$

Demonstrated with 50 randomly distributed points in Figure 2.7, this can be utilized in step b) from *Parallel 2D Planes* (subsection 2.2.2) to form the plane representing the local surface.

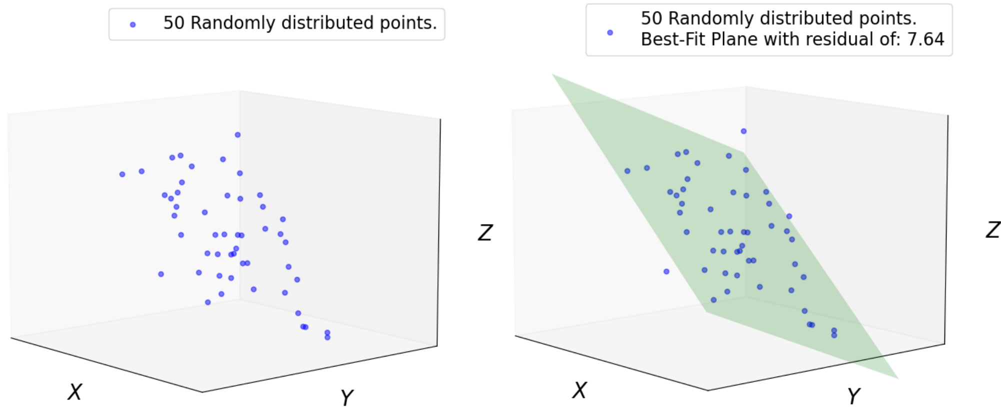


Figure 2.7: Example Of Least Squares Plane Fitting With 50 Randomly Distributed Points.

This improvement accomplishes an invariance to prior knowledge by removing the need to know the direction of the plane normal. However it is still very sensitive to both changes in point density and orientation as well as the initial selected point. This can be demonstrated in Figure 2.8 using the simple 3D dataset from (subsection 2.12).

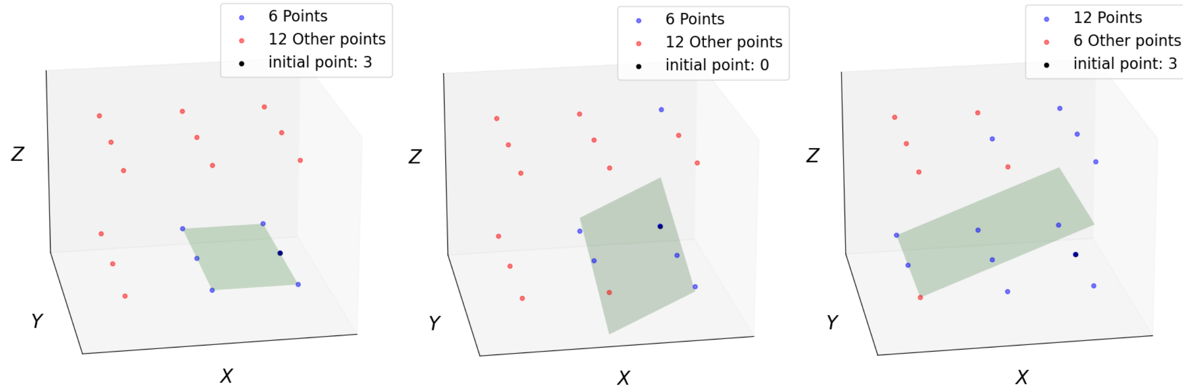


Figure 2.8: Least Squares Plane Fitting On 3D Sample Dataset With Varying p_0 And k Neighbouring Points.

Viewing Figure 2.8 from left to right, the different subplots showcase that 1) if the number of k nearest neighbours are properly selected (displayed as blue dots surrounding the black p_0 initial point) a plane (displayed in green) can be created in the direction of the surface. 2) If the initial point lies such that one of the set number of closest neighbours lie on another surface, the plane is fitted across the two surfaces and fails to explain the singular surface of the initial point p_0. Finally in 3) if the number k exceeds a certain threshold dependant on the point density, the plane will likewise fail to explain the surface surrounding the initial point as it is forced to include points in other surfaces.

There also exist a bias within the *Least Squares* method used to fit a plane when it comes to the preferred direction to minimize the error. In this case the solution is solved for the z direction implying that for this dataset it is "known" that z or "height" is the preferred direction to optimize for.

2.2.4 Neighbourhood Optimized Planes

Once again reviewing the flaws of the previous method with respects to a general solution, there are mainly two improvements that can be implemented. Firstly the number of k nearest neighbours should be scalable depending on the surface where the initial point lies. For this the *Shannon Entropy* (subsection 2.1.2) can be incorporated to determine how many points should be evaluated before the "uncertainty" of the dimensionality in the cluster increases. In other words the Shannon Entropy should be minimized such that only points within a single plane should be included. The second improvement is

to remove the bias towards any direction when fitting a plane to the cluster. This can be done through a *Singular Value Decomposition* (SVD) such that the plane is fitted by allowing for errors across all the individual variables x, y, and z, not restricting the error to only occur in one "known" vertical direction.

From *Shannon Entropy for 3D Structures* (subsection 2.1.2) we have that the dimensionality can be described by the normalized eigenvalues and eigenvectors. Notably a cluster of points can be said to confide within a 2D surface of arbitrary direction or rotation if the third eigenvalue is minimal. This can be explained by considering the normalized eigenvalues where $e_1 \geq e_2 \geq e_3$ and $e_1 + e_2 + e_3 = 1$. If the first eigenvalue e_1 explains close to 100% of the variation within the cluster, the points that make up the cluster would only lie along the first eigenvector or in other words along a linear line. If e_1 and e_2 explain 50% of the variance each, the points that make up the cluster would be spread out symmetrically across the first and second eigenvector. These two concepts related to the eigenvalues are demonstrated in Figure 2.9.

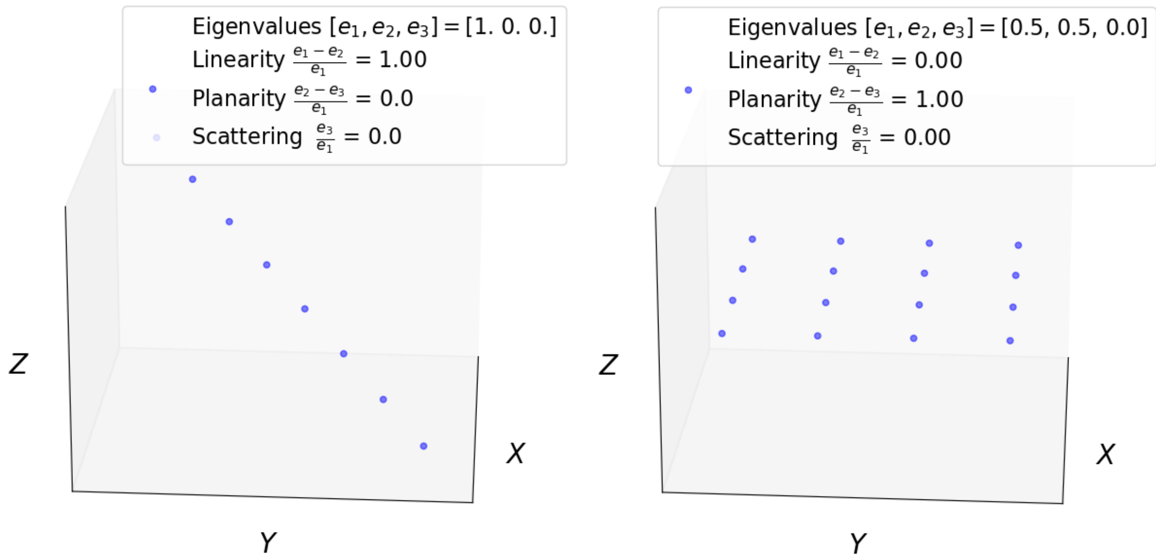


Figure 2.9: Normalized Eigenvalues, Linearity, Planarity And Scattering Of A Linear And Planar Dataset.

Furthermore for any combination where $e_1 + e_2 \approx 100\%$ the points that make up the cluster must be confined to a 2D plane made up by the first and second eigenvector. This is true because e_3 contributes approximately nothing towards explaining the variance and the cluster is almost fully explained by the two most important eigenvalues spanning a 2D plane. This feature of *planar dimensionality* λ_{2D} can be expressed as low or close to zero *scattering* S_λ in (2.20)

$$\text{The dataset is } \lambda_{2D} \text{ if: } S_\lambda = \frac{e_3}{e_1} \approx \frac{e_3 + \varepsilon}{e_1 + \varepsilon} \approx 0 \tag{2.20}$$

if ε is an infinitesimally small number to avoid any possible numerical division errors. These features can be taken advantage of by evaluating either the λ_{2D} directly for the cluster or by evaluating the specific Shannon Entropy given by (2.21)

$$S_k = -L_\lambda \ln(L_\lambda + \varepsilon) - P_\lambda \ln(P_\lambda + \varepsilon) - S_\lambda \ln(S_\lambda + \varepsilon) \tag{2.21}$$

where S_k is the Shannon Entropy for a cluster given by the k number of nearest neighbours. S_k can in turn be optimized for the number k such that only points on the same surface is evaluated and thus removing the problems displayed in Figure 2.8. Note that in theory the Shannon Entropy would optimize for a linear dimensional λ_{1D} system because the least uncertainty in any given 3D point cluster would be if e_1 explained 100% of the variance. In reality due to both measurement uncertainties and the nature of the objects that are scanned, such a scenario can't, or is extremely unlikely to occur when facing real datasets. There is still one scenario where this is true, and that is if only two points are evaluated, pairing only two points will always produce a λ_{1D} system simply from the fact that a straight line can be drawn through any two points provided they are not the same point. The optimization of S_k should then have a lower cutoff range at least above 2 in order to avoid this scenario, otherwise the ideal k would always be equal to 2. When this scenario is taken care of the least uncertainty in a cluster is then defined by one dimension higher where both e_1 and $e_2 > 0$. Meaning that for real datasets the Shannon Entropy optimizes for λ_{2D} systems.

Having optimized k from a range of e.g. $[5 \rightarrow 40]$ the resulting S_k is a cluster of the initial selected point p_0 and it's k nearest neighbours which in real practical scenarios should be the best λ_{2D} representation of the surface in which p_0 lies. From this cluster a best-fit SVD plane can be constructed such that the material thickness can be determined.

From the *Total Least Squares* page [16] the SVD best-fit can be implemented as a plane constructed from the least significant, right singular vector v_3 and the initial selected point p_0 by using equation (2.14). From the SVD deconstruction of the 3D point cluster the resulting v vector in $u\sum v$ contain much alike the eigenvectors the direction of the highest order explainable variance. This means that v_3 is the best-fit vector orthogonal to the plane described by v_1 and v_2.[2]

[2]The best-fit SVD method is not expressed in it's entirety as this falls outside the scope of the thesis. The interested reader is referred to [16] and [17].

Combining the two presented methods and applying them to the simple dataset from (subsection 2.2.1) makes the *Neighbourhood Optimized Planes* method capable of structuring local regions around a given point p_0 where the surface can be expressed best as a λ_{2D} plane. The sensitivity towards changes in point density and point localisation is highly decreased as seen when comparing the results with the identical dataset in Figure 2.8 to Figure 2.10. Furthermore the bias towards a certain orientation has been removed by introducing the SDV best-fit plane.

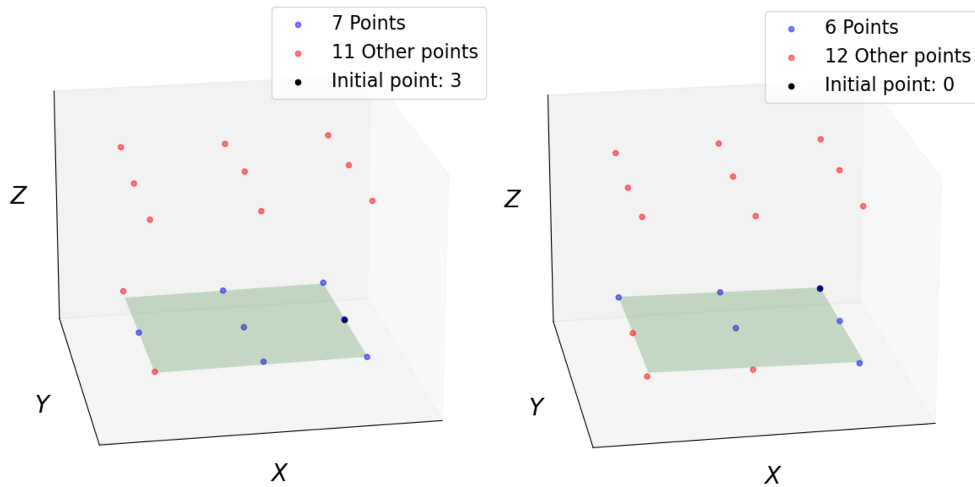


Figure 2.10: SVD Plane Fitting On 3D Sample Dataset With Varying $p0$ And k Neighbouring Points.

The method satisfies the initial requirement from (subsection 1.2.1) such that it is invariable towards changes in *point density* and *orientations*. However the final step in the algorithm presented in (subsection 2.2.2) still has a reliance upon finding parallel planes, thus relying on a *known* object *shape*.

2.2.5 Material Thickness Optimized Plane Search

It is fairly simple to demonstrate why more complex systems can't be solved in this manner. Evaluating the material thickness across the 2D example in Figure 2.11 the thickness varies along the X-axis as a result of the bottom surface not being flat. Picking the initial point B, a plane can be constructed such that the local feature for the surface surrounding B is represented as a λ_{2D} plane. Investigating where the normal $\vec{b}$ to the plane intersects with the other surface, a plane can be constructed in the same manner to fit the nearby local region. This plane will however not be parallel to the one in B, nor will the length of the normal $\vec{b}$ between them be the material thickness.

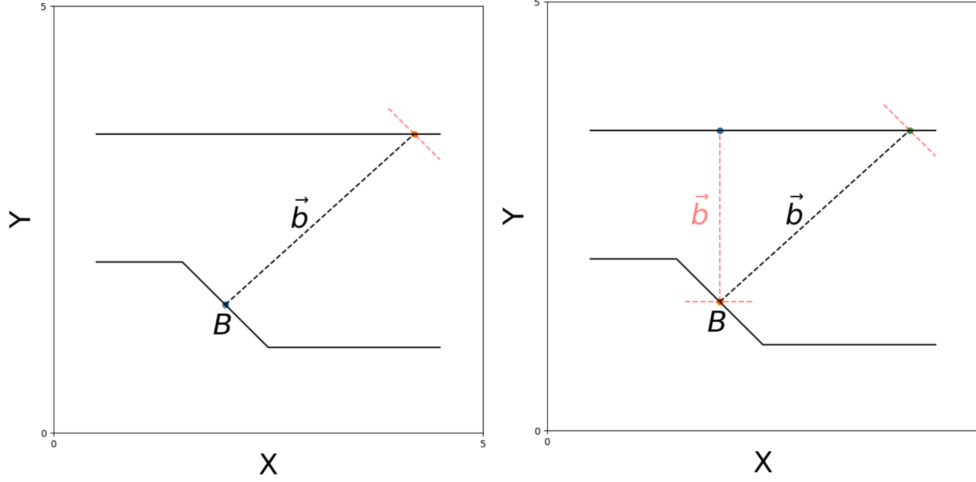


Figure 2.11: 2D Material Thickness From Point B By The Shortest Distance Plane Normal $\vec{b}$ Search.

In Figure 2.11 the correct material thickness is the vector $\vec{b}$, notably this is a normal vector constructed from the intersection plane found in the initial search from point B. This means that the following two part search algorithm (Listing 2.1) can be carried out in order to ensure that the shortest distance is found.

```
1 Select an initial point p0.
2 Determine the best k nearest neighbours through Sk (2.21) .
3 Construct a SVD best-fit plane for the cluster of k points.
4 Use the plane normal v to determine the intersecting opposing surface.
5 From the intersected point, determine the best k nearest neighbours by Sk.
6 Construct a SVD best-fit plane plane for the cluster of k points.
7 Let the plane normal v intersect the first point p0.
8 Calculate the euclidean distances of the two normal vectors.
9 Select the shortest distance as the material thickness.
```

Listing 2.1: Material Thickness Two Way Search Algorithm.

The last hurdle to overcome is the fact that in a point cloud the plane normal is not guaranteed to intersect a point on the opposing surface. Here two methods can be implemented to determine the point closest to the theoretical intersected point.

The distance from any point p_n to the normal vector $\vec{v}$ can be found through (2.22)

$$dist = \frac{\sqrt{((p_n - p_0) \times \vec{v}) \cdot ((p_n - p_0) \times \vec{v})}}{\sqrt{\vec{v} \cdot \vec{v}}} \tag{2.22}$$

where p_0 is the initial point, "$\times$" is the cross product and "$\cdot$" is the dot product.

Similarly the angle between the normal vector $\vec{v}$ and a vector formed by the initial point p_0 and any point p_n is given by (2.23)

$$angle = \frac{\arccos(\vec{u_n} \cdot \vec{u}) * 180}{\pi}, \text{ where} \tag{2.23}$$

$$\vec{u} = \frac{\vec{v}}{\sqrt{\vec{v} \cdot \vec{v}}} \text{ and } \vec{u_n} = \frac{(p_n \vec{-} p_0)}{\sqrt{(p_n \vec{-} p_0) \cdot (p_n \vec{-} p_0)}}. \tag{2.24}$$

Both methods for finding the closest point p_n to the theoretical intersecting point have their advantages and disadvantages. Evaluating the distance to the normal means that points on either two planes can be selected as the closest point. Using this method thus means having to exclude at the very least the points k that made the original plane in order to find the correct intersection point. Depending on the point distribution this may be more or less than the k points, e.g. simple to calculate but hard to generalize. On the other hand the angle is periodical and dependant on the direction of the normal vector $\vec{v}$. Meaning that the minimum value of $abs(\arccos(\vec{u_n} \cdot \vec{u}))$ and $abs(\arccos(\vec{u_n} \cdot \vec{u}) - \pi)$ has to be selected in (2.23) in order to get the correct intersection point.

Implementing the material thickness algorithm from (Listing 2.1) combined with the approximation for the closest intersection point through the minimum angle search the method is finally decoupled from all the dependencies presented in (subsection 1.2.1).

In order to demonstrate the method, two new datasets A and B are introduced, both A and B are comprised of 3D points structured randomly across two opposing disks. Unique for B is that the bottom disk has a step function along the distribution of the disk. A and B are demonstrated in Figure 2.12

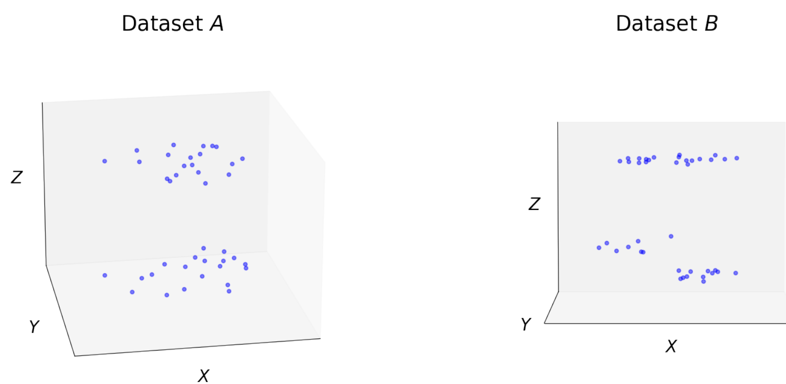


Figure 2.12: Dataset A and B Comprised of Randomly Distributed 3D Points Along Two Opposing Disks.

Applying the two way material thickness search algorithm from (Listing 2.1) on dataset A using a randomly selected point as the starting point p_0. The material thickness is initially evaluated to be 1.9922, whereas the second search reveals that there exist a better pairing which returns a material thickness of 1.9789 as demonstrated in Figure 2.13.

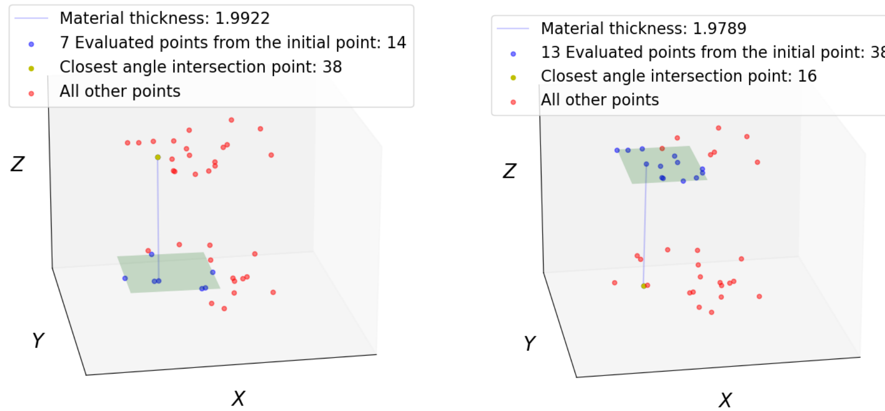


Figure 2.13: Material Thickness Two Way Search Demonstrated On Dataset A.

Likewise for dataset B a better approximation is found on the second search as shown in Figure 2.14.

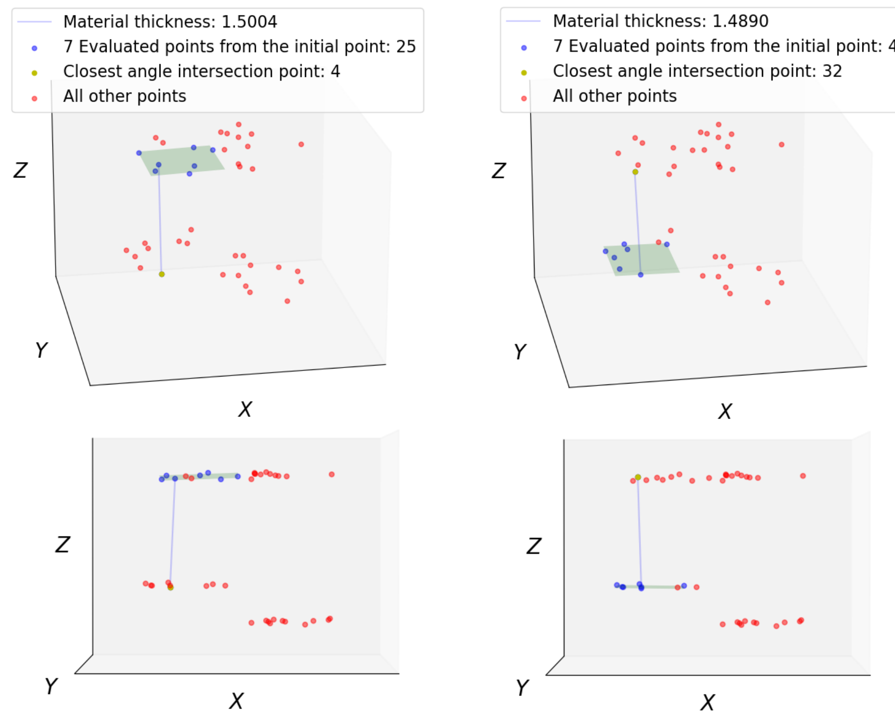


Figure 2.14: Material Thickness Two Way Search Demonstrated On Dataset B.

2.3 ATOS Professional

The software solution for the GOM measurement cell is ATOS Professional, a 3D inspection and metrology system designed for use in ATOS 3D scanning systems. The most notable functionality of the software in respect to this thesis is it's capability of being scripted through the programs own script functions. This allows for inspections and operations of the scanned component data to be carried out autonomously in accordance with the script. The scripts are written in the programming language Python and are capable of doing any and/or all operations the software normally is capable of.[3] While also having the possibility to introduce new functionality through programming.

2.3.1 ATOS Professional Scripts

To resolve the open point regarding autonomous digitization of components from AWA by Austefjord, R. 2020 [1], a script has to be developed in order to select the welding junctions of the components and calculate their material thickness.

To achieve a suitable selection of the welding junctions some prior setup is required, notably custom Computer Aided Design (CAD) models can be made for each unique component type. Within these CAD models the welding junctions can be classified as predefined areas that the software easily can recognize once the CAD model is imported. An example of this is shown in Figure 2.15 where the red areas are the welding junctions for that component type. These areas can then easily be evaluated by the programs built in material thickness function and exported for use and storage. The python code to achieve the automated material thickness evaluation is described in Appendix (A).

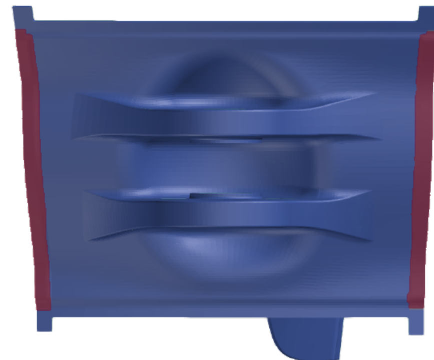


Figure 2.15: CAD Model Of A Mount-Strut With Welding Junctions Indicated In Red, Model In Courtesy Of GKN Trollhättan [18].

[3]Notable functions are described in the project work AWA by Austefjord, R. 2020 [1].

3 Results

As this thesis investigates the realm of 3D scanning as measurement system, 3D data structures, 3D data classification and evaluation as well as developing general frameworks and methods based upon known studies and methods. The results are divided into two sections in order to provide a clear overview of the work that has been done in the thesis, contrasted to how existing methods have been utilized. The division is structured by what is *known* and what *has been done*.

3.1 Investigational Results

In order to present general methods and frameworks for 3D scan data, four known data structures have been discussed on the terms of usability and functionality. Out of the four representations (*Image*, *Volumetric*, *Meshes* and *Point Clouds*) it has been determined through this thesis that for the goal of analyzing the datasets, the point cloud representation provide the best analytical base.

The 3D point cloud information has been investigated in order to determine the dependencies both with respects to the measurement systems as well as the originating object. The problems surrounding 3D scanning as a measurement system have been presented where a claim has been stated that in order for any particular applicable method to be truly general, it has to be capable of invariance towards *point density*, *orientation* and prior *empirical* or *heuristic knowledge*.

From the presented invariance claim, methods to remove the dependencies has been investigated in order to infer useful knowledge from the information within the datasets. Firstly a method for providing general data frameworks was considered in order to structure the otherwise unstructured and reference system dependant collection of 3D points. It has been shown through the thesis that any point cloud can be structured as a *graph* framework, making it simple to classify and structure each point by their nearest neighbouring points as *adjacency lists* through the known method of *KD-Tree* structuring.

From the graph structure further knowledge can be inferred such as classifying local point clusters by their dimensionalities through known methods for eigenvalue analysis. These clusters can be found invariant to point densities, orientation or location through a minimization of the ”uncertainty” entropy through the known *Shannon Entropy* method.

3.2 Development Results

Utilizing the methods discussed in *Investigational Results*, a general method for determining material thickness in unstructured 3D measurement data has been presented. Drawing inspiration from the human intuition and knowledge of what and how material thickness is determined, the method was developed from a point to point measurement of *Euclidean Distance* to a complex *Material Thickness Optimized Plane Search* algorithm. Importantly the final algorithm operates invariant towards *point density*, *orientation* and prior *empirical* or *heuristic knowledge*. However it is not infallible and have only been tested on simple fabricated 3D datasets which is discussed further in *Discussion*.

A method for autonomously determining the material thickness through the ATOS Professional software has been presented where the scanned 3D data is compared to CAD models with predefined segments of interest.

The methods have been implemented through code available in Appendix A.

4 Conclusion

For methods applied to 3D scan measurement data represented as *point clouds* to truly be general, they have to be invariant to variations in *point density*, *orientation* and prior *empirical* or *heuristic knowledge*.

A general *graph* framework can be built using *KD-tree* clustering. Local clusters can be classified by their surrounding nearest points to form different dimensionality features such as *Linearity* L_λ, *Planarity* L_λ and *Scattering* L_λ. These clusters can be defined from varying point densities through the optimization of the *Shannon Entropy* for the normalized dimensionality features expressed as ”pseudo probabilities”.

Material thickness for low curvature objects can be determined through evaluating the distance between two opposing planes formed by the local clusters discovered by the *Shannon Entropy* evaluation of the individual points. See the implemented algorithm in Listing A.4.

Material thickness can be autonomously determined through the ATOS Professional software by comparing 3D scan data with predefined CAD models. See the implemented script in Listing A.2.

Functions, methods and scripts have in general been implemented through Python code available in Appendix A.

5 Discussion

Due to the extremely broad goal of making a general solution applicable to any 3D point cloud dataset, the development had to be segmented into solving a subset of simpler/smaller problems rather than tackling everything at once. This means that for testing purposes the methods have been tested on appropriately challenging fabricated data in stead of real measurement data. There is a couple reasons for this, firstly it ensures that the method can be tested under a controllable environment. Secondly the results are intuitive and understandable compared to results from highly complex datasets. Maybe more importantly for the sake of development progress, inconsistencies and errors are simpler to handle because of the controllable and easier to understand test environment. Likewise due to the sheer size of the real datasets, the results from the tests can be produced in milliseconds as opposed to minutes or hours.

There has been done an effort in collecting real measurement data for three commonly used reference objects used in the aerospace manufacturing industry, a *sphere*, a *cylinder* and a *slab*. These reference objects have known measurable properties such as diameter, length, width, depth, angle and run-out. However it has been known for quite some time that highly curved objects would pose at least in my perceivable theory, unsolvable problems for a general method. The reason is paradoxical in nature, taking a sphere as an example, the material thickness of a sphere is it's diameter which is the longest distance between two points on the sphere. This breaks the previously assumed notion that the material thickness is the shortest distance between two opposing surfaces. For these reasons both the spherical and cylindrical dataset was deemed unfit to evaluate the current model. The "square-ish" slab could be a good candidate to test the current model, however it was never tested because it's too large (43M 3D points). Reducing the size almost exactly equates to the same problem of the fabricated simple 3D dataset otherwise used, so this would be a needlessly complex operation for little benefit. Regardless the presented method will most assuredly fail for spherical or highly curved objects.

The two way material thickness search algorithm is somewhat clumsy and not the best solution. A better method for determining the closest point would be to let a sphere expand from the initial point p_0 until it hits another point p_n. This would ensure that the most optimal point would be selected every time. However since the sphere expands in all directions, the first point encountered would very likely be the neighbouring point on the same surface. This means that the method would have to know how to differentiate

between points along the same surface and ”other points of interest”. A solution to this problem should be possible and is noted in *Further Work*.

In the presented general material thickness method, SVD is used to best-fit a plane to the point cluster. Notably this might be needlessly complex since the only value extracted from the SVD is the least significant right singular value. This could likewise be done by selecting the least significant eigenvector in e.g. a PCA decomposition.

λ_{2D} which is utilized in (subsection 2.2.4) could be expressed more accurately as (5.1)

$$\lambda_{2D} = \frac{e_3}{e_1 + e_2} \approx \frac{e_3 + \varepsilon}{e_1 + e_2 + \varepsilon} \approx 0 \tag{5.1}$$

though this formulation can't be used in the Shannon Entropy equation. This is because the evaluated probabilities has to sum up to 1, this is of course achievable by for example including the ”*not* λ_{2D}” such that (5.2)

$$\frac{e_3}{e_1 + e_2} + \frac{e_1 + e_2 - e_3}{e_1 + e_2} = 1 \ . \tag{5.2}$$

However it is difficult to express what scenario ”*not* λ_{2D}” refers to or in other words it does a poor job of describing the dimensionality present if the data is not *planar dimensional* λ_{2D}. It is also unknown if the natural tendency within a dataset is to have a lower value for ”*not* λ_{2D}” than λ_{2D} or vice versa. Notably $\frac{e_3}{e_1} \approx \frac{e_3}{e_1+e_2}$ because $e_1 \geq e_2$ and $e_1 \gg e_3$ meaning this simplification introduces very little error.

5.1 Further Work

From the work done in the thesis many new questions have arisen which could prove insightful in order to improve upon the presented solutions.

The broad field of *graph theory* contain many elegant and computational cost efficient algorithms. Many of the problems attempted solved in this thesis might be possible to rephrase into problems solvable by such algorithms. Thus a study into the capabilities of *graph theory* may improve the methods substantially.

Due to a design choice in the implementation process of the material thickness algorithm, the *Shannon* entropy is optimized using a brute force approach, ranging from $5 \rightarrow 40$. This is done because other optimizers such as *Newton-Cotes* or the *Bernt's Algorithm* do not allow for integer solutions. Such algorithms would, and in fact does produce errors if they do not get to investigate the numerical realm between two whole numbers. However it makes no sense to look at 6 and a half nearby points, meaning the algorithm gets stuck at the first local minima it finds. The brute function works decently but is not truly invariant to point density changes since the roof of the search is capped. It is also slow

since it has to evaluate 35 points every time the function is called. Here a better optimizer should be implemented.

Similarly when the material thickness algorithm searches for the smallest angle difference to the ideal "p_n intersection", brute force is applied as the optimizer. There seems to be no good way of improving such a search which means that currently the algorithm has to evaluate every point in the dataset, twice because the algorithm is repeated when finding the second material thickness back towards the initial point p_0. Here there might be vastly better ways of doing such a search if the problem could be formulated as a graph problem.

SVD is in this thesis only used to find the least significant right singular value in the local cluster. It should be worth investigating whether more useful insight might be derived from other parts of the decomposition. SVD might also be useful for analysing the whole dataset or groups of clusters, reducing the structure or determining the most important features (highest rank sigmas), etc...

For highly curved objects it could be a posibility to best-fit a sphere and use the tangent plane intersecting the initial point p_0 to determine the normal direction of the material thickness. For complete spheres datasets, best-fit spheres could also give the radius, e.g. material thickness directly from the formula.

It could be worth investigating methods to triangulate the dataset and subsequently evaluating the triangles. It was noted while reviewing mesh Grid structures in dialogue with Associate Proff. Håkon Viumdal and Proff. Ola Marius Lysaker that in planar areas, the triangles usually had similar angles between the edges. In areas with high curvature or close to edges there would be considerably more sharp angles, where one angle would be very small compared to the other two. There then might be an analytical benefit from investigating the triangles angles and their edge lengths in order to see if any useful information could be extrapolated.

Bibliography

[1] R. Austefjord, *Adaptive Welding Automation based on 3D Point Clouds*, Project, 2020.

[2] (1993). 'Stanford 3d scanning repository,' [Online]. Available: `http://graphics.stanford.edu/data/3Dscanrep/`.

[3] C. R. Qi, H. Su, K. Mo and L. J. Guibas, *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*, Presentation, PDF, 2017. [Online]. Available: `https://web.stanford.edu/~rqi/pointnet/docs/cvpr17_pointnet_slides.pdf`.

[4] H. Su, E. Maji S. Kalogerakis and E. Learned-Miller, *Multi-view Convolutional Neural Networks for 3D Shape Recognition*, Paper, 2015.

[5] T. Zhou, M. Brown, N. Snavely and D. Lowe, *Unsupervised Learning of Depth and Ego-Motion from Video*, Paper, 2017.

[6] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan and L. J. Guibas, *Volumetric and Multi-View CNNs for Object Classification on 3D Data*, Paper, 2016.

[7] J. H. Friedman, J. L. Bentley and R. A. Finkel, *An Algorithm for Finding Best Matches in Logarithmic Expected Time*, Paper, 1977.

[8] S. Maneewongvatana and D. M. Mount, *It's okay to be skinny, if your friends are fat*, Paper, 1999.

[9] A. H. Barr, *Superquadrics and Anglepreserving Transformations*, Paper, 1981.

[10] L. Zhu, A. Kukko, J. Virtanen, J. Hyyppä, H. Kaartinen, H. Hyyppä and T. Turppa, *Multisource Point Clouds, Point Simplification and Surface Reconstruction*, Paper, 2019.

[11] Y. Jiang, C. Li, F. Takeda, E. A. Kramer, H. Ashrafi and J. Hunter, *3d point cloud data to quantitatively characterize size and shape of shrub crops*, Paper, 2019.

[12] M. Weinmann, B. Jutzi, S. Hinz and C. Mallet, *Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers*, Paper, 2015.

[13] ——, *Semantic 3d Scene Interpretation: A Framework Combining Optimal Neighborhood Size Selection With Relevant Features.* Paper, 2014.

[14] C. E. Shannon, *A Mathematical Theory of Communication*, Paper, 1948.

[15] (2021). 'Ordinary Least Squares,' [Online]. Available: https://en.wikipedia.org/wiki/Ordinary_least_squares.

[16] (2021). 'Total Least Squares,' [Online]. Available: https://en.wikipedia.org/wiki/Total_least_squares.

[17] (2021). '3D SVD,' [Online]. Available: https://math.stackexchange.com/questions/2810048/plane-fitting-using-svd-normal-vector.

[18] *Models developed by and subject to copyright at GKN Trollhättan*, Intellectual Property, Flygmotorvägen 1, 461 38 Trollhättan.

[19] (2020). 'Halftones,' [Online]. Available: https://github.com/philgyford/python-halftone.

Appendix A

Software Source Code

Below the software developed through the master thesis are presented. The code is documented inline and with a short description preceding the code listing.

A.1 GOM Inspect Suite Script

The following script in (Listing A.1) may be used in the ATOS Professional GOM Inspect Suite in order to extract both the scanned point cloud and it's triangulated mesh grid from an element named "Name of Element" (exchangeable in the code).

```python
import numpy as np
import gom

points = np.array(gom.app.project.actual_elements[
                       'Name of Element'].data.coordinate)
tri = np.array(gom.app.project.actual_elements[
                         'Name of Element'].data.triangle)
print(points)
print("The shape of the points array: {}\nThe size of the "
      "points array: {}".format(np.shape(points), np.size(points)))

print(tri)
print("The shape of the triangle array: {}\nThe size of the "
      "triangle array: {}".format(np.shape(tri), np.size(tri)))
```

Listing A.1: GOM Point Collection

A.2 GOM Inspect Suite Automation Script

The following script may be used to open, establish and extract the material thickness across any predefined area by location or name through the "gom.script.selection3d.select_patch" function.

```
import gom
import numpy as np


"""Import cad model and rename"""
gom.script.sys.import_stl (
  bgr_coding=True,
  body_split='no_color',
  files=['C:/Users/Ruben/School/IIA Master/03_02 Master Thesis/ATOS
   Professional/Mounts/VAN_Y00001035_07.stl'],
  geometry_based_refining=False,
  import_mode='clipboard',
  length_unit='mm',
  stl_color_bit_set=False,
  target_type='cad_body')

gom.script.part.create_new_part (name='Mount-Strut-CAD')

gom.script.part.add_elements_to_part (
  delete_invisible_elements=True,
  elements=[gom.app.project.clipboard.nominal_elements['VAN_Y00001035_07'
   ]],
  import_mode='new_elements',
  part=gom.app.project.parts['Mount-Strut-CAD'])


"""Import Mesh and rename"""

gom.script.sys.import_stl (
  bgr_coding=True,
  files=['C:/Users/Ruben/School/IIA Master/03_02 Master Thesis/ATOS
   Professional/Scans/31860513_GUK905HA.stl'],
  geometry_based_refining=False,
  import_mode='clipboard',
  length_unit='mm',
  stl_color_bit_set=False,
  target_type='mesh')

gom.script.part.create_new_part (name='Mount-Strut-Mesh')

gom.script.part.add_elements_to_part (
  delete_invisible_elements=True,
  elements=[gom.app.project.clipboard.actual_elements['31860513_GUK905HA'
   ]],
```

```
41   import_mode='new_elements',
42   part=gom.app.project.parts['Mount-Strut-Mesh'])
43 
44 
45 
46 """Prealign"""
47 CAD_ALIGNMENT=gom.script.alignment.create_prealignment (
48   computation_mode='normal',
49   compute_additional_bestfit=True,
50   name_expression='Prealignment',
51   parent_alignment=gom.app.project.parts['Mount-Strut-Mesh'].
      original_alignment,
52   part=gom.app.project.parts['Mount-Strut-Mesh'],
53   target_element=gom.app.project.parts['Mount-Strut-CAD'].nominal)
54 
55 """Set visability of Mesh off"""
56 gom.script.cad.hide_element (elements=[gom.app.project.parts['Mount-Strut-
      CAD'].actual])
57 
58 """Select welding junction"""
59 
60 
61 
62 """Mimic select CAD feature
63 gom.script.selection3d.select_patch (
64   coordinate=gom.Vec3d (7527.677051, 803.1583909, 178.786535),
65   target=gom.app.project.parts['Mount-Strut-CAD'].nominal)
66 """
67 
68 
69 """Create material thickness"""
70 MCAD_ELEMENT=gom.script.comparison.create_multiple_material_thickness (
71   max_angle_between_normals=0.436332313,
72   max_opening_angle=1.047197551,
73   max_thickness=6.0,
74   min_thickness=1.0,
75   name='Material Thickness Welding Junction Right 1')
76 
77 Material_thickness = np.array(gom.app.project.actual_elements['Material
      Thickness Welding Junction Right 1'].data.coordinate)
78 
79 print(Material_thickness)
80 print(Material_thickness.shape)
81 print(Material_thickness.size)
```

Listing A.2: GOM Automation

A.3 Methods

From the theoretical methods used throughout the thesis the vast majority has been implemented through Python code in the following (Listing A.3). The code has been developed in Python with some added features for ease of visualization. The code is presented piece-wise as functions for solving the different problems faced in the methods. The functions have been commented with intended input and output such that they may be used separately from any one method. An example of use is given in (Listing A.4).

```
import numpy as np
from scipy.spatial import KDTree
from scipy.optimize import brute


def generate_points(xx, yy, zz):
    """
    Function for creating a 3D-box of evenly distributed points.
    :param xx: the desired z dimensionality.
    :param yy: the desired y dimensionality.
    :param zz: the desired x dimensionality.
    :return: coordinates: a numpy array containing the 3D coordinates
    for the points.
    """
    coordinates = []
    for z in range(zz):
        for y in range(yy):
            for x in range(xx):
                coordinates.append("{} {} {}".format(x, y, z))
    coordinates = [w.split(' ') for w in coordinates]
    coordinates = np.asarray(coordinates).astype(np.float)
    coordinates = coordinates + ((np.asarray(np.random.rand(18, 3)) -
                                  0.5) * 0.05)
    coordinates = coordinates * [1, 1, 2]
    return coordinates


def generate_points_disk(number, radius, offset):
    """
    Function for generating randomly distributed 3D points along a disk.
    :param number: The desired number of points.
    :param radius: The radius of the disk.
    :param offset: The offset in the z direction.
    :return: A numpy array containing the 3D coordinates for the points.
    """
    r = np.random.uniform(low=0, high=radius, size=number)
    theta = np.random.uniform(low=0, high=2 * np.pi, size=number)
    noise = np.random.uniform(low=-0.05, high=0.05, size=number)

    x = np.sqrt(r) * np.cos(theta)
```

```
41      y = np.sqrt(r) * np.sin(theta)
42      z = ([offset] * number) + noise
43      return np.asarray(list(zip(x, y, z)))
44 
45 
46  def generate_plane_svd(points):
47      """
48      Function for fitting a plane to a set of points by the Singular
49      Value Decomposition method.
50      :param points: A collection of points (x,y,z) that should be
51      considered to find the best-fit plane.
52      :return: x, y, z and normal: Returns the plane function z bounded
53      by the max/min values of x and y structured in a meshgrid(xx, yy)
54      for graphing. Returns the normal to the plane.
55      """
56      # defining the domain of the plane
57      max_x = np.max(points[:, 0])
58      max_y = np.max(points[:, 1])
59      min_x = np.min(points[:, 0])
60      min_y = np.min(points[:, 1])
61      x, y = np.meshgrid([min_x, max_x], [min_y, max_y])
62      # Note that the meshgrid is bounded by the x and y values,
63      # meaning that for planes orthogonal to the xy-plane this will
64      # produce a very bad visualization.
65 
66      # centering the data.
67      centroid = points.mean(axis=0)
68      points = points - centroid
69      u, sigma, v = np.linalg.svd(points)
70      normal = v[2]
71      d = normal[0] * centroid[0] + normal[1] * centroid[1] + normal[2] \
72          * centroid[2]
73 
74      z = (-normal[0] * x - normal[1] * y + d) * 1. / normal[2]
75      return x, y, z, normal
76      # Reference:
77      # https://stackoverflow.com/questions/53591350/plane-fit-of-3d-points-
       with-singular-value-decomposition
78      # https://math.stackexchange.com/questions/2810048/plane-fitting-using-
       svd-normal-vector
79 
80 
81  def generate_plane_lstsq(points):
82      """
83      Function for fitting a plane to a set of points by the least
84      squares method.
85      :param points: A collection of points (x,y,z) that should be
86      considered to find the best-fit plane.
87      :return: x, y, z and normal: Returns the plane function z
88      bounded  by the max/min values of x and y structured in a
89      meshgrid(xx, yy) for graphing. Returns the normal to the plane.
```

```
90      """
91      matrix = np.column_stack((points[:, 0], points[:, 1],
92                                np.ones(points[:, 0].size)))
93      (a, b, c), residual, rank, s = \
94          np.linalg.lstsq(matrix, points[:, 2], rcond=None)
95      normal = unit_vector(np.asarray([a, b, -1]))
96      # defining the domain of the plane
97      max_x = np.max(points[:, 0])
98      max_y = np.max(points[:, 1])
99      min_x = np.min(points[:, 0])
100     min_y = np.min(points[:, 1])
101     # creating a point in the plane
102     point = np.array([0.0, 0.0, c])
103     d = -point.dot(normal)
104     x, y = np.meshgrid([min_x, max_x], [min_y, max_y])
105     z = (-normal[0] * x - normal[1] * y - d) * 1. / normal[2]
106     # Comment regarding non-linear plane:
107     # https://gist.github.com/amroamroamro/1db8d69b4b65e8bc66a6
108     return x, y, z, normal
109 
110 
111 def unit_vector(vector):
112     """
113     Function for finding the unit vector of a vector.
114     :param vector: The original vector.
115     :return: The unit vector of the vector.
116     """
117     return vector / np.sqrt(vector.dot(vector))
118 
119 
120 def vector_3d(p1, p2):
121     """
122     Function for substracting one point/vector from another.
123     :param p1: (x_a, y_a, z_a)
124     :param p2: (x_b, y_b, z_b)
125     :return: ((x_b - x_a), (y_b - y_a), (z_b - z_a))
126     """
127     return p2 - p1
128 
129 
130 def euclidean_distance(p1, p2):
131     """
132     Function for calculating the euclidean distance between two points.
133     :param p1: Point A
134     :param p2: Point B
135     :return: The euclidean distance between point A and B.
136     """
137     vector = vector_3d(p1, p2)
138     return np.sqrt(vector.dot(vector))
139 
140 
```

```
141 def angle_between_vectors(vector1, vector2):
142     """
143     Function for finding the angle between two vectors by applying
144     the common formula : angle = arccos(vector1 dot vector2 /
145     magnitude(vector1) * magnitude(vector2)).
146     :param vector1: The first vector.
147     :param vector2: The second vector.
148     :return: The radian and angle between the two vectors.
149     """
150     v1 = unit_vector(vector1)
151     v2 = unit_vector(vector2)
152 
153     radians = np.arccos(np.clip(np.dot(v1, v2), -1.0, 1.0))
154     angle = radians * 180 / np.pi
155 
156     return radians, angle
157 
158 
159 def brute_optimal_angle(data_set, point, normal):
160     """
161     Function for determining the optimal angle based on brute force
162     optimization.
163     :param data_set: Data of (x,y,z) values / points to be evaluated.
164     :param point: Point of origin.
165     :param normal: The normal direction the points should be
166     evaluated  against.
167     :return: The point in data_set that lie closest in terms of
168     angle  to the origin point.
169     """
170     def radians(n):
171         v1 = unit_vector(vector_3d(point, data_set[int(n)]))
172         v2 = unit_vector(normal)
173         radian = min(np.arccos(np.clip(np.dot(v1, v2), -1.0, 1.0)),
174         abs(abs(np.arccos(np.clip(np.dot(v1, v2), -1.0, 1.0))) - np.pi))
175         return radian
176 
177     result = brute(radians, (slice(1, data_set.shape[0], 1),))
178     return data_set[int(result[0])]
179 
180 
181 def brute_optimal_distance(data_set, point, normal):
182     """
183         Function for determining the optimal distance based on brute
184         force optimization.
185         :param data_set: Data of (x,y,z) values / points to be evaluated.
186         :param point: Point of origin.
187         :param normal: The normal direction the points should be
188         evaluated against.
189         :return: The point in data_set that lie closest in terms of
190         distance to the normal vector.
191         """
```

```
192     def dist(n):
193         cross_product = np.cross((data_set[int(n)] - point), normal)
194         distance = np.sqrt(cross_product.dot(cross_product)) /  \
195                    np.sqrt(normal.dot(normal))
196         return distance
197 
198     result = brute(dist, (slice(1, data_set.shape[0], 1),))
199     return data_set[int(result[0])]
200 
201 
202 def flatten_2d(data):
203     """
204     Function for reducing a numpy array down to two dimensions.
205     :param data: A numpy array of n>2 dimensions.
206     :return: A numpy array of n <= 2 dimensions.
207     """
208     return data.reshape(-1, data.shape[-1])
209 
210 
211 def pca(points):
212     """
213     Function for Principle Component construction for further analysis.
214     :param points: An array of points to be evaluated.
215     :return: The Eigenvalues and Eigenvectors of the data. The
216     centered data, the mean and the covariance matrix.
217     """
218     mean = np.mean(points, axis=0)
219     data = (points - mean)
220     matrix = np.cov(data.T)
221     eigenvalues, eigenvectors = np.linalg.eig(matrix)
222     return eigenvalues, eigenvectors, data, mean, matrix
223 
224 
225 def kd_tree(data):
226     """
227     Function for creating a KD-tree for the given data.
228     :param data: The data to be structured in a KD-tree structure.
229     :return: The data structured in a KD-tree.
230     """
231     return KDTree(data)
232 
233 
234 def neighborhood_selection(tree, data_set, initial_point):
235     """
236     Function for finding the best neighbourhood point cluster selection.
237     :param tree: KD-tree of the data
238     :param data_set: The dataset to be evaluated, this is the same
239     dataset as the KD-tree is structured by.
240     :param initial_point: The initial point where the neighbourhood
241     should be optimized around.
242     :return: S_k, the optimal number of k nearest neighbours.
```

```
243     """
244     def shannon_entropy(points):
245         epsilon = np.finfo(float).eps
246         cov_matrix = np.cov(points.T)
247         eigen_value, eigen_vector = np.linalg.eig(cov_matrix)
248         idx = eigen_value.argsort()[::-1]
249         eigen_value = eigen_value[idx]
250         sum = eigen_value[0] + eigen_value[1] + eigen_value[2]
251         eigen_value = eigen_value / sum
252         linearity = (eigen_value[0] - eigen_value[1]) / (
253                 eigen_value[0]  + epsilon)
254         planarity = (eigen_value[1] - eigen_value[2]) / (
255                 eigen_value[0]  + epsilon)
256         scattering = eigen_value[2] / (eigen_value[0] + epsilon)
257         shannonentropy = (- linearity * np.log(linearity + epsilon)
258                           - planarity * np.log(planarity + epsilon)
259                           - scattering * np.log(scattering + epsilon))
260         return shannonentropy
261
262     def query(n):
263         distances, nearest_neighbours = \
264             tree.query([data_set[initial_point]], k=int(n))
265         return shannon_entropy(flatten_2d(data_set[nearest_neighbours]))
266
267     result = brute(query, (slice(5+1, 18+1, 1),))
268     # optimizes the minimum local solution within the given bounds
269     # using brute force.
270     # result = minimize_scalar(query, bounds=(5, 10), method='bounded')
271     # optimizes the minimum local solution within the given bounds
272     # using Bernt's Algorithm.
273     return result[0]
274
275
276 def general_shannon_entropy(points):
277     """
278     Function for evaluating a collection of points (x,y,z) in terms
279     of local features and entropy.
280     :param points: A cluster of 3D coordinates (x,y,z)
281     :return: The Covariance Matrix, Eigenvalues, Eigenvectors,
282     Linearity, Planarity, Scattering, Omnivariance, Anisotropy,
283     Curvature, Sum Eigenvalues and the Eigenentropy.
284     """
285     epsilon = np.finfo(float).eps
286     cov_matrix = np.cov(points.T)
287     eigen_value, eigen_vector = np.linalg.eig(cov_matrix)
288     idx = eigen_value.argsort()[::-1]
289     eigen_value = eigen_value[idx]
290     eigen_vector = eigen_vector[:, idx]
291     sum = eigen_value[0] + eigen_value[1] + eigen_value[2]
292     eigen_value = eigen_value/sum
293     linearity = (eigen_value[0] - eigen_value[1]) / (eigen_value[0]
```

```
294                                                         + epsilon)
295     planarity = (eigen_value[1] - eigen_value[2]) / (eigen_value[0]
296                                                         + epsilon)
297     scattering = eigen_value[2] / (eigen_value[0] + epsilon)
298     omnivariance = abs(eigen_value[0] * eigen_value[1] *
299                        eigen_value[2])**(1./3.)
300     anisotropy = (eigen_value[0] - eigen_value[2]) / (eigen_value[0]
301                                                         + epsilon)
302     curvature = eigen_value[2]/(eigen_value[0] + eigen_value[1]
303                                 + eigen_value[2] + epsilon)
304     sum_eigenvalues = eigen_value[0] + eigen_value[1] + eigen_value[2]
305     eigenentropy = (- eigen_value[0] * np.log(eigen_value[0] + epsilon)
306                     - eigen_value[1] * np.log(eigen_value[1] + epsilon)
307                     - eigen_value[2] * np.log(eigen_value[2] + epsilon))
308     shannonentropy = (  - linearity * np.log(linearity + epsilon)
309                         - planarity * np.log(planarity + epsilon)
310                         - scattering * np.log(scattering + epsilon))
311 
312     return cov_matrix, eigen_value, eigen_vector, linearity, \
313            planarity, scattering, omnivariance, anisotropy,  \
314            curvature, sum_eigenvalues, eigenentropy, shannonentropy
```

Listing A.3: Methods implemented in code

```
1  import Methods
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from mpl_toolkits.mplot3d import Axes3D
5  
6  
7  def material_thickness_example(initial_point=10):
8      """
9      Function for demonstrating the use of different functions in
10     Methods.
11     :param initial_point: The initial point, default is 10.
12     """
13     lower_bound = Methods.generate_points_disk(20, 1, 0)
14     upper_bound = Methods.generate_points_disk(20, 1, 2)
15     all_points = np.concatenate((lower_bound, upper_bound), axis=0)
16     general_kd_tree = Methods.kd_tree(all_points)
17 
18     number_of_nearest_neighbours = int(Methods.neighborhood_selection(
19         general_kd_tree, all_points, initial_point))
20     distances, nearest_neighbours_index = general_kd_tree.query(
21         [all_points[initial_point]], k=number_of_nearest_neighbours)
22     nearest_neighbours_points = Methods.flatten_2d(
23         all_points[nearest_neighbours_index])
24     # *least_squares_plane, least_squares_normal =
25     # Methods.generate_plane_lstsq(nearest_neighbours_points)
26     # *least_squares_plane, least_squares_normal =
27     # Methods.generate_plane_pca(nearest_neighbours_points)
28     *least_squares_plane, least_squares_normal = \
```

```
29          Methods.generate_plane_svd(nearest_neighbours_points)
30      reduced_points = np.delete(all_points, [nearest_neighbours_index], 0)
31      smallest_angle_point = Methods.brute_optimal_angle(
32          reduced_points, all_points[initial_point], least_squares_normal)
33      euclidean_distance = Methods.euclidean_distance(
34          all_points[initial_point], smallest_angle_point)
35  
36      plt3d = plt.figure(figsize=(8, 8)).gca(projection='3d')
37      plt3d.set_xlim3d(-1.5, 1.5)
38      plt3d.set_ylim3d(-1.5, 1.5)
39      plt3d.set_zlim3d(-0.5, 2.5)
40      plt3d.set_xlabel('$X$', fontsize=20)
41      plt3d.set_ylabel('$Y$', fontsize=20)
42      plt3d.set_zlabel('$Z$', fontsize=20)
43      plt3d.set_xticks([])
44      plt3d.set_yticks([])
45      plt3d.set_zticks([])
46      plt3d.view_init(elev=20, azim=80)
47      plt3d.scatter(*nearest_neighbours_points.T, c='b', marker='o',
48      alpha=.5, label='{} Evaluated points from the initial point: '
49                      '{}'.format(number_of_nearest_neighbours,
50                                      initial_point))
51      plt3d.plot_surface(*least_squares_plane, color='g', alpha=.2,
52                         linewidth=0, zorder=1)
53      plt3d.scatter(*smallest_angle_point.T, color='b')
54      plt3d.scatter(*smallest_angle_point.T, color='y', label=
55      'Closest angle intersection point: {}'.format(
56          np.where(all_points == smallest_angle_point)[0][0]))
57      matrix = np.asarray(list(zip(
58          all_points[initial_point],  smallest_angle_point)))
59      plt3d.plot3D(*matrix, color='b', alpha=.2, label=
60      'Material thickness: {0:.4f}'.format(euclidean_distance))
61      dataset_removed = np.delete(reduced_points,
62      [np.where(reduced_points == smallest_angle_point)[0][0],
63       np.where(reduced_points == smallest_angle_point)[0][0]], 0)
64      plt3d.scatter(*dataset_removed.T, color='r', alpha=.5, label=
65      'All other points')
66      plt.legend(prop={"size": 16})
67      plt.show()
68  
69  
70  if __name__ == '__main__':
71      material_thickness_example()
```

Listing A.4: Material thickness example use of Methods

A.4 Cover Image

For the cover image I wanted to create a visible connection between my previous work related to the manufacturing of TRF's to the more general research on point clouds. To achieve this I wanted to repurpose the original grid-line TRF view as a reminder of where the work originated and gradually morph the image into a collection of points to indicate and introduce the current work. The shift goes from right to left in adherence to natural reading patterns generally used in "western civilisations" since this has given us a basic understanding/interpretation of progress forward in time going from right to left. As opposed to some "eastern civilisations" where the reading pattern is left to right and the opposite assumption holds.

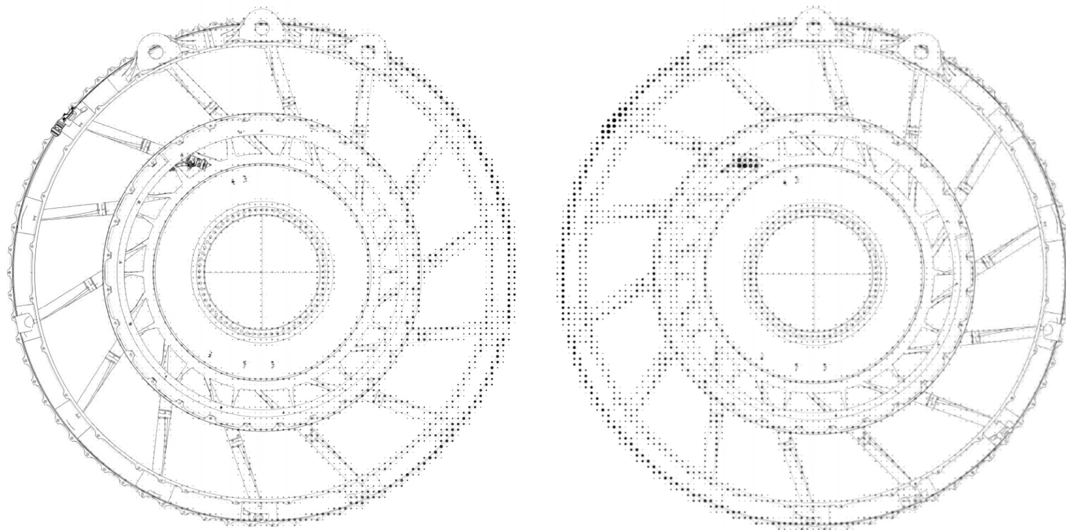


Figure A.1: Comparison Of Gradients From Left To Right / Right To Left.

There was two steps in creating this image, firstly the base image had to be represented as a collection of points. This was achieved through image processing in Python, mainly based upon the library composed by Phil Gyford [19].

A very quick and dirty introduction of how the code works is that you section the image into a fixed size grid of $n$by$m$ size. Then you evaluate the content of each "box" within the grid and from that determine an appropriate size and colour of a circle that you will fill within that box. Generally speaking, if the original box was entirely yellow, it would result in a large yellow circle, if it was half yellow and half blue the circle would be green and if it was half yellow and half white, the circle would be yellow with a smaller radius.

The result is then an increase in contrast and reduction in resolution since darker colors are exaggerated and more empty space is introduced.

To achieve the result I had in mind I had to make some improvements mainly on how the content of the boxes was evaluated, how the circle color and radius was calculated and whether the background should be black filled with white circles or the background should be white filled with black circles.

The Python code for the halftone is listed in Listing A.5.

```
import os
import sys
import numpy as np
from PIL import Image, ImageDraw, ImageStat


class Halftone(object):
    def __init__(self, path):
        """
        path is the path to the image we want to halftone.
        """
        self.path = path

    def make(self, sample=10, scale=1, filename_addition="_halftoned",
             angles=[0, 15, 30, 45]):
        """
        Leave filename_addition empty to save the image in place.
        Arguments:
            sample: Sample box size from original image, in pixels.
            scale: Max output dot diameter is sample * scale
            (which is also the number of possible dot sizes)
            filename_addition: What to add to the filename
            (before the extension).
            angles: A list of 4 angles that each screen channel
            should be rotated by.
        """
        f, e = os.path.splitext(self.path)
        outfile = "%s%s%s" % (f, filename_addition, e)

        try:
            im = Image.open(self.path)
        except IOError:
            raise

        angles = angles[:1]
        gray_im = im.convert("L")

        dots = self.halftone(im, gray_im, sample, scale,
                             angles)
        new = dots[0]
        new.save(outfile)

    def halftone(self, im, cmyk, sample, scale, angles):
        """
```

```
45          Returns list of half-tone images for cmyk image.
46          sample (pixels), determines the sample box size from the
47          original image. The maximum output dot diameter is given by
48          sample * scale (which is also the numberof possible dot sizes).
49          So sample=1 will presevere the original image resolution,
50          but scale must be >1 to allow variation in dot size.
51          """
52
53          cmyk = cmyk.split()
54          dots = []
55
56          for channel, angle in zip(cmyk, angles):
57              channel = channel.rotate(angle, expand=1)
58              size = channel.size[0] * scale, channel.size[
59                  1] * scale
60              half_tone = Image.new("L", size, color=255)
61              draw = ImageDraw.Draw(half_tone)
62
63              # Cycle through one sample point at a time, drawing a
64              # circle for each one:
65              for x in range(0, channel.size[0], sample):
66                  for y in range(0, channel.size[1], sample):
67                      # Area we sample to get the level:
68                      box = channel.crop(
69                          (x, y, x + sample, y + sample))
70
71                      # The average level for that box (0-255):
72                      mean = ImageStat.Stat(box).mean[0]
73
74                      # The diameter of the circle to draw based on the
75                      # mean (0-1):
76                      # diameter = (mean / 255) ** 0.5
77                      diameter = np.sqrt((255 - mean) / 255)
78                      # Size of the box we'll draw the circle in:
79                      box_size = sample * scale
80
81                      # Diameter of circle we'll draw:
82                      # If sample=10 and scale=1 then this is (0-10)
83                      draw_diameter = diameter * box_size
84
85                      # Position of top-left of box we'll draw the
86                      # circle in:
87                      # x_pos, y_pos = (x * scale), (y * scale)
88                      box_x, box_y = (x * scale), (y * scale)
89
90                      # Positioned of top-left and bottom-right of
91                      # circle:
92                      # A maximum-sized circle will have its edges at
93                      # the edges of the draw box.
94                      x1 = box_x + ((box_size - draw_diameter) / 2)
95                      y1 = box_y + ((box_size - draw_diameter) / 2)
```

```
                    x2 = x1 + draw_diameter
                    y2 = y1 + draw_diameter
                    draw.ellipse([(x1, y1), (x2, y2)], fill=0)

            half_tone = half_tone.rotate(-angle, expand=1)
            width_half, height_half = half_tone.size

            # Top-left and bottom-right of the image to crop to:
            xx1 = (width_half - im.size[0] * scale) / 2
            yy1 = (height_half - im.size[1] * scale) / 2
            xx2 = xx1 + im.size[0] * scale
            yy2 = yy1 + im.size[1] * scale

            half_tone = half_tone.crop((xx1, yy1, xx2, yy2))

            dots.append(half_tone)
        return dots


if __name__ == "__main__":
    path = sys.argv[1]
    h = Halftone(path)
    h.make()
```

Listing A.5: Gray-scale Halftone.

Now that a dotted representation of the image has been made, we can blend the original image and the halftone image to create the gradual shifting image. This can be done either mathematically through e.g. a Sigmoid function or by emulate the intensity of a premade gradual shift image as seen in Figure A.2. The Python code for the blend is listed in Listing A.6.

Figure A.2: Gradient Used In Blend Function.

```
import numpy as np
from PIL import Image

background = np.array(Image.open('path\to\image'))
overlay = Image.open('path\to\image').convert("RGB")
overlay = overlay.resize(background.shape[1::-1], Image.BILINEAR)
overlay = np.array(overlay)
mask = Image.open('path\to\image')
mask = mask.resize(background.shape[1::-1], Image.BILINEAR)
```

```
10 mask = np.array(mask)
11 mask = mask / 255
12
13 dst = background * (1 - mask) + overlay * mask
14 Image.fromarray(dst.astype(np.uint8)).save('path\to\save "name".jpg')
```

Listing A.6: Image Blending.