

Haakon Gunnarsli & Jonathan Brooks

Comparative Analysis of Object Recognition Techniques Using 3D Shape Descriptors

Master's thesis in Informatics

Supervisor: Bart Iver van Blokland

Co-supervisor: Theoharis Theoharis

June 2023

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



Haakon Gunnarsli & Jonathan Brooks

Comparative Analysis of Object Recognition Techniques Using 3D Shape Descriptors

Master's thesis in Informatics
Supervisor: Bart Iver van Blokland
Co-supervisor: Theoharis Theoharis
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



ABSTRACT

In this thesis the extent to which shape descriptors' object recognition abilities are affected by simulating real-world scenarios is tested. We present a benchmark consisting of three processes - Data Collection, Performance Data Generation, and Data Analysis. This thesis aimed to answer the lack of research on what leads to a good or bad matching performance for a given shape descriptor, and the absence of large and varied datasets used in shape descriptor tests. By using $n=1193$ objects gathered from the Google Research dataset, a series of transformations were generated, and through the benchmark the shape descriptors' performance in different environments were tested and visualised. The performance of a shape descriptor was tested through our newly proposed method, which involves comparing the results to a previously calculated poor matching average. We found that the Spin Image and Radial Intersection Count Images shape descriptors gave the overall most reliable performance, while the 3D Shape Context, Fast Point Feature Histogram, and Quick Intersection Count Change Image shape descriptors gave more varied results.

SAMMENDRAG

I denne oppgaven har vi testet hvilken grad shape descriptors ferdigheter knyttet til å kjenne igjen objekter påvirkes ved simulering av virkelige scenarier. Vi presenterer en benchmark som består av tre prosesser - Datainnsamling, Generering av ytelsesdata, og Dataanalyse. Denne oppgaven hadde som mål å svare på den manglende kunnskapen knyttet til hva som fører til god eller dårlig gjenkjenningssytelse for en gitt shape descriptor, og fraværet av store og varierte datasett brukt i tester av shape descriptorer. Ved å bruke et datasett på $n=1193$ objekter samlet av Google Research, ble en serie transformasjoner generert, og gjennom en benchmark ble shape deskriptorens ytelse i forskjellige scenarier testet og visualisert. Ytelsen til en shape deskriptorene ble testet gjennom vår foreslåtte metode, som innebærer å sammenligne resultatene med en tidligere beregnet dårlig gjennomsnittsverdi. Vi fant ut at Spin Image og Radial Intersection Count Images shape deskriptorene ga den mest pålitelige generelle ytelsen, mens 3D Shape Context, Fast Point Feature Histogram, og Quick Intersection Count Change Image shape deskriptorene ga mer varierte resultater.

PREFACE

Through the process of writing this thesis, we have been fortunate to explore the object recognition domain in depth, identifying gaps in knowledge and highlighting the role shape descriptors play in various computer vision applications.

A special thanks to Bart Iver van Blokland from NTNU who has been flexible in helping us creating this framework and understanding concepts related to shape descriptors.

CONTENTS

Abstract	1
Sammendrag	2
Preface	3
Contents	6
List of Figures	6
List of Tables	9
1 Introduction	2
1.1 Research Questions and Objectives	3
1.2 Stakeholders	3
2 Background & Related Works	5
2.1 3D Object Representation	5
2.1.1 Triangle Mesh	5
2.1.2 Point Cloud	6
2.2 Shape descriptors	7
2.2.1 Spin Image	8
2.2.2 3D Shape Context	9
2.2.3 Fast Point Feature Histogram	10
2.2.4 Radial Intersection Count Images	11
2.2.5 Quick Intersection Count Change Image	12
2.3 Distance functions	13
2.4 Other Methods for Object Recognition	14
2.5 Challenges Faced by Recognition Methods	15
2.6 Datasets for 3D Object Recognition	16
2.6.1 ABC: A Big CAD Model Dataset for Geometric Deep Learning	17
2.6.2 ShapeNet: An Information-Rich 3D Model Repository	17

2.6.3	Google Scanned Objects: A High-Quality Dataset of 3D Scanned Household Items	18
2.6.4	Procedural Generation for 3D Object Datasets	19
2.7	Evaluating Shape Descriptors	20
2.8	Gap in knowledge	20
3	Methods	23
3.1	Data Collection	24
3.1.1	Dataset Requirements	24
3.1.2	Dataset: Scanned Objects by Google Research	25
3.1.3	Dataset Preparation	27
3.1.4	Transformation Requirements	28
3.1.5	Transformations	29
3.1.6	Replicability	42
3.1.7	Transformed Objects and Metadata	43
3.2	Performance Data Generation	44
3.2.1	Comparing Descriptors	44
3.2.2	Descriptor Parameters	50
3.2.3	Implementation	52
3.3	Data Analysis	53
3.3.1	Implementation	54
3.4	Architecture Overview	55
4	Results	57
4.1	Generation Times	58
4.2	Comparison Times	58
4.3	Basic Matching Performance	60
4.3.1	Rotated Objects	60
4.3.2	Resized Objects	60
4.3.3	Moved Objects	60
4.3.4	Mirrored Objects	60
4.4	Deformation Resistance	65
4.4.1	Twisted Deformation of Objects	65
4.4.2	Rippled Deformation of Objects	65
4.5	Noise Resistance	65
4.5.1	Gaussian Vertex Displacement Along Normals	65
4.5.2	Deviation and Rotation of Vertex Normals	69
4.6	Clutter Resistance	69
4.6.1	Clustered Objects	69
4.6.2	Ability to Match with Partial Overlap	69
4.7	Occlusions and Incomplete Surfaces	69
4.7.1	Objects with Holes	73
4.7.2	Partial Surface Visibility	73

5	Discussion	77
5.1	Evaluating the Shape Descriptors	77
5.1.1	Basic Matching Performance	77
5.1.2	Deformation Resistance	78
5.1.3	Noise Resistance	78
5.1.4	Clutter Resistance	79
5.1.5	Occlusions and Incomplete Surfaces	79
5.1.6	Overall	79
5.2	Project Architecture	80
6	Conclusions	81
6.1	Future Work	82
	References	85
	Appendices:	93
A	Github repositories	94

LIST OF FIGURES

2.1.1 Representations of a bottle using a triangle mesh and point cloud	6
2.1.2 Visualisation of an object’s normals	7
2.2.1 Example of a Support Radius	8
2.2.2 Spin Image taken from different points in an object [10] . .	9
2.2.3 Support Volume of the 3DSC sphere [11]	10
2.2.4 Visualisation of how FPFH only connects each point to its closest neighbours [12]	11
2.2.5 Radial Intersection Count Image Generation Example [13] .	12
2.2.6 How QUICCI generates its histogram [14]	13
2.5.1 Example of a cluttered environment	15
2.5.2 Example of an object with occlusion	16
2.5.3 Example of object with Gaussian noise	16
2.7.1 Sketch of Spin Image’s evaluation method [10]	21
2.7.2 Performance of different descriptors in a cluttered environment [13]. Red representing areas they managed to recognise, and blue being the original object	21
3.1.1 Setup of Google’s scanning rig for high-quality model creation [49]	26
3.1.2 The original object (left) and its transformed versions with gradually increasing twisting intensity, illustrating the impact of twisted deformation on the object’s surface. Object ID 0009.	33
3.1.3 Original object (left) and its transformed versions with gradually increasing ripple effect intensity, showcasing the impact of ripple deformation on the object’s surface. Object ID 0036	34

3.1.4	Original object 0069 and its ten transformed versions with gradually increasing Gaussian distributed vertex displacement, ranging from the original object on the right to the version with the largest noise on the left. Each transformed object is individually tested, allowing for a comprehensive evaluation of shape descriptor performance under varying levels of noise.	35
3.1.5	Simultaneous depiction of deviated normals for ten object categories for illustrative clarity: the left portion displays an overlay of normals with deviations from 0 to 45 degrees, while the right portion presents the normals after undergoing a random rotation in addition to the deviation. The right one is used for evaluating the shape descriptors benchmark. During testing, each object category is assessed individually.	36
3.1.6	An example of a clustered arrangement of objects, where the primary object (Object ID 0054) is surrounded by 0, 5, 10, 20 and 100 other objects within the cluster.	38
3.1.7	An illustration of objects with varying degrees of partial overlap. The overlapping regions can be observed as the main object becomes increasingly covered by the additional objects. Object ID 0098	39
3.1.8	Illustration of object number 0476 in its original state and through various stages of the "Object with Holes" transformation, showcasing the gradual increase in the number and size of holes in the object's surface.	41
3.1.9	Illustration of the Partial Surface Visibility transformation using Unity raycasting. The image shows a side view and a raycast start position view of an object, along with the raycast path traced for visualization. Two androids are also shown, one shot from one angle and the other one demonstrating raycasting from two closely positioned points. . . .	43
3.2.1	Floor tests for the shape descriptors. Red line = Average distance, Green line = 90th percentile	48
3.2.2	Standard Deviation distribution for the shape descriptors	49
3.2.3	Red sphere representing the area taken into consideration with a 2.5 support radius	51
3.3.1	Examples of diagrams with all distances per category and average	54
3.4.1	Overview of Benchmark architecture	56
4.2.1	Generation time sorted on object's vertex count from low to high	59
4.3.1	Rotated Objects Results	61
4.3.2	Resized Objects Results	62
4.3.3	Moved Results	63
4.3.4	Mirrored Results	64

4.4.1 Twisted Deformation Results	66
4.4.2 Rippled Deformation Results	67
4.5.1 Gaussian Vertex Displacement Along Normals Results . . .	68
4.5.2 Deviation and Rotation of Vertex Normals Results	70
4.6.1 Clustered Objects Results	71
4.6.2 Ability to Match with Partial Overlap Objects Results . . .	72
4.6.3 Ability to Match with Partial Overlap Objects results for RICI, when object 977's result in category "65.1-75.0" is ignored	73
4.7.1 Objects With Holes Results	74
4.7.2 Partial Surface Visibility Results	75

LIST OF TABLES

2.3.1 Recommended shape descriptor and distance function combinations	15
3.2.1 Average Distance and 90th percentile for each shape descriptor	47
3.2.2 Average Distance for test on equal objects	47
3.2.3 Bin sizes and resolutions of the shape descriptors	51
4.0.1 Amount of time spent running each test, in hours	58
4.2.1 Average generation time for each descriptor	59
4.2.2 Average comparison time for each descriptor	59

INTRODUCTION

¹In the field of computer vision, 3D shape descriptors play a critical role in the identification and understanding of 3D object surfaces. These algorithms extract numerical values representing specific attributes of an object, which enables the analysis and processing of visual data. Which is faster than comparing the 3D surfaces directly. Shape descriptors have been utilised in various applications including shape registration [2–4], shape segmentation [5–7], and retrieval [8, 9]. Despite these advancements, the general consensus of the most suitable descriptor for different scenarios and applications is lacking.

The shortcomings are twofold: a) current evaluation strategies have only tested a narrow set of objectives, and b) they lack a proper measurement strategy. Thus, this thesis aims to address these issues by proposing a more comprehensive and systematic evaluation method, thereby enhancing our understanding of the strengths and weaknesses of different descriptor methods.

The scope of this study is limited to the evaluation of five widely used shape descriptors: the Spin Image [10], 3D Shape Context [11], Fast Point Feature Histogram [12], Radial Intersection Count Image [13], and the Quick Intersection Count Change Image [14]. This thesis aims to evaluate the effectiveness of these descriptors in various situations, encompassing categories of scenarios such as changes in either object orientation and scale, the introduction of noise, and the presence of clutter. To ensure a reliable and robust benchmark, attention is dedicated to the design and implementation of the testing environment and dataset, enabling evaluation across diverse scenarios.

¹*This chapter is based on the Autumn 2022 preparatory project [1].*

1.1 Research Questions and Objectives

With the lack of research in which shape descriptor has the best performance in different scenarios we define the following research question:

RQ1: To what extent are the object recognition abilities of shape descriptors affected when simulating real-world scenarios?

In order to answer this research question we define the following objectives, which will help us structure how to solve the challenges related to the research question:

1. Identify the criteria by which a shape descriptor should be evaluated.
2. Develop a method to measure these criteria in an objective manner.
3. Evaluate the performance of popular shape descriptors based on the identified criteria.

1.2 Stakeholders

This project's main stakeholders can be split into three different groups:

Authors: The authors, Haakon Gunnarsli and Jonathan Brooks, are active stakeholders in this project, as they are the ones building the framework and writing the thesis. Their main motivation for this project is to get a deeper understanding of shape descriptors, and helping future researchers within the field by constructing the benchmark.

Supervisor: This project's supervisor, Bart Iver van Blokland, is also an active stakeholder in this project. He has aided the authors with understanding concepts and in development. This project will be a useful tool for van Blokland in further understanding how to compare shape descriptors, and can be used in his future research.

Researchers: Researchers are passive stakeholders in this project. This group will benefit from the framework, as they get a universal standard that can be used to compare their own shape descriptors to others, which will help in improving their performance.

BACKGROUND & RELATED WORKS

¹This section will present the different methods and theories relevant for the thesis and 3D object recognition in general, as well as introduce the different shape descriptors and their corresponding distance functions that were chosen. In addition to the theory, the gap in knowledge that this research aims to answer will be presented.

2.1 3D Object Representation

There are many ways to represent 3D objects, where each method has its own pros and cons. Some popular methods include voxels, skeletons, point clouds, and triangular meshes [15]. However for this thesis only the point cloud and triangular mesh methods are utilised, since these are the ones used by the shape descriptors in this thesis. Point clouds are an effective digital representation of 3D scans, from for example LiDAR sensors [16], while triangular meshes provide an effective way to store digital 3D object data for rendering [17]. These two differ in what data they store for an object, which gives them some advantages over the other in specific situations (see Figure 2.1.1 for a visual representation). The following subsections will present these two methods for 3D object representation.

2.1.1 Triangle Mesh

A triangle mesh is a data structure for representing and visualising 3D objects. As a minimum, a mesh must have vertex coordinates, but it may also include additional attributes such as vertex normals and texture coordinates. The number and arrangement of these elements determine the overall shape and detail of the mesh.

Vertexes are used as a reference in order for a program to know where to place an object in 3D space. They are defined as a three dimensional

¹*This chapter is based on the Autumn 2022 preparatory project [1].*

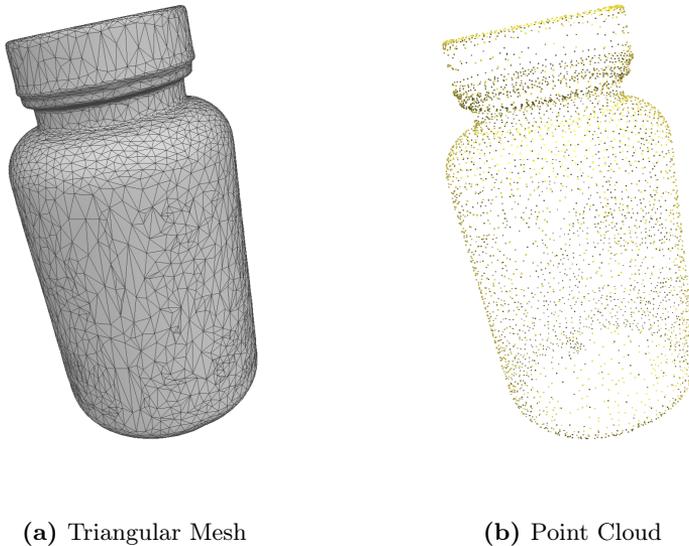


Figure 2.1.1: Representations of a bottle using a triangle mesh and point cloud

point, with an x -, y -, and z -coordinate. An object's normals are values that determine how an object should interact with light [18] (example can be seen in Figure 2.1.2). Faces determine which vertices are connected together. Usually, three vertices are connected together to form a triangle and the empty room between them is filled in. Meshes are useful for a variety of applications, including 3D modelling, animation, and simulation. Programs such as Unity use meshes extensively for this reason [19]. In addition to their computational efficiency, meshes have the advantage of being able to represent a wide range of shapes and geometries, from simple geometric primitives to complex organic looking forms [20].

2.1.2 Point Cloud

A point cloud is a collection of points in 3D space, typically representing the surface of a real-world object. It consists of a set of points in 3D space, which are defined using x -, y -, and z -coordinates [21]. They are the output of real world 3D scanners such as the aforementioned LiDAR scanner [16] and other laser based scanners [22]. Point clouds are also often used in computer graphics and computer vision applications, as converting a point cloud from a scanner to a triangle mesh, or other representations, is a demanding task [23].

Point clouds lack connectivity information between points, which is crucial for creating a triangle mesh. Converting a point cloud to a triangle mesh requires additional algorithms to establish connectivity and

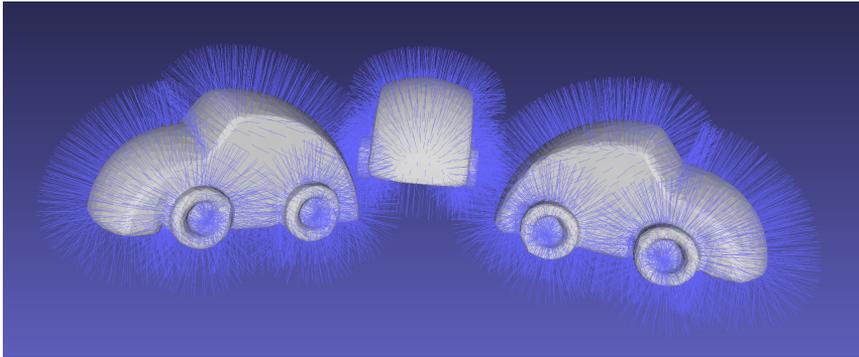


Figure 2.1.2: Visualisation of an object's normals

create a coherent surface representation [23]. This conversion can be difficult for several reasons; real world scanning methods often produce point clouds with noise and outliers, leading to inaccuracies and artefacts in reconstructed triangle meshes [24]. Point clouds may also have missing or sparse data, resulting in an incomplete or non-uniformly sampled representation, making continuous and coherent triangle mesh creation more challenging [25]. Furthermore, given only the point cloud, there might be multiple plausible ways to connect the points to form a triangle mesh, leading to ambiguity in surface reconstruction and potentially resulting in incorrect or sub-optimal surface reconstructions. Several algorithms, such as Poisson Surface Reconstruction [26], Ball Pivoting [27], and Alpha Shapes [28], have been developed to address these challenges. Despite these methods, unforeseen results may still occur.

Unlike meshes, which are made up of vertices, normals, and faces, point clouds do not have any inherent structure or connectivity between the points. In addition, point clouds cannot store any information about an object's texture or colour, making them less attractive to use in rendering applications [29].

2.2 Shape descriptors

A shape descriptor is a method used to describe the features and shape of an object, by registering and performing calculations on the 3D volume around given points of the object. They are generally used to compare an object to a set of predefined objects, in order to find the best match. The main idea being that computing numbers from descriptors is computationally cheaper than trying all different permutations when comparing object surfaces. The volume that is used in the calculations around a given point is called the *Support Volume*, and its shape is based on the shape descriptor in use. The volume can in some cases be a sphere (Figure 2.2.3), while others a cylinder. Additionally, the width and height used from a given point is called the *Support Radius*, and changes the area a

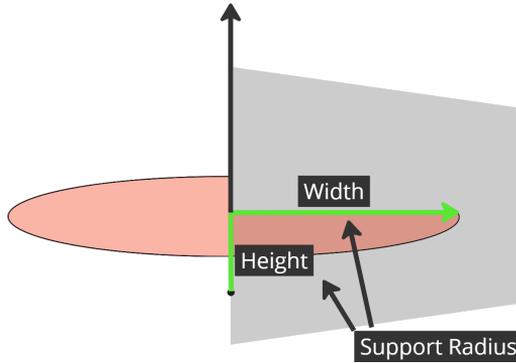


Figure 2.2.1: Example of a Support Radius

shape descriptor takes into account (see Figure 2.2.1).

How these calculations are done and how the output is used to represent the object varies from descriptor to descriptor, additionally some descriptors utilise point clouds [10–12] as their object reference while others use triangle meshes [13, 14]. This section aims to provide a comprehensive rundown of the chosen shape descriptors, along with their respective operational principles².

2.2.1 Spin Image

The Spin Image (SI) descriptor was introduced by Johnson in 1997 [30]. The goal of creating the Spin Image was to create a descriptor that managed to match surfaces well in real life scenes, where clutter and occlusion are common occurrences.

The descriptor utilises *Oriented Points*, points in 3D space with an associated direction, to generate the spin images. These points are defined by an objects’ vertex positions and their corresponding surface normal. For each vertex SI_v a plane consisting of $N_{bin} * N_{bin}$ bins of equal size is created. The plane rotates around the vertex’s normal SI_n , and increases the value for each bin if a point intersects with it. The Spin Image also has some parameters that can be changed, depending on the use case. The size of each of its bins can be modified, in order to change the resolution of the output. Additionally, the *Support Angle* can be modified which is the maximum angle between the given point’s direction and the normal of points within the Support Radius [10].

The output of the descriptor can be seen as a 2D image, where the darker parts are areas with a high number of overlapping points. An example of this can be seen in Figure 2.2.2, where the β axis denotes the vertical displacement of a point concerning the reference point SI_n and the α axis represents the horizontal distance of the same point from the

²Implementations for the following descriptors can be found in the *libShape-Descriptor* Github repository (Appendix A).

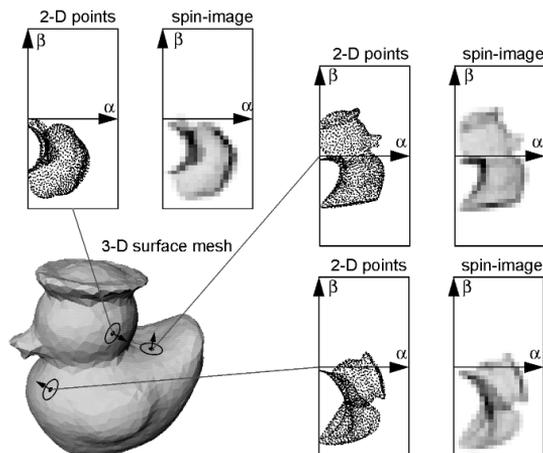


Figure 2.2.2: Spin Image taken from different points in an object [10]

said reference point [10].

Johnson et al. performed a test in their paper on using Spin Images for 3D Object Recognition, where they observed how well the descriptor worked in environments containing a varying amount of occlusion and clutter. They then compared the results of the common Spin Image to a compressed version using the Principle Component Analysis. The dataset they utilised consisted of 20 objects, where 10 were toys and the rest were types of pipes. They found that the recognition rate of the Spin Image decreased in cluttered environments when occlusion was more prevalent. Recognition will work up to around 70% occlusion where the recognition rate will begin to drop off for both the compressed and non-compressed spin images [10].

2.2.2 3D Shape Context

The 3D Shape Context (3DSC) shape descriptor was proposed by Frome et al. in 2004, and is built as a 3D version of the 2D Shape Context descriptor by Belongie et al. [31]. One of the goals of the 3DSC shape descriptor was to be able to perform object recognition in noisy and cluttered scenes, for example the output of a range scanner. Another problem that can occur in these environments is occlusion, where a part of an object is not visible in the scan. Similar to other shape descriptors, 3DSC takes an oriented point $3DSC_p$ as its starting point and orients itself in the same direction as its normal $3DSC_n$. The support volume is represented as a sphere and is divided into bins, which have equally spaced boundaries in the azimuth and elevation dimensions and logarithmically spaced boundaries along the radial dimension (visualised in figure 2.2.3).

The bins closest to $3DSC_p$ are very small compared to the others, therefore in order to compensate for not being so sensitive to small differ-

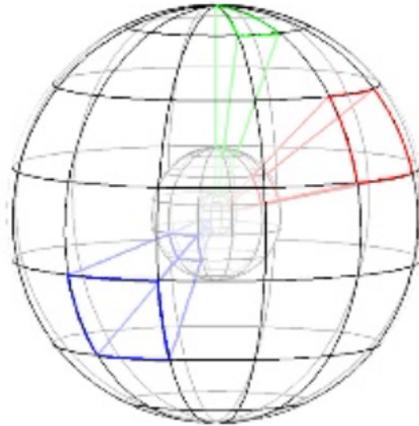


Figure 2.2.3: Support Volume of the 3DSC sphere [11]

ences in the shapes a minimum radius is used. It is also possible to change the max radius of the sphere (i.e. Support Radius), which can be changed determined of the size of the objects. Each bin uses a weighted count of all the points which fall inside its boundaries. Since the descriptor doesn't take into account the orientation of an object, to be able to compare two descriptors one of the descriptors needs to be rotated around its axis. Then the lowest distance of the different permutations will be chosen [11]. This results in a slower matching rate than other descriptors [13].

In its paper, the 3DSC descriptor's performance was compared to the Harmonic shape contexts (HSC) and Spin Image descriptors. The dataset used was a set of 56 3D models of different cars, scaled up to their actual sizes. Further, the objects were converted into point clouds using a laser sensor simulator. The 3DSC descriptor outperformed both SI and HSC when adding Gaussian noise to the query scans. Additionally, it also managed to outperform the other descriptors in cluttered environments where it on average managed to identify 78% of the models using the top five choices for each scene [11].

2.2.3 Fast Point Feature Histogram

The Fast Point Feature Histogram was proposed in 2009 by Rusu et al., and is an optimised version of the previous Point Feature Histogram (PFH) descriptor [32]. FPFH has a significantly better algorithmic run-time compared to the previous version, at $\mathcal{O}(nk)$ as opposed to PFH's $\mathcal{O}(nk^2)$. It works by taking each point $FPFH_p$ and calculates the relationships between it and its neighbours (figure 2.2.4), called Simplified Point Feature Histogram ($SPFH$). Following this, for each point its k neighbours are redetermined and the neighbouring $SPFH$ values are used to weight

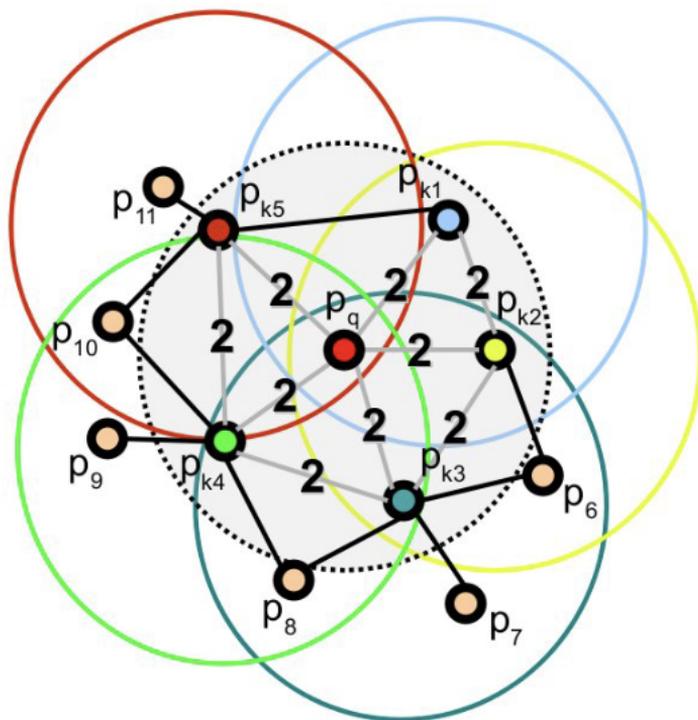


Figure 2.2.4: Visualisation of how FPFH only connects each point to its closest neighbours [12]

the final FPFH value for that point [12]. With FPFH’s default parameters, of 11 bins, the output histogram will be a 33-byte array of float values [33]. Which compared to the other descriptors only a QUICCI with 32x32 bin size, equivalent to the size of 32 floats, would out perform [12].

2.2.4 Radial Intersection Count Images

The Radial Intersection Count Images (RICI) was proposed by van Bloklund et al. in 2020. As clutter has been named one of the larger factors to contribute towards poorer performance in current shape descriptors [34], RICI aimed to provide a clutter-resistant descriptor with faster generation times than previously existing shape descriptors. The RICI descriptor uses a similar concept to the Spin Image in how to compute the descriptors. Using Oriented Points, it creates a histogram using a square, of size $N_{bin} * N_{bin}$, which can be visualised as an image. The difference between SI and RICI is how the histogram is calculated. Instead of counting oriented points, RICI counts the amount of intersections made with circles which surrounds the vertex normal. Each column in the histogram rep-

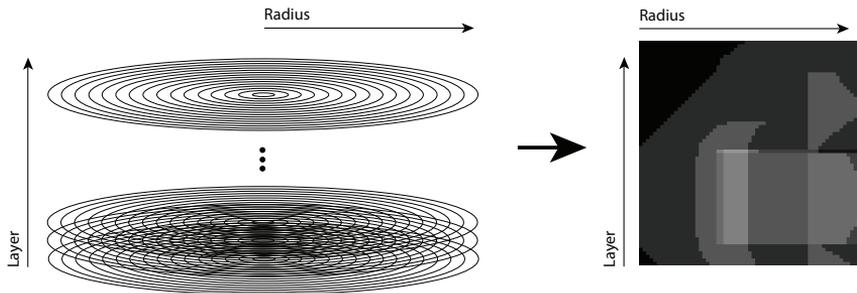


Figure 2.2.5: Radial Intersection Count Image Generation Example [13]

resents a layer of the object, while the rows represent the different circles (visualised in Figure 2.2.5). As with the Spin Image, it is also possible to increase or decrease the Support Radius which will determine the radius of covered by the circles [13].

In its original paper, the RIC I descriptor was compared to the Spin Image and 3D Shape Context descriptors. The criteria tested were clutter resistance, generation time, and matching performance. SHREC2017 [35] was used as the dataset of choice, with over 51000 triangle meshes over 55 common categories. As both the 3DSC and SI descriptors utilises point clouds, while RIC I uses triangle meshes, it was important when converting the triangle meshes to point clouds to use a high sample count to ensure a low level of noise. The tests show that RIC I significantly outperforms the others in both clutter resistance and generation time, while RIC I only outperformed the rest while using an early exit when looking at the matching rate. When the early exit is not used RIC I has a similar matching rate to the SI, but they both outperform the 3DSC descriptor [13].

2.2.5 Quick Intersection Count Change Image

The Quick Intersection Count Change Image descriptor (QUICCI) was also proposed by van Blokland et al. in 2020, and was made to solve the problem of retrieving partial objects efficiently. Searching for objects containing a specific feature in a large dataset can be quite memory intensive, as the descriptors generated has to describe the object in such small detail. QUICCI addresses this by basing itself on how RIC I generates histograms, but instead of storing the sum of intersection changes it only stores boolean values (example can be seen in Figure 2.2.6). These boolean values represent if there has been any changes in intersections from one circle to the next. This results in a much smaller descriptor size compared to other descriptors, which utilises datatypes such as floats or integers in their output [14].

Using the same testing environment as in the RIC I tests, QUICCI's performance was compared to the RIC I, SI, 3DSC, and Fast Point Feature Histogram shape descriptors. It is found that QUICCI performs at

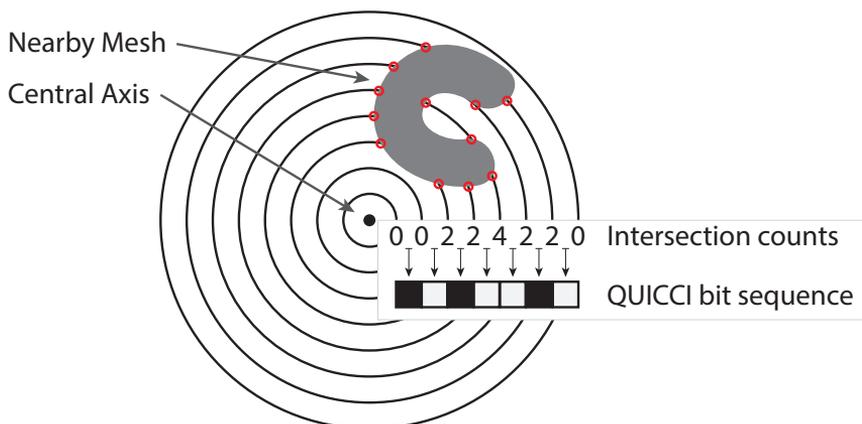


Figure 2.2.6: How QUICCI generates its histogram [14]

the same level as RICCI in clutter resistance, but with a slightly higher performance. Additionally, with QUICCI's binary output it manages to have a much higher comparison and generation rate than the other shape descriptors [14].

2.3 Distance functions

When performing object recognition, distance functions can be used to compare the sets of descriptor features. Many of the shape descriptor papers provide a recommendation for which distance function should be used in tandem with them (can be seen in Table 2.3.1). The distance functions work in different ways by weighting various factors, which can cause the confidence of two objects being similar to change depending on which distance function you choose. There are a lot of different distance functions, for example Euclidean and Pearson Correlation which have been used in the previously mentioned tests [11, 13]. The following section will give an introduction to the different distance functions relevant for the paper, which will show how they differ to each other.

Euclidean Distance is one of the more commonly used distance metrics for computer vision tasks, it was the metric of choice for the original 3DSC paper [11] and is also used in the Point Cloud Library³ [36] with FPFH. The distance output is the direct distance between two points in Euclidean space [37].

$$\text{euclidean distance}(A, B) = \sqrt{\sum_{i=1}^n (A_i - B_i)^2} \quad (2.1)$$

Clutter Resistant Distance was introduced together with the RICCI

³<https://pointclouds.org/>

descriptor. It uses the same principles as RICCI, but instead of just counting intersection changes it sums up the the difference in intersection counts for the rows where there is a change from one descriptor to another, then it squares the sum. As only the rows where there has been a change in the intersection count is included, the clutter is safely ignored [13].

$$D(rici, r, c) = rici(r, c) - rici(r, c - 1) \quad (2.2)$$

$$CRD(n, h) = \sum_{r=0}^{Nbins} \sum_{r=1}^{Nbins} \begin{cases} (D(n, r, c) - D(h, r, c))^2, & \text{if } D(n, r, c) \neq 0 \\ 0, & \text{otherwise} \end{cases}$$

Weighted Hamming Distance is a modified version of the Hamming Distance made for the QUICCI descriptor. The Hamming Distance is a metric used to compare binary strings, where the result is the amount of 1s after doing the XOR operation [38]. The WHD is a solution to when the output descriptor has a large surplus of either 1 or 0, which would cause the normal Hamming Distance to give irrelevant shapes when performing shape retrieval. WHD addresses this by weighting the bits depending on the amount proportion of 1s and 0s [14].

$$D_{WH}(I_n, I_h) = \frac{\sum_{r=0}^N \sum_{c=0}^N (I_n[r, c](1 - I_h[r, c]))}{\max\left(\sum_{r=0}^N \sum_{c=0}^N I_n[r, c], 1\right)} \quad (2.3)$$

$$+ \frac{\sum_{r=0}^N \sum_{c=0}^N ((1 - I_n[r, c])I_h[r, c])}{\max\left(N - \sum_{r=0}^N \sum_{c=0}^N I_n[r, c], 1\right)}$$

Pearson Correlation is a commonly used metric to measure the linear correlation, which has been previously used to compare Spin Image descriptors [13]. It is used to determine the strength and direction of the relationship between two variables. Its output is between 1 and -1, where 1 indicates a strong correlation, 0 is no correlation, and -1 represents a negative correlation between the variables [39].

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}} \quad (2.4)$$

2.4 Other Methods for Object Recognition

In addition to the improvements and introduction of new methods in the field of shape descriptors, there have also been done considerable advancements in object recognition through machine learning. Such as the MCRBM model proposed by Han et al. in 2016, which was compared amongst others to the Spin Image and even proved to perform better in the task of shape retrieval [40]. Even though machine learning methods can in some situations perform better than shape descriptors, they come

Shape Descriptor	Distance Function
RICI	Clutter Resistant Distance [13]
QUICCI	Weighted Hamming Distance [14]
SI	Pearson Correlation [10]
3DSC	Euclidean Distance [11]
FPFH	Euclidean Distance [33]

Table 2.3.1: Recommended shape descriptor and distance function combinations

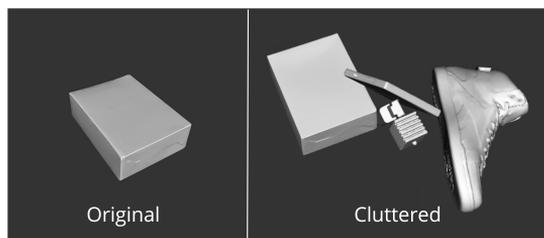


Figure 2.5.1: Example of a cluttered environment

with some computational problems compared to shape descriptors. As they require a lot of processing power, re-training a model can become quite resource intensive which could limit them to using smaller datasets. Additionally, the task of how to structure 3D data for a neural network has been deemed not trivial [41].

2.5 Challenges Faced by Recognition Methods

In the real world, shape descriptors need to be able to handle a wide range of different challenges which can be introduced by the environment the objects are in or the scanner used. As mentioned, many of the previously named shape descriptors were created to tackle the problem of cluttered environments. Such as SI, 3DSC, and RICI [10, 11, 13]. An environment is cluttered when the original object you want to recognise is surrounded by other objects, as can be seen in Figure 2.5.1. As the shape descriptors use the volume around given points in an object, if another object is within that volume it can affect the values the shape descriptors' output. Making it harder to perform object recognition.

Another problem which is usually present in real world scans is occlusion [42]. Occlusion can happen when an object is not fully scanned, leaving parts of the object out of the scan. The amount of occlusion can vary significantly, but it can still affect the shape descriptors' outputs. If

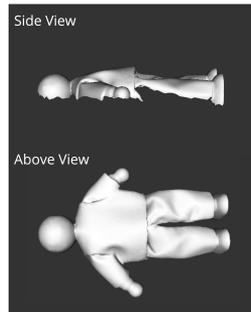


Figure 2.5.2: Example of an object with occlusion



Figure 2.5.3: Example of object with Gaussian noise

parts of an object is not present it can have an effect on the shape of the object, leading to descriptors for the same given point in two objects being dissimilar. We can see an example of an object with occlusion in Figure 2.5.2, here the back side of the doll has not been scanned.

Noise from scans can also be a problem shape descriptors are affected by. This can be caused by inaccuracies in the scanners' optical components or electronic noise [43]. This noise is usually distributed following the Gaussian distribution, meaning that it follows a normal distribution [44]. A scan that has been exposed to this type of noise can often look disfigured, such as in Figure 2.5.3. Even with the advancements of algorithms that can remove this type of noise [43, 45, 46], it can still be present and is a problem shape descriptors need to be aware of.

2.6 Datasets for 3D Object Recognition

In the field of 3D object recognition, numerous datasets have been developed to facilitate research and evaluation of various methods and techniques [35, 47–49]. These datasets provide researchers with comprehens-

ive collections of 3D objects, which serve as benchmarks for evaluating the performance of their algorithms. The significance of discussing these datasets lies in their wide usage in related research fields, as they allow for comparison of results and promote standardisation across different studies. In this study, we examine three prominent datasets: the ABC-Dataset, the ShapeNet dataset, and the Google Scanned Objects dataset.

Moreover, the process of creating large, diverse datasets can be labour-intensive and time-consuming, particularly if done manually. Therefore, an automated process, such as procedural generation, can be invaluable. Procedural generation uses algorithms to produce vast datasets with diverse content. This approach enables researchers to generate specific datasets tailored to their needs, offering potential advantages in terms of scalability and customisation. Therefore, we explore the use of procedural generation techniques for creating 3D object datasets.

2.6.1 ABC: A Big CAD Model Dataset for Geometric Deep Learning

Introduced by Koch et al., the ABC-Dataset is a unique resource that houses over one million CAD models, predominantly featuring mechanical shapes. These models are defined by parameterised curves and surfaces and offer valuable ground truth data for tasks such as patch segmentation, geometric feature detection, and shape reconstruction. The richness of this dataset lies in its diversity, which allows for meaningful testing and refinement of algorithms that analyse and characterise mechanical shapes.

Supported by Onshape’s extensive CAD model repository, the dataset has been utilised for benchmarking surface normal estimation, enabling comparison between data-driven methods and traditional geometry processing algorithms. The models’ encoding as triangle meshes allows for resampling at arbitrary resolutions, offering flexibility for research, such as testing shape descriptor algorithms on objects with varying levels of detail.

The ABC-Dataset offers an ever-growing resource for geometric deep learning research, with new models continually being added to the public collection [47].

2.6.2 ShapeNet: An Information-Rich 3D Model Repository

ShapeNet is an ongoing, collaborative effort between researchers at Princeton, Stanford, and TTIC to establish a richly-annotated, large-scale dataset of 3D shapes [48]. The aim of ShapeNet is to provide researchers worldwide with data that enables research in computer graphics, computer vision, robotics, and related disciplines. ShapeNet contains 3D models from various semantic categories, organised under the WordNet taxonomy [50]. The dataset offers numerous semantic annotations for each 3D model,

such as consistent rigid alignments, parts and bilateral symmetry planes, physical sizes, keywords, and other planned annotations.

Currently, ShapeNet consists of several subsets, including ShapeNet-Core and ShapeNetSem. ShapeNetCore is a subset of the full ShapeNet dataset, with single clean 3D models and manually verified category and alignment annotations. The dataset covers a range of 55 object categories, which includes the 12 object categories from PASCAL 3D+, a well-known computer vision benchmark dataset [51]. In total, the dataset consists of approximately 51,300 unique 3D models. ShapeNetSem, on the other hand, is a smaller and more densely annotated subset, consisting of 12,000 models spread across a broader set of 270 categories. These models are annotated with real-world dimensions, material composition estimates at the category level, and total volume and weight estimates.

ShapeNet has indexed more than 3 million models, with 220,000 models classified into 3,135 categories (WordNet synsets) [48]. It provides extensive sets of annotations for every model and links between models in the repository and other multimedia data outside the repository. Annotations in ShapeNet include geometric attributes such as upright and front orientation vectors, parts and key points, shape symmetries, and the scale of objects in real-world units.

The raw 3D model data for ShapeNet comes from public online repositories or existing research datasets. ShapeNet’s data is collected from two popular public repositories, Trimble 3D Warehouse and Yobi3D. The Trimble 3D Warehouse contains 2.4 million user-designed 3D models and scenes, while Yobi3D contains 350,000 additional models collected from a wide range of other online repositories. Together, these sources provide a diverse set of shapes from various object and scene categories.

In the study “Radial intersection count image: A clutter-resistant 3D shape descriptor” [13], the researchers chose to use the combined SHREC2017 dataset, which consists of 51,162 triangle meshes, emphasising the relevance of ShapeNet for benchmarking shape descriptors [35].

2.6.3 Google Scanned Objects: A High-Quality Dataset of 3D Scanned Household Items

The Google Scanned Objects dataset, developed by Google Research, is a collection of 3D-scanned common household items consisting of 1,030 objects [49]. Presented in the paper titled “Google Scanned Objects: A High-Quality Dataset of 3D Scanned Household Items” at the ICRA 2022 conference, this dataset addresses the need for realistic 3D object models in interactive 3D simulations, synthetic perception, and robotic learning applications. Focusing on real scans of objects, the dataset offers a more accurate representation of real-world scenarios compared to CAD models or other artificial representations.

Comprising a diverse range of household items, such as furniture, appliances, tools, and decorative items, the Google Scanned Objects dataset is well-suited for training and testing algorithms across various computer

vision and robotics domains. Each object in the dataset comes with high-resolution shape and texture information in the form of Simulation Description Format (SDF) models, which are compatible with Gazebo and PyBullet robotics simulators [49].

The 3D models in the dataset are generated from real scans, ensuring that the shape and texture data are of high quality and closely resemble real-world objects. The scanning process involves capturing images of objects from multiple angles under controlled and calibrated conditions using a structured-light 3D scanner, thereby providing an accurate representation of the objects' geometry and appearance [49].

An automated pipeline preprocesses and cleans the dataset by filtering out invalid or duplicate objects, estimating simulation properties, constructing collision volumes, and downsampling the models to usable sizes. Additionally, the pipeline converts the models to SDF format, creates thumbnail images, and packages the models for use in simulation systems.

The Google Scanned Objects dataset offers a diverse and variable set of objects that reflect real-world object properties rather than idealised recreations. This diversity facilitates the transfer of learning from simulation to real-world scenarios and promotes the development of robust computer vision algorithms.

This dataset, which is publicly available and has already been utilised in over 25 papers across various projects, including computer vision, computer graphics, robot manipulation, robot navigation, and 3D shape processing. The majority of these projects have employed the dataset to provide synthetic training data for learning algorithms, with applications such as Kubric [52], an open-source generator of scalable datasets for over a dozen vision tasks, and LAX-RAY[53], a system for searching shelves with lateral access X-rays to automate the mechanical search for occluded objects on shelves [49].

2.6.4 Procedural Generation for 3D Object Datasets

Procedural generation is a technique that uses algorithms and rules to artificially generate content. It is used to create intricate and diverse 3D objects that are essential for a variety of applications such as computer graphics, virtual reality, video games, and simulations [54, 55]. This approach enables the creation of vast datasets of 3D objects without requiring manual design or modelling [56].

A common method for generating 3D objects through procedural generation is L-systems, which were initially proposed by Aristid Lindenmayer to model plant growth [57]. L-systems use formal grammars and production rules to generate 3D shapes iteratively. These grammars dictate how a starting shape, known as an axiom, transforms into a more complex shape by recursively applying production rules. L-systems have been successfully employed to create a wide array of 3D objects, including plants, trees, and fractals.

Another technique for generating 3D objects involves shape grammars,

which rely on a set of rules to determine how shapes can be combined or transformed to create new shapes. Shape grammars have been used in various fields, such as architecture, where they have facilitated the generation of building models and cityscapes. For example, Müller et al. [58] introduced a procedural modelling system for creating buildings based on user-defined shape grammar rules. This system enables the generation of large datasets containing diverse and realistic building models with minimal manual input.

Recently, machine learning techniques have been incorporated into the procedural generation of 3D objects. Deep learning models, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), have been employed to learn the underlying distribution of 3D object shapes and generate new objects by sampling from this distribution [59]. This method can create extensive datasets comprising diverse and realistic 3D objects, which can be utilised for a range of applications, including the training and evaluation of computer vision and robotics algorithms.

2.7 Evaluating Shape Descriptors

During the creation of a shape descriptor it is important to be able to evaluate its performance against existing ones. This can aid in the development phase, as a researcher is able to constantly see what can be improved. As well as, when it is fully developed evaluate it to existing alternatives to discover where it outperforms the rest. Each of the aforementioned shape descriptor papers [10–14] have their own evaluation methods, however they follow similar steps. Usually they start by calculating all the descriptors for an object O and all the descriptors in a comparison object C . Then they map the descriptors in O to the descriptors in C with the lowest distance between them, for example if euclidean distance is used, they want to find the corresponding descriptor which gives the distance closest to 0. Then it is possible to use some sort of ground truth, that can analyse how well the shape descriptor managed to find the correct descriptors in C . A take on this process can be seen in the SI paper’s sketch, where Johnson et al. [10] presents their evaluation method (Figure 2.7.1).

This method can work, but as brought into light in the RICI paper self-similarity within the objects can cause this test to be up to luck [13]. As seen in Figure 2.7.2, some of the bookcase’s shelves are practically identical which causes the descriptors to be matched with a descriptor that is the same, but just at the wrong place. This shows a clear need for a more general method in order to evaluate shape descriptors.

2.8 Gap in knowledge

The available documentation regarding the effectiveness of each shape descriptor is inadequate, and the factors that contribute to a good or bad

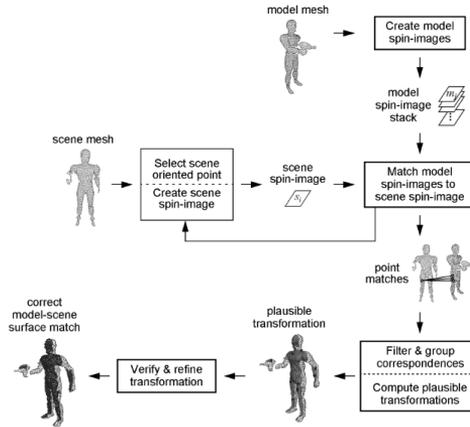


Figure 2.7.1: Sketch of Spin Image’s evaluation method [10]

Scene	RICI	3DSC	SI

Figure 2.7.2: Performance of different descriptors in a cluttered environment [13]. Red representing areas they managed to recognise, and blue being the original object

matching performance have not been adequately tested. The limited or small sets of objects used in previous tests may lead to biased data, which is not optimal. For example, the Spin Image paper tested only 20 models [10], and the 3D Shape Context paper tested their descriptor on only 56 car models [11]. By using such a small sample of objects, and in 3DSC's case only using cars, the tests become too specific to gather any general conclusions from. The tests presented in the previous papers also usually only cover one or two different challenges shape descriptors can face in the real world. It is clear that more extensive and diverse performance tests are required to accurately assess the effectiveness of various shape descriptors. Additionally, a new evaluation method for comparing shape descriptors without any biases or luck needs to be created to ensure fairness while testing. These gaps in knowledge will be addressed by this thesis.

METHODS

The main challenge of this thesis is to determine an effective method for comparing 3D shape descriptors. As previously mentioned, these shape descriptors are used in a wide range of different computer vision applications. Therefore it is important to figure out which shape descriptors are best for certain types of tasks. Previous papers [10–14] have showed how their shape descriptors perform in different environments. However, their findings are varied and they have utilised different methods for evaluation. Hence, the necessity for a new standardised evaluation method becomes apparent.

This necessity can be met by implementing a shape descriptor benchmark. A benchmark is a standardisation of how things should be compared or assessed, according to the Cambridge Dictionary. By creating a standardisation we can define general methods for comparing the shape descriptors’ performance, as well as defining which metrics should be used to analyse them. It is important to standardise these metrics, as when future researchers want to compare their shape descriptors to existing ones, using a benchmark will legitimise their findings and save them from time spent creating a testing environment. Additionally, it would guide researchers and developers in selecting the optimal shape descriptors for their specific applications.

There are however many requirements and challenges related to implementing this. Defining these requirements will ensure that the implementation covers the necessary areas in order to answer the question of how shape descriptors are affected in different real world scenarios. Furthermore, they will provide an overview of the challenges related to creating a benchmark. Three separate categories were identified for the requirements. Each of these categories represent an individual process within the benchmark, and they go as follows:

- Data Collection (Section 3.1): The process of selecting a dataset and produce transformations that can be used for performance data generation.

- **Performance Data Generation** (Section 3.2): Running the datasets through a set of tests, in order to generate performance data for each shape descriptor.
- **Data Analysis** (Section 3.3): Analysing the generated performance data in order to draw conclusions on how the shape descriptors are affected.

Another general requirement that is relevant for all these categories, is making sure that all generated results are replicable. Replicability allows other researchers to verify, validate, and build upon the findings of a previous study [60, 61]. Ensuring replicability in a master’s thesis demonstrates the rigor, transparency, and reliability of the research methodology, contributing to the production of high-quality and trustworthy results [62, 63].

As the aforementioned benchmark categories represent the individual processes of the benchmark, they all represent their own unique challenges. Each of processes’ challenges and requirements to solve these challenges will be separately introduced in the following sections.

3.1 Data Collection

The data collection process is divided into two primary stages. The first stage involves establishing the criteria for a suitable dataset and then detailing the process of obtaining a dataset that meets these requirements. The second stage focuses on identifying the requirements for necessary transformations, which will ensure appropriate testing of a shape descriptor. The subsequent sections will elaborate on the process of creating these transformations, but first we will introduce some general principles regarding the data collection implementation.

3.1.1 Dataset Requirements

We define the following requirements for finding a suitable dataset:

Adoption and Relevance: An ideal dataset should have been adopted by other researchers for evaluating applications within the same field. This demonstrates the dataset’s relevance and suitability for the task at hand, allowing for better comparisons against other work and fostering advancements in the field.

Availability: Prioritising availability promotes transparency, reproducibility, and collaboration within the scientific community, enhancing the credibility and reach of our research.

Datatype: The datatype the shape descriptors will have as input, with some based on point clouds and others on triangle meshes. Converting from triangle mesh to point cloud is relatively simple, as a

point cloud can be sampled from the vertices of the triangle mesh. However, converting from point cloud to triangle mesh poses more challenges due to the inherent nature of point clouds and the information they contain [23, 26], as previously discussed in section 2.1.2.

Diversity: A suitable dataset should encompass a wide variety of object classes, capturing the rich complexity of real-world objects ranging from small everyday items like matches to large objects like planes. Such diversity allows for a more comprehensive evaluation of shape descriptors, as it ensures that they are tested on a diverse set of shapes and surface characteristics.

Labelling: A satisfactory dataset has great labelling so that structuring, sorting, and filtering the results of a shape descriptor benchmark. Detailed and accurate labels provide a means of categorising and organising the data, simplifying the task of data analysis.

Large cardinality: The cardinality influences the statistical significance and precision of empirical findings. Generally, increased dataset cardinality tends to promote enhanced accuracy and reliability in results, subject to various assumptions.

Real Objects: Using 3D scanners to capture real objects ensure accurate evaluation of shape descriptors performance on real object detection. As many computer vision and robotic applications require interacting with genuine objects in everyday environments, it is crucial to test shape descriptors under conditions that accurately represent the complexity and randomness of real-world objects [64].

Single-Entity Objects When designing a testing methodology such as this benchmark, an important step is to minimise uncontrolled variables in order to obtain accurate and reliable results. By isolating individual variables, researchers can focus on measuring the effects of these variables without introducing confounding factors that might compromise the integrity of the study. By having a dataset with single-entity objects we can fully control the number of objects that the shape descriptors are tested on.

3.1.2 Dataset: Scanned Objects by Google Research

As introduced in Background section 2.6, the Scanned Objects dataset, developed by Google Research, is a collection of 3D-scanned common household items. It comprises a diverse set of objects, ranging from kitchen utensils to office supplies and toys, making it an ideal choice for evaluating shape descriptors performance in various real-world scenarios [49]. The dataset has been curated to ensure high-quality scans with accurate labelling. In this section, justification for selecting this dataset for the project will be provided.



Figure 3.1.1: Setup of Google’s scanning rig for high-quality model creation [49]

Google Research’s Scanned Objects dataset meets our data needs primarily due to its quality, diversity, and its representation of real-world objects. The data quality is ensured by the use of a system that consists of two machine vision cameras for object scanning, a DSLR camera for high-quality HDR color frame extraction, and a computer-controlled projector for pattern recognition (as seen in Figure 3.1.1). The scanning rig uses a structured light technique that infers a 3D shape from camera images with patterns of light that are projected onto an object. These scans represent real items, offering a practical, real-world context to test shape descriptor performance [49].

The diversity of the dataset derives from its inclusion of a wide assortment of household items. This results in objects that are roughly similar in size, which is beneficial for testing shape descriptors. This is because a large variance in size introduces a performance factor not yet thoroughly researched. Even though object size wasn’t a requirement, the relatively equal sizes of the objects turned out to be beneficial for the benchmark. As will be discussed in Section 3.2.2.1, objects of equal sizes makes the performance tests more fair when comparing shape descriptors.

As this project is as much about proposing a new benchmark for shape descriptors’ performance as it is producing relevant results, we have made our solution with scalability in mind. A well-planned strategy for working with a benchmark involves starting with only a few objects to enable faster debugging and optimisation at each stage. As the solution is debugged and improved, the number of objects can be gradually increased, eventually adding as many as time and computational constraints allow. Which will make the results more accurate.

This exhibits how the Scanned Objects dataset, consisting of over 1000 objects, serves as an good choice for this research project. Larger data-

sets such as the ABC and ShapeNet datasets, as well as merging several datasets together was also considered. However, these options came with some drawbacks:

- **Pre-processing efforts:** Merged datasets can have varying labelling systems, file formats, and data structures, which might require substantial preprocessing efforts. These tasks would add extra work that may not align with the project timeline constraints.
- **Time-consuming test runs:** Larger datasets can extend the time needed for running tests and generating results. This increased computational demand makes it less feasible for timeline of this project. However, is highly encouraged as future work for this project.

These drawbacks further encourages the use of the Scanned Objects dataset, as it provides a good variety of objects and enables us to test all its objects in a timely fashion. The previously presented datasets also mostly consists of synthetic 3D models, which may not accurately represent real-world objects. This limitation reduces their effectiveness in evaluating shape descriptors in real-world scenarios.

The Google dataset’s validity is even further ratified by its use in the same other computer vision research projects, indicating its applicability and effectiveness in this field [65–70]. Lastly, its public availability facilitates easy access, encouraging collaboration and promoting a culture of open-source knowledge sharing. All these attributes make the Scanned Objects dataset a fitting choice for our project.

3.1.3 Dataset Preparation

When designing a testing methodology such as this benchmark, an important step is to minimise uncontrolled variables in order to obtain accurate and reliable results. By isolating individual variables, researchers can focus on measuring the effects of these variables without introducing confounding factors that might compromise the integrity of the study. Consequently, the dataset underwent several modifications to enhance its validity and ensure that the conclusions drawn from the tests would be meaningful. These modifications were meant to isolate variables and promote consistency throughout all tests.

Firstly, some files contained several objects that were disconnected from one another. We addressed this by separating them into distinct objects. Secondly, this subset of disconnected objects also contained elements that were too similar to each other. To ensure that we were working with only distinct objects, we established a criterion for uniqueness, which required a minimum of 15% difference in vertex count. We then removed any elements from these objects that were too similar to each other, thus ensuring that we were working with unique objects in our tests. This increased our dataset to a total of 1193 unique objects.

Next, we recalculated vertex normals for all objects in the dataset. This step was for maintaining consistency in the computation method used for normals throughout the tests. Recalculating normals from the beginning is important in scenarios such as noise or objects missing parts, where all normals have to be recalculated after the transformation has been applied to the object. Consistent vertex normal calculations helped to eliminate variations, allowing for a more accurate comparison of objects and their properties during the benchmark tests.

3.1.4 Transformation Requirements

This section outlines the process of determining the requirements that a benchmark must meet to evaluate the performance of shape descriptors under various real-world scenarios. A well-constructed test should be designed to evaluate the algorithm’s performance across various scenarios, replicating the complexities and challenges that would typically be encountered in real-world applications.

In the real world, objects are subject to transformations, including rotations, scaling, deformations, noise, clutter, occlusions, and surface incompleteness. These transformations can significantly impact the performance of a shape descriptor algorithm. With this in mind, we have identified the following key performance requirements:

Basic matching performance: To determine whether the algorithm consistently produces the same descriptor for the same surface point on two copies of the same shape, even when one of the copies is rotated or scaled. This requirement tests the algorithm’s robustness to basic transformations that are common to encounter in real-world scenarios. These are transformations such as rotation, scaling, translation, and mirroring. These types of experiments should test the shape descriptor performance when faced typical object variations that is frequently encountered in real-world applications.

Deformation resistance: Everyday objects are not rigid; they bend, warp, and deform. If shape descriptors can successfully resist these deformations and recognise the object, it demonstrates their applicability in a dynamic, real-world environment and not just static scenarios.

Noise resistance: Noise can be introduced to 3D objects in application where there is environmental factors or limitations of the data collection equipment. Thus, it is valuable to test the algorithm’s ability to produce reliable descriptors in the presence of noise.

Clutter resistance: In real-world scenarios, an object of interest may be surrounded by other objects, leading to partial overlap or clustering. This makes it challenging to isolate and analyse them. This requirement aims to promote experiments that produces tests that

can be used for testing clutter resistance as this is something that computer vision applications are likely to encounter.

Occlusions and Incomplete Surfaces: A relevant scenario is partial visibility of objects or incomplete surfaces, typically due to occlusions or limitations in data collection techniques. These limitations often come to the forefront when an object is placed on a surface. For example, consider scanning a book lying flat on a table. The scanner will only capture the visible parts of the book, including its cover, spine, and pages if opened. However, the underside, which is resting on the table, remains hidden and therefore unscanned. As a result, the 3D representation of the book will be incomplete.

3.1.5 Transformations

This section presents how we addressed the previously stated transformation requirements and the transformation implementations that accommodate them. The purpose, implementation, limitations, and, if applicable, a comparison to existing methods will be discussed. While the chosen transformations cover a diverse range of scenarios, it is important to acknowledge that real-world situations are infinitely varied. However, we believe that our selection tests some of the most crucial aspects in the field of shape recognition.

Before diving into the each individual transformation, we will introduce the motivation behind the technology choices made for generating the transformed datasets. Our end goal was creating a system that uses that takes a suitable 3D object dataset as input, and generates transformations which fulfils the transformation requirements stated above.

To reach this goal we needed a tool that could automate transformations for a whole dataset. Through looking at available alternatives such as Aspose.3D, OpenTK, Unity [1] and Blender, we found that there was no one tool that could effectively produce the entire range of transformations we needed. We therefore selected two well-known software applications, Unity and Blender, which between them did everything we needed.

These tools were chosen as they are both very versatile, free¹, and provide the tools we needed to implement our transformations. Unity is a well-optimised game engine with a built-in physics engine. Additionally, Unity uses C# as its scripting language which is extensively documented² making development a lot easier [71]. Blender is a open-source 3D creation suite, which can be used for animation, VFX shots, and object creation. Blender was chosen for its boolean operations implementation. In addition, Blender's scripting language is Python which makes scripting very efficient [72]. Blender and Unity also follow standard practices for handling and modifying triangle meshes, making sure the project adheres to the usual norms and conventions in the 3D modeling and design industry.

¹Unity is only free for non-commercial use

²<https://learn.microsoft.com/en-us/dotnet/csharp/>

The benefit of both of these being free for research purposes, is that future researchers can easily access our work and modify it for their own tests. This makes the project a more suitable framework for future work.

3.1.5.1 Basic Matching Performance

Based on the Transformation requirements, basic matching performance has been tested by creating the following transformations:

Rotated objects The primary objective of the rotation transformation is to assess a shape descriptor’s capability to identify objects irrespective of their orientation in 3D space. Objects in real-world situations can be viewed from various angles and orientations, making it essential for a robust shape descriptor to be invariant to these changes.

The objects in the original dataset undergo a series of rotation transformations, resulting in four distinct instances of the entire dataset. For each instance, one of the four types of transformations is applied: rotations along the X-axis, rotations along the Y-axis, rotations along the Z-axis, or simultaneous rotations across all three axes. A random rotation angle, ranging from 0 to 359 degrees, is used for each transformation.

The implementation applies Quaternion Euler rotations to the vertices of the 3D object mesh. Quaternions are mathematical constructs that can represent 3D rotations without suffering from gimbal lock, a problem that occurs with Euler angles when two axes become aligned. The function generates a quaternion rotation for each axis. The rotations are then applied sequentially to each vertex in the mesh by multiplying the quaternion by the vertex position. This operation preserves the relative relationships between vertices while adjusting their positions. After updating the vertex positions, the mesh normals are recalculated to maintain the consistency of the object’s faces and normals. This method enables an accurate evaluation of shape descriptor performance on rotated objects while preserving the original 3D structure’s integrity.

Resized objects The primary objective of the resizing transformation is to evaluate a shape descriptor’s behaviour when provided objects with different scale. It is essential to recognise that the results from shape descriptors are dependent on the users’ interests, and that both high and low matching scores can be suitable, provided they are consistent and significant. For instance, in a real-life scenario, two objects with different sizes can never be the same object; however, if the scanners used to capture the 3D data are not calibrated to provide the same size, it could lead to a situation where the same object appears to have different sizes in the data. This scenario underscores the importance of scale invariance in shape descriptors.

The objects in the original dataset undergo a series of resizing transformations, resulting in several instances of the entire dataset. For each instance, a scale factor is applied to the object’s dimensions, enlarging

or reducing its size while maintaining its shape. To investigate the shape descriptor’s response to different degrees of size change, we chose to examine both small and large change of size. Consequently, we selected scale factors of 0.5, 0.9, 1.1 and 2.0 for each transformation, as these values represent modest and significant changes in size.

The implementation applies scaling transformations to the vertices of the 3D object mesh. The function accepts a mesh and a scale factor as input parameters. Each vertex in the mesh is then multiplied by the scale factor, resulting in a resized object. This approach allows for an accurate evaluation of shape descriptor performance on resized objects while preserving the original 3D structure’s integrity. Next, we recalculated vertex normals for all objects in the dataset. This step was for maintaining consistency in the computation method used for normals throughout the tests.

Moved objects The primary objective of the moved objects transformation is to evaluate a shape descriptor’s behaviour when provided with objects at different positions. In contrast to the scale transformation, it is crucial for a robust shape descriptor to be invariant to translations, as objects in real-world situations can be located at various positions.

In many instances, objects provided to shape descriptors may not align with the original object’s positioning. This misalignment can result from factors such as arbitrary origins chosen by 3D scanners, noise, partial overlap, or the presence of multiple objects. While it is relatively simple to calculate the centroid of an object and centre it, this may not be possible when dealing with the aforementioned scenarios. Therefore, it is essential to test the shape descriptors’ ability to handle translated objects and investigate how translation alone can affect performance. To examine the shape descriptors’ response to different degrees of translation, we chose to explore both small and large displacements. Consequently, we selected distances of 0.2 and 10.0 meter for each transformation, as these values represent modest and significant changes in position, respectively. The direction of translation was chosen randomly to reflect the unpredictable nature of real-world scenarios.

For each translation value that is tested, a new instance of the dataset is created, and the translation value is applied to the object’s x, y, and z position, moving the object in 3D space while maintaining its shape. The implementation applies translation transformations to the vertices of the 3D object mesh. This approach allows for an accurate evaluation of shape descriptor performance on translated objects while preserving the original 3D structure’s integrity.

Mirrored Objects The primary goal of the mirror transformation is to assess a shape descriptor’s capacity to distinguish between original and mirrored objects. In real-world scenarios, objects might appear mirrored due to the presence of another identical but mirrored object. An interesting aspect to consider when analysing the results from mirrored objects is

symmetric objects. These, when mirrored, are identical to their original form. For instance, a perfect sphere or cube will look unaltered when mirrored.

To evaluate a shape descriptor’s ability to handle mirrored objects, we introduce a mirror transformation that reflects the object across a plane. This transformation is applied to the 3D object mesh by altering the scale of the vertices within the mesh. The function takes a mesh and a scale factor as input parameters. The scale factor is applied to only the x-coordinates of the mesh vertices and is set to -1.0 for the mirrored transformation. This effectively inverts the object’s vertices, producing a mirrored object. After implementing the mirror transformation, the mesh normals are recalculated. However, we acknowledge that selecting a different axis or a combination of axes might produce different results and could be considered for future improvements to the study.

The mirror transformation serves to test the shape descriptor’s capacity to differentiate between original and mirrored objects, which is essential for real-world applications where objects may be mirrored due to various factors.

3.1.5.2 Deformation Resistance

Based on the transformation requirements essential for accurate testing, we have assessed Deformation Resistance using the transformations outlined below. Upon reviewing current literature and previous experiments conducted for this project, we have found no prior research that has specifically explored deformation resistance. This absence of previous studies makes the upcoming experiments particularly intriguing. They might potentially fill an existing knowledge gap in our understanding of 3D shape descriptors.

The selection of tests is influenced by time and feasibility considerations. While broader deformation tests — such as those involving bending or stretching deformations — could potentially provide additional insights, they would necessitate extensive resources and exceed the scope of this study. However, the selected twisted and rippled deformation tests provide initial insights and establish a foundation for further research into deformation resistance.

Twisted Deformation of Objects The primary goal of the twisted deformation transformation is to evaluate a shape descriptor’s ability to identify and analyse objects that have undergone twisting deformations. In real-life scenarios objects may undergo twisting, making their recognition and analysis more difficult. For example, a computer vision application may need to recognise soft objects that have been deformed in various ways since the original scan of the object. The twisted deformation dataset focuses on objects with different degrees of twisting, allowing evaluation of the shape descriptor’s performance.

The twisted deformation transformation comprises a 3D object with

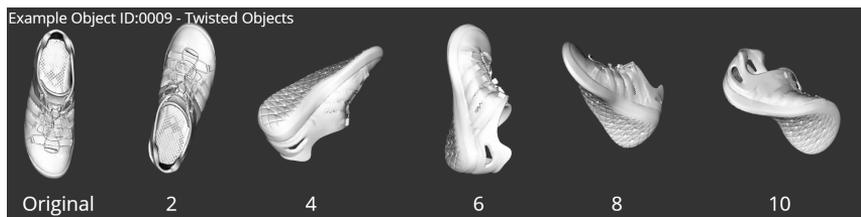


Figure 3.1.2: The original object (left) and its transformed versions with gradually increasing twisting intensity, illustrating the impact of twisted deformation on the object’s surface. Object ID 0009.

twist angles ranging from 2 to 10 degrees. The aim was to simulate twisted deformation as closely as possible to real situations while remaining feasible for generation. A custom script in the Unity engine, along with an asset, was developed and customised for this purpose, using algorithms to manipulate object vertices based on the twist angle and twist size.

The implementation starts by importing the object to the game object class. Once the object is imported, it will undergo twist deformation by adding a TwistDeformer component to each game object, which is responsible for applying the twist deformation. The transformation involves adjusting the object’s vertices based on the twist angle, twist size, and an AnimationCurve for fine-tuning the deformation. To rotate the object, we selected the X-axis, as it was the most feasible option for our time constraints. However, testing on multiple axes is recommended to provide a more comprehensive evaluation of the shape descriptor’s performance.

Figure 3.1.2 presents the original object number 0009 on the left and its transformed versions with gradually increasing twisting intensity. Each transformation showcases the impact of twisted deformation on the object’s surface.

Regardless of the selected axis (X, Y, or Z), the code calculates the new vertex positions by rotating them around the axis using trigonometric functions sine and cosine. In this study a asset for Unity engine named the Mesh and Object Deformers for Unity 3D, which is available for free in the Unity Asset Store, was used to handle and manipulate the 3D object’s twist [73].

Rippled deformation of objects Rippling may occur in applications recognising objects such as, wrinkled clothes or distorted vehicle parts after a collision. A custom Unity script was used for this purpose. This script adjusts the object’s vertices based on the ripple frequency, peak multiplier, and the chosen deformation axis (X-axis in this case) using a sine function. The frequency and peak multiplier values control the number of ripples and their heights, respectively. The peak multiplier values used in this dataset range from 0.005f to 0.030f.

Figure 3.1.3 displays the original object on the left and its transformed versions with gradually increasing ripple effect intensity. Each category of

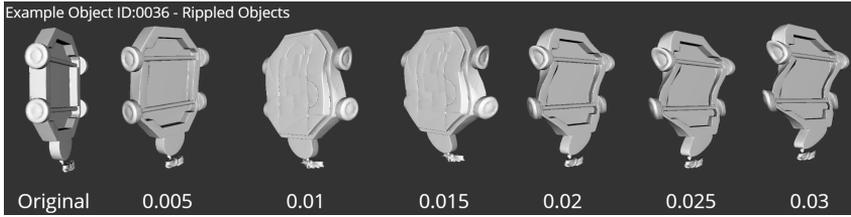


Figure 3.1.3: Original object (left) and its transformed versions with gradually increasing ripple effect intensity, showcasing the impact of ripple deformation on the object’s surface. Object ID 0036

transformation showcases an increase in the ripple effect, demonstrating the impact of the ripple deformation on the object’s surface.

The ripple deformation dataset, consisting of objects with diverse ripple patterns generated using different peak multiplier values, enables a comprehensive evaluation of the shape descriptor’s performance in recognising and analysing objects with rippled surfaces. While the current implementation focuses on the X-axis for deformation, testing on multiple axes is recommended for a more complete assessment. This dataset proves valuable for understanding and improving shape descriptor algorithms in various real-world applications involving objects with ripple deformations.

3.1.5.3 Noise Resistance

The ability to withstand various kinds of noise is an important measure of how well local shape descriptors perform. In this research, we concentrate on two noise resistance evaluations: Gaussian vertex displacement along normals, and deviation and rotation of vertex normals. We chose these tests due to their applicability in real-world situations, their capacity to offer an extensive analysis of shape descriptor performance, and with established research on shape descriptor evaluation.

Other noise types such as speckle, salt-and-pepper, and Poisson noise could also have been implemented for this requirement. However, due to considerations of feasibility and time, they were not incorporated in this study. Although broader, they provide a foundation for future investigations into the performance of shape descriptor algorithms under diverse noise scenarios.

Gaussian vertex displacement along normals The purpose of this transformation is to evaluate the robustness of shape descriptors in matching objects despite the presence of random vertex displacements with Gaussian distribution. This transformation is essential as it simulates noise commonly encountered in real-world applications, such as 3D scanning or object reconstruction. Gaussian noise is a more realistic representation of real-world noise compared to uniform noise, as it models the distribution of errors and imperfections often found in real datasets [74].

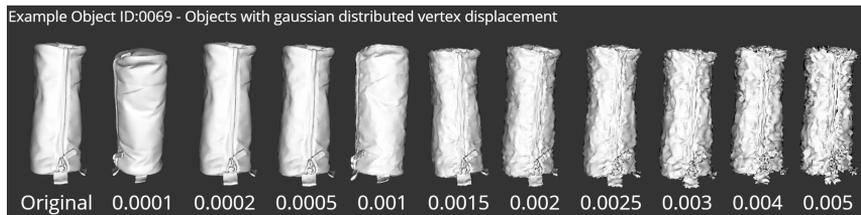


Figure 3.1.4: Original object 0069 and its ten transformed versions with gradually increasing Gaussian distributed vertex displacement, ranging from the original object on the right to the version with the largest noise on the left. Each transformed object is individually tested, allowing for a comprehensive evaluation of shape descriptor performance under varying levels of noise.

For context, existing experiments, such as [11], have tested against Gaussian noise, but only on a limited scale. In their case, they only tested based on 5 and 10 cm of Gaussian distributed noise along the normals. Moreover, they only used 20 randomly selected models from their 56 vehicle database. In contrast, the current evaluation involves testing against 1193 objects with ten different vertex displacement values.

The Gaussian noise, also called normal noise, models the distribution of errors and imperfections often found in real datasets, making it particularly suitable for simulating the noise encountered in real-world scenarios, such as those generated by 3D scanning and object reconstruction processes [74]. Moreover, Gaussian noise takes into account the accumulation of various factors that contribute to the errors in measurement and reconstruction, which result in a normal distribution of deviations. This characteristic of Gaussian noise allows for a more accurate evaluation of the shape descriptors' ability to handle imperfections and noise in real-world applications.

The implementation iterates through each object and applies random displacements to its vertices based on the defined list of displacement ranges. The displaced 3D meshes are saved to the specified directory. The displacement ranges are selected to cover a wide spectrum of possible vertex displacements, ensuring a comprehensive evaluation of the shape descriptors' performance. The values in the list represent the maximum deviation, and the displacement value will be a random number within this range generated using a Gaussian distribution. Vertices are displaced along the direction of the normal, resulting in a more realistic representation of noise. Example can be seen in Figure 3.1.4. Some limitations of this method include the assumption that random displacements follow a Gaussian distribution and that the displacement ranges themselves are representative of real-world noise levels.

Deviation and rotation of vertex normals The purpose of this transformation is to evaluate the performance of shape descriptor al-

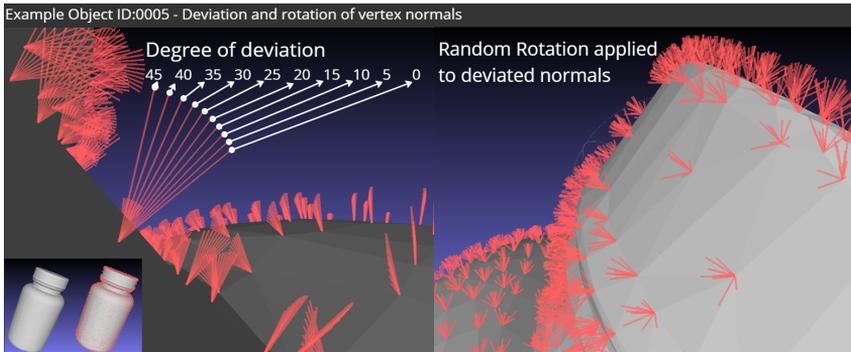


Figure 3.1.5: Simultaneous depiction of deviated normals for ten object categories for illustrative clarity: the left portion displays an overlay of normals with deviations from 0 to 45 degrees, while the right portion presents the normals after undergoing a random rotation in addition to the deviation. The right one is used for evaluating the shape descriptors benchmark. During testing, each object category is assessed individually.

gorithms when handling objects with altered normal vectors. Normal deviations may occur during the process of scanning objects with LiDAR and transforming the outputted point cloud into a mesh, making it essential to assess shape descriptor algorithms' robustness in such scenarios.

To create the transformation a method which computes a new vector by deviating a given input normal vector by specified angles is implemented. This method takes three parameters: an input normal vector, a deviation angle in degrees, and an orientation deviation angle in degrees. The method aligns the normal with the z-axis before applying deviation to simplify the rotation process, easily apply the deviation, and avoid edge cases such as Gimbal lock.

The parameter selection for the transformation determines the range of deviation angles applied to the normals. The delta angles represent a gradual increase in deviation, allowing for a sensitivity analysis. By testing the performance of shape descriptor algorithms on the generated dataset, it is possible to identify trends and patterns that highlight their strengths and weaknesses in handling altered normal vectors.

Figure 3.1.5 demonstrates the deviated normals for the ten object categories, each with deviations ranging from 0 to 45 degrees. The figure presents an overlay of all object categories to illustrate the deviation process effectively. The left portion of the figure displays the deviated normals, while the right portion shows the normals after undergoing a random rotation in addition to the deviation. This rotation varies from 0 to 360 degrees and is unique for each normal in the object. During testing, each object category is assessed individually.

3.1.5.4 Clutter Resistance

Several previous studies have addressed the effects of clutter [10, 11, 13, 14]. In particular, the clutterbox experiment, a method that investigates the relationship between increasing levels of clutter and descriptor performance, has been proposed [13]. In these experiments, objects were clustered in an intersecting manner, forming an artificial environment where objects exist within each other. In addition to this, there are overlapping triangles resulting from the maintained intersections. Building on these previous works, this study proposes evaluating shape descriptors by creating clusters in a more controlled way. This approach is subdivided into two transformations. The initial transformation involves clustering objects together, but without any overlapping. The subsequent transformation creates partially overlapping objects but includes the removal of intersected triangles. The goal of this approach is to better simulate real-world scenarios as they would be captured by a 3D object scanner. It will also provide more control over the factors that affect a shape descriptor algorithm. Further details of these transformations will be provided in the subsections below.

Clustered Objects The Clustered Objects transformation covers the understanding how shape descriptors respond to varying degrees of object clustering. The primary object and the other objects are placed in close proximity to create a clustered 3D environment forming a cluster as depicted in Figure 3.1.6. The goal was to simulate a cluster that was realistic and likely to be encountered, while still remaining feasible to develop. To achieve this, a simulation in the Unity Engine was created, simulating gravitational forces and collisions, allowing the objects to collide and bounce away from each other in a natural way.

In the implementation of this transformation, the meshes are first loaded into a game object class provided by the Unity Engine Library. These objects then receive two extensions to the game object class: rigid bodies and mesh colliders. Rigid bodies enable the objects to be affected by the physics engine, allowing them to interact with other objects and the environment, while mesh colliders define the shape of the objects for collision detection. The clustering process is simulated by applying strong attractive forces between the objects, causing them to cluster tightly together. Additionally, regular gravitational and collision physics are applied to the objects, enabling natural interactions with each other and the environment. The strong force between the objects is gradually reduced over the simulation period, causing the objects to first cluster together intensely and then fall naturally into a pile on the floor 3.1.6.

A timer is used to control the interaction between objects, and it records their final positions within the cluster. After a predefined time, the objects' positions and rotations are saved for the purpose of replicability (see Section 3.1.6), and the combined mesh is generated by merging the objects' meshes into one larger mesh. The dataset was created for five

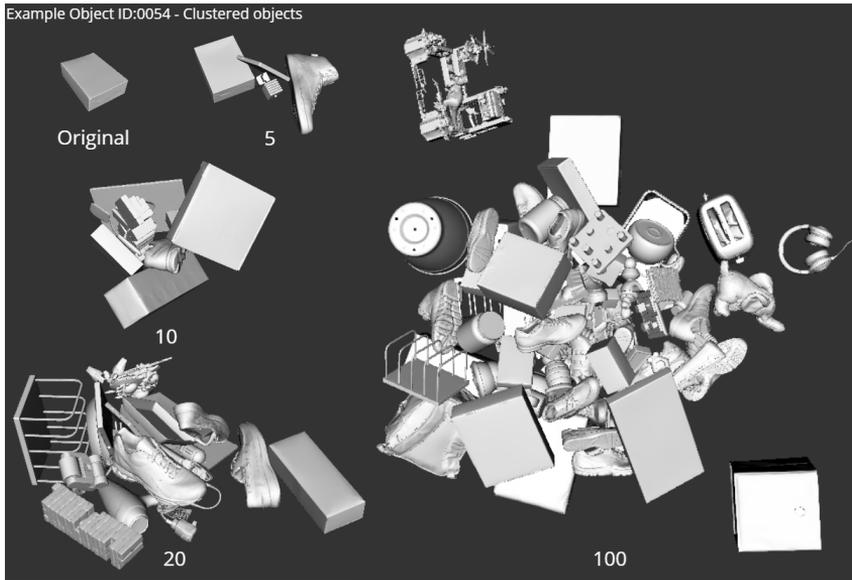


Figure 3.1.6: An example of a clustered arrangement of objects, where the primary object (Object ID 0054) is surrounded by 0, 5, 10, 20 and 100 other objects within the cluster.

different cluster sizes, $n=1$, $n=5$, $n=10$, $n=20$, $n=100$, resulting in a total of 1193 tests for each cluster size.

Partial Overlapping objects The main purpose of the partial overlap transformation is to assess a shape descriptor’s capability to identify objects that are partially overlapped with others. In real world situations, objects may partially overlap, making the separation and analysis of individual objects more challenging. For instance, a robotic arm in an industrial setting might need to recognise and pick up objects from a cluttered environment with items partially overlapping. The sensor scanning these objects captures a large scan with several objects connected in a single outer mesh shell.

To analyse the shape descriptor’s performance in partially overlapping environments, a dataset containing various degrees of object partial overlap was generated. A dataset instance includes one main object and a random selected additional object that is positioned to partially overlap the main object as depicted in Figure 3.1.7. A custom script in Python using Blender was developed for this purpose.

The Python class library implementation begins by importing and processing objects using the Blender Python API. The objects then undergo random rotations and translations, followed by scaling adjustments to control the overlap percentage. Boolean operations create new objects from the overlapping sections, with the Union operation merging their

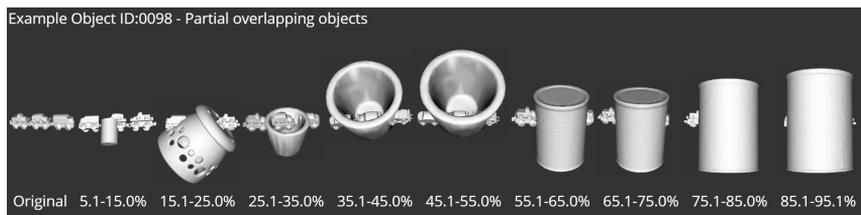


Figure 3.1.7: An illustration of objects with varying degrees of partial overlap. The overlapping regions can be observed as the main object becomes increasingly covered by the additional objects. Object ID 0098

geometry. The algorithm identifies all shared vertices, edges, and faces between the two objects, combines unique vertices, edges, and faces, and removes any internal geometry, retaining only the outer shell[75].

The proportion of altered vertices after overlapping is calculated, and the resulting objects and vertex maps are saved in corresponding subcategory folders based on the specified percentage ranges. A recursive divide-and-conquer algorithm controls the partial overlap process, adjusting the object’s scale to achieve the desired overlap percentage range for each subcategory. These subcategories are defined by the following overlap ranges: (5.1-15.0%), (15.1-25.0%), (25.1-35.0%), (35.1-45.0%), (45.1-55.0%), (55.1-65.0%), (65.1-75.0%), (75.1-85.0%), and (85.1-95.1%). This approach allows a comprehensive evaluation of shape descriptor performance on partially overlapping objects.

However, it is important to note that only manifold meshes guarantee proper results. Other cases, particularly open meshes and non-manifold meshes without self-intersections, may usually work well but could produce odd glitches and artefacts in certain scenarios. Although some objects may not perfectly represent real-life scenarios, they still serve their purpose of testing shape descriptors ability to detect partially overlapping objects.

3.1.5.5 Oclusions and Incomplete Surfaces

In this benchmark have we focused on two tests for this requirement: objects with holes and partial surface visibility. These tests were selected based on their relevance to real-world scenarios and their ability to provide a comprehensive assessment of shape descriptor performance under challenging conditions.

Object with Holes The Object with Holes transformation is designed to simulate the effect of missing or incomplete surface data by introducing holes in the objects’ mesh. The purpose of this transformation is to evaluate the performance of local shape descriptors under the presence of incomplete surfaces. In real-world scenarios, objects may have missing data due to various reasons, such as occlusion, sensor noise, limited view angles, or physical damage.

This transformation is implemented in the same manner as partial overlapping objects, using Blender. This is made possible thanks to Blender's advanced Boolean operations, specifically the Difference operation, used in this case. The Difference operation requires two input meshes: the target mesh and a subtraction mesh. The subtraction mesh subtracts its overlapping surface from the target mesh, leaving everything outside of the target mesh intact. In this experiment, we are using a very long cylinder to subtract holes in the target mesh.

However, Blender's Boolean operations primary limitation is its reliance on objects being manifold. Non-manifold or open meshes can lead to occasional glitches and artefacts. According to Blender's documentation, only manifold meshes are guaranteed to provide accurate results [75]. While non-manifold meshes without self-intersections typically work well, they may produce odd glitches and artefacts in some cases. Additionally, the transformation does not account for the possible variations in hole shape, size, or distribution that might be encountered in real-world scenarios, which may limit its overall effectiveness.

In this implementation, the number of holes and their placement are dynamically adjusted based on the object's size and position. The process begins by calculating the initial and maximum radii for the cylinders that will be used to create the holes. These radii are determined by taking a fraction of the average dimension of the object. The code then generates a grid of cylinders, where the space between them is determined by a function which ensures that the space between cylinders increases as the radius of the cylinder decreases. The code then performs the 'Difference' operation, between the original object and the grid of cylinders. This operation creates holes in the original object at the intersection points with the cylinders (see Figure 3.1.8). After creating the holes, the code generates a vertex mapping that maps the vertices of the original object to the vertices of the modified object. The script performs a series of transformations and calculations to generate the vertex map and categorises the objects based on the percentage of vertices removed. This categorisation ranges from 10 to 90 percent.

The objects with the lowest vertex count, specifically 21 objects, were not included in this process due to the limitations of the re-meshing algorithm used. When a hole is created, a high number of faces are required to maintain the round shape, leading to issues with the algorithm when the object has too few vertices. This is something that can be improved in the future but was not prioritised due to the amount of work required and the limited number of objects affected.

The code also categorises the modified objects based on the percentage of changed vertices. It uses a divide-and-conquer approach to find the correct radius for each subcategory by having a function that recursively calls on itself. This function adjusts the radius until the correct percentage range for each subcategory is achieved. It is important to note that some objects might be skipped if the maximum number of attempts is reached.

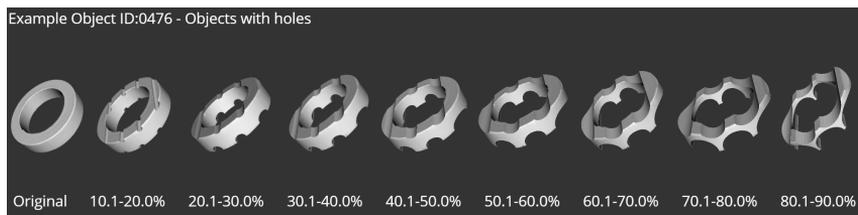


Figure 3.1.8: Illustration of object number 0476 in its original state and through various stages of the "Object with Holes" transformation, showcasing the gradual increase in the number and size of holes in the object's surface.

Partial Surface Visibility The purpose of this transformation is to evaluate the robustness of local shape descriptors when handling the partial visibility of 3D objects. In real-world scenarios, sensors capturing 3D objects might not fully capture the entire object due to occlusions, limited sensor range, or the object's position relative to the sensor. To simulate such scenarios and evaluate the performance of local shape descriptors under partial surface visibility, we introduce the Partial Surface Visibility dataset. This method is based on generating new meshes consisting only of the faces visible from specific points, mimicking the partial visibility that may occur in real-world conditions.

The implementation of the Partial Surface Visibility transformations employs Unity raycasting to create the dataset, as illustrated in Figure 3.1.9. Each 3D model is divided into individual faces, which are then converted into separate game objects with colliders. These colliders are what makes the faces detectable by the Unity raycasting class. To ensure that the rays hit all the faces and each face is shot by the raycasting twice, the implementation casts rays from two predetermined points towards the 3D model's faces. For each face, the implementation calculates the directions from the ray starting points to the triangle formed by the face's vertices. These directions are then used to cast rays from the starting points.

If a ray intersects a face collider, it signifies that the face has been hit from the specified viewpoint, as demonstrated with the two androids in Figure 3.1.9. The raycasting class will also tell us if the hit was the first intersected face or not, this enables us to only pick the first intersection. The implementation then adds the vertices of the hit face to a list of new vertices, ensuring that each vertex is only added once. After all rays have been cast and hit faces have been identified, the implementation creates a new mesh that includes only the vertices of the visible faces. It then calculates the new triangles that connect these vertices, based on the original mesh's triangles. The new mesh represents the visible portion of the original 3D model and is saved to a file for further analysis. By using Unity raycasting and properly configuring colliders, the implementation can simulate real-world sensor limitations and partial visibility scenarios,

allowing for the evaluation of local shape descriptors' robustness in various conditions.

One limitation of the current implementation is that it only tests partial visibility from one angle (Two angles placed close together). Future work could include testing the transformation with more angles, varying sensor-object distances, and incorporating more complex occlusion patterns.

During our research and development related to the creation of this transformation, we encountered two issues. The first issue is that many triangles that should have been hit were not. To address this, we transformed each triangle into a triangular prism by duplicating each face and shifting it slightly along the raycast direction. This technique significantly improved the Unity engine's physics and collision detection system's ability to detect hits.

We also discovered that employing raycasting from two distinct angles effectively addresses the issue of specific faces not being hit by a single angle, as illustrated in the left and right Android object in Figure 3.1.9. However, despite the problem being mitigated, two remaining inaccuracies may arise from numerical issues, such as floating-point errors, limitations of Unity's physics engine and its raycasting method, or certain edge cases where the angles approach zero degrees, and our mathematical functions fail to handle them appropriately.

The raycasting process is represented by the red lines (ray hits) and white lines (ray misses), as seen in Figure 3.1.9. The white misses predominantly impact objects at the outer edges rather than those in the centre. These white lines indicate a minor issue that we were unable to resolve; nonetheless, it has minimal impact as it only affects the object's outer edges and could be attributed to minute numerical inaccuracies related to the choice of datatype or collision detection accuracy in the Unity engine.

The other issue, evident in the left Android object, reveals the number of triangles not detected in the object's centre despite being visible from the raycast starting point. This problem was addressed by implementing two raycast starting points and performing the raycast twice. Note that the few triangles not hit in the right Android object are not visible from the raycast shooting angles and that shooting from two angles as far we know have worked perfectly.

3.1.6 Replicability

We have facilitated for replicability for all transformations by making sure the code is available for further development and research. This allows other researchers to expand our benchmark with additional shape descriptors. Moreover, we have employed pseudo-randomisation with a fixed seed to guarantee that all datasets can be regenerated with the same random state. This approach ensures that all random functions will produce the same results each time, preserving the consistency and reliability of the all generated data. The random state is saved regularly and has been

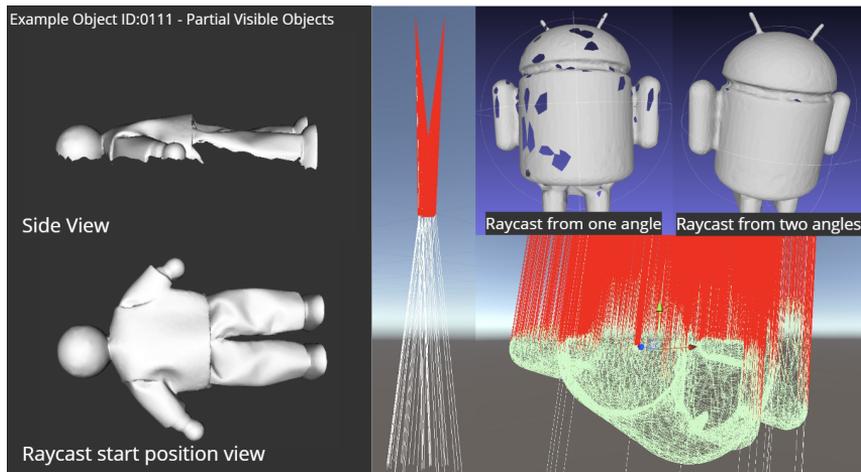


Figure 3.1.9: Illustration of the Partial Surface Visibility transformation using Unity raycasting. The image shows a side view and a raycast start position view of an object, along with the raycast path traced for visualization. Two androids are also shown, one shot from one angle and the other one demonstrating raycasting from two closely positioned points.

incorporated into all recovery functions that are used to skip already generated objects when restarting the generator midway through processing a transformation. By taking these steps, we have made our benchmark implementation replicable, fostering transparency and encouraging further research and development in the field of local shape descriptor benchmarking.

3.1.7 Transformed Objects and Metadata

In the process of generating transformations, each transformed object is saved as a new file together with its corresponding metadata. The metadata is generated by mapping the vertex indices from the original object to the corresponding indices in the transformed object. This mapping process allows for accurate evaluation of the descriptors' performance in matching the original and transformed objects. In the case of objects with missing parts, the mapping is achieved by finding all matching vertices between the original and transformed objects. For objects that are resized or have their vertex values altered in any way, a one-to-one mapping is maintained by keeping the vertex indices in the exact same order. This ensures that the metadata accurately reflects the relationship between the original and transformed objects, enabling a comprehensive evaluation of the local shape descriptors' performance.

3.2 Performance Data Generation

One of the more difficult challenges when trying to evaluate the performance of shape descriptors, as previously pointed out, is to ensure fairness in all tests. If we accept any factors that might lead to randomness or advantages for any of the shape descriptors, we can't confidently extract any conclusions from our results. These factors can be small, such as modifying the shape descriptors' parameters, to more general ones which includes how to set up the testing environment. Therefore, in order to handle these challenges the following requirements are set:

1. Create a method for fairly evaluating the performance of different shape descriptors.
2. Find the optimal parameters for each shape descriptor, to ensure their best performance.
3. Make it easy for future researchers to implement their own shape descriptors and distance functions.

These requirements are general enough to give some creative freedom in order to produce new methods, and still being concrete enough to be able to directly answer.

3.2.1 Comparing Descriptors

To be able to answer the first requirement we need to look at the previous evaluation methods and find areas of improvement, this will help us find a foundation which we can base our new method on. As previously introduced, each shape descriptor has their own methods for generating descriptors which have been compared using a set of different distance functions. Additionally, each of the papers have their own methods for evaluating the performance of their shape descriptors. Their methods followed similar principles, by first creating two descriptor sets from an original and a comparison object. Then matching the descriptors to each other using a distance function. As discussed, this introduced randomness into the tests which has led to the need for a more general evaluation method in order to ensure fairness while testing new shape descriptors.

The first possible approach we considered was generating a similarity score within in the range of 0 to 1, which represented how similar a shape descriptor considered two objects. To counteract the aforementioned self-similarity problems, we used a ground truth to correctly map descriptors from one object to another. This mapping was created for all the different transformations, and acted as a summary of changes from the transformation. This approach, for any given shape descriptor and distance function combination, would consist of the following steps:

1. Choose two objects; one original object and one to compare it with.

2. Calculate two separate descriptor sets for each of the objects.
3. Using the ground truth, create a set of distances for the corresponding descriptors.
4. Find the total average distance, and convert it into a similarity score.

The original idea was that with this score we could easily compare the performance difference between the different shape descriptors, as they all ended up with values in the same range. However, determining what makes one distance score better than another turned out to be a difficult task. This resulted from the values from different distance functions representing entirely different things. For example, the Euclidean distance representing how far one point is to another in Euclidean space, while the Pearson Correlation represents a linear correlation between two sets of data. To further exemplify, if the average Euclidean distance between two objects is 12 and the Pearson Correlation says 0.8, how can we determine which is better?

To solve this challenge we experimented with only using one distance function, as this would allow us to create a general score for the different shape descriptors. However, this also presented some problems. First off, as previously presented the different shape descriptors have a preference for which distance function to use. Which can lead to some shape descriptors getting an advantage if their distance function of choice was chosen. Secondly, the output of the different shape descriptors don't calculate the shape and features of an object in the same way. This can be exemplified by comparing the outputs of QUICCI and SI. The QUICCI shape descriptor represents a given points intersections through binary data, while SI counts the amount of points within a volume around given points in the object. Since a distance function measures the difference in two values, if one of these shape descriptors always outputs lower numbers than the other it will seem like it always performs the best.

With this we needed to consider a new method for comparing the shape descriptors' performance. In order to comply with the requirements of creating a fair evaluation method and ensuring that the shape descriptors would perform at their best, we decided that another solution could be defining a floor which would represent when a given performance is considered bad. As the distance between two descriptors could potentially be infinite, defining a floor would aid us in identifying good results. A possible downside of using this approach is that we cannot directly compare the performance of the shape descriptors to each other, since the results depend on each individual shape descriptor and distance function combination. However, this will give us a good indication for how well each shape descriptors performs in different environments. In future studies, a method for directly evaluating the performance of each shape descriptor to each other would be a valuable contribution.

To create this floor we needed to get insight into what a bad matching performance for each shape descriptor and distance function combination

would look like. To do this we created some requirements for a dataset, which would ensure us that we got as bad results as possible:

- Items should not be the same.
- Items should represent a range of different categories (shoes, boxes, toys).

With this we hand-picked a dataset of 9 dissimilar objects from different categories. Regardless of its small size, this dataset would give a good indication of how we could expect a bad result for each shape descriptor to be. Additionally, observing how the different shape descriptors act when two completely equal objects are used is also useful. This will give us a realistic range of where we can expect good performances to be within.

Following the creation of the dataset, we created a testing environment to register how the different shape descriptors performed when given two different objects. Each combination of dissimilar objects were sent through the following steps, which were the same for all shape descriptor and distance function combinations:

1. Choose two dissimilar objects.
2. Separately compute the descriptors for each of the objects, with the chosen shape descriptor.
3. Calculate a set of distances for pairs of descriptors from the two objects³.

The same steps are also followed when looking at equal objects. By looking at the distribution of distances for each set of objects, one can get an insight into how a bad matching performance would look like for a given shape descriptor. The results for unequal objects can be seen in Figure 3.2.1⁴ and Table 3.2.1, and the results when using equal objects can be seen in Table 3.2.2. These results show that the distribution of distances keep the same throughout the tests, which makes it a bit hard to define where the floor should be. However, looking at the results we set the floor to be the average of the total distances calculated from the tests. This gives an indication of where an average distance for a test should be in order to be considered to be a good performance. We also considered using the 90th percentile of the worst distances to represent the floor. As we could more confidently say that any distance above that floor would be noise, but when we tried to use it with the transformed objects this floor was hardly ever reached. This led to using the average for the sake of comparing the different shape descriptors. For the reason that if a shape descriptor produces a distance above the average, we can conclude that

³No ground-truth is needed as the objects don't have any corresponding descriptors

⁴*Note:* SI's best possible distance is 1 and every distance below that is worse, while the other's best distance is 0. This results in its distances being a good match when they are over the floor, while the others are good when they are under.

Shape Descriptor	Average Distance	90th percentile
RICI (3.2.1a)	52.7	148.0
QUICCI (3.2.1b)	513	1027.0
SI (3.2.1c)	0.49	0.19
3DSC (3.2.1d)	3.6	8.0
FPFH (3.2.1e)	342.4	982.4

Table 3.2.1: Average Distance and 90th percentile for each shape descriptor

Shape Descriptor	Average Distance
RICI	0.0
QUICCI	0.0
SI	1.0
3DSC	0.0
FPFH	0.0

Table 3.2.2: Average Distance for test on equal objects

result being an above average poor performance. On the other hand, when we compared equal objects we see that all the shape descriptors generate perfect scores. This aids in defining the range of good performances, as we now know that it is possible to get a perfect score for all the shape descriptors.

Additionally, even though the distribution of distances are equal throughout the tests we noticed that the variation of distances within each test is quite large. This could indicate that a bad matching performance is not only defined by the previously created floor, but also by the variation of distances calculated. We expect a good matching performance to have a smaller variation, as most of the distances should be close to the best possible distance. With this we came to the conclusion that the standard deviation for each test should also be taken into account, together with the floor. Using them together will prevent the case of where two descriptors create consistently bad distances, which the standard deviation would interpret as a good result. Using the same test as before, each shape descriptors' average standard deviation was calculated (Figure 3.2.2) and was used together with the floor to determine if a descriptor had a good match or not.

With this we propose a new solution for fairly comparing the performance of shape descriptors, which we used in our final experiments. First off we start by finding the average poor performance for each shape descriptor and distance function combination. By using a dataset of dissimilar objects we calculate the distance between all pairs, and take the average of all of them. This average defines a floor which represents a range of where a good performance for a given shape descriptor should be. Then by calculating the total average distance between two objects, using the

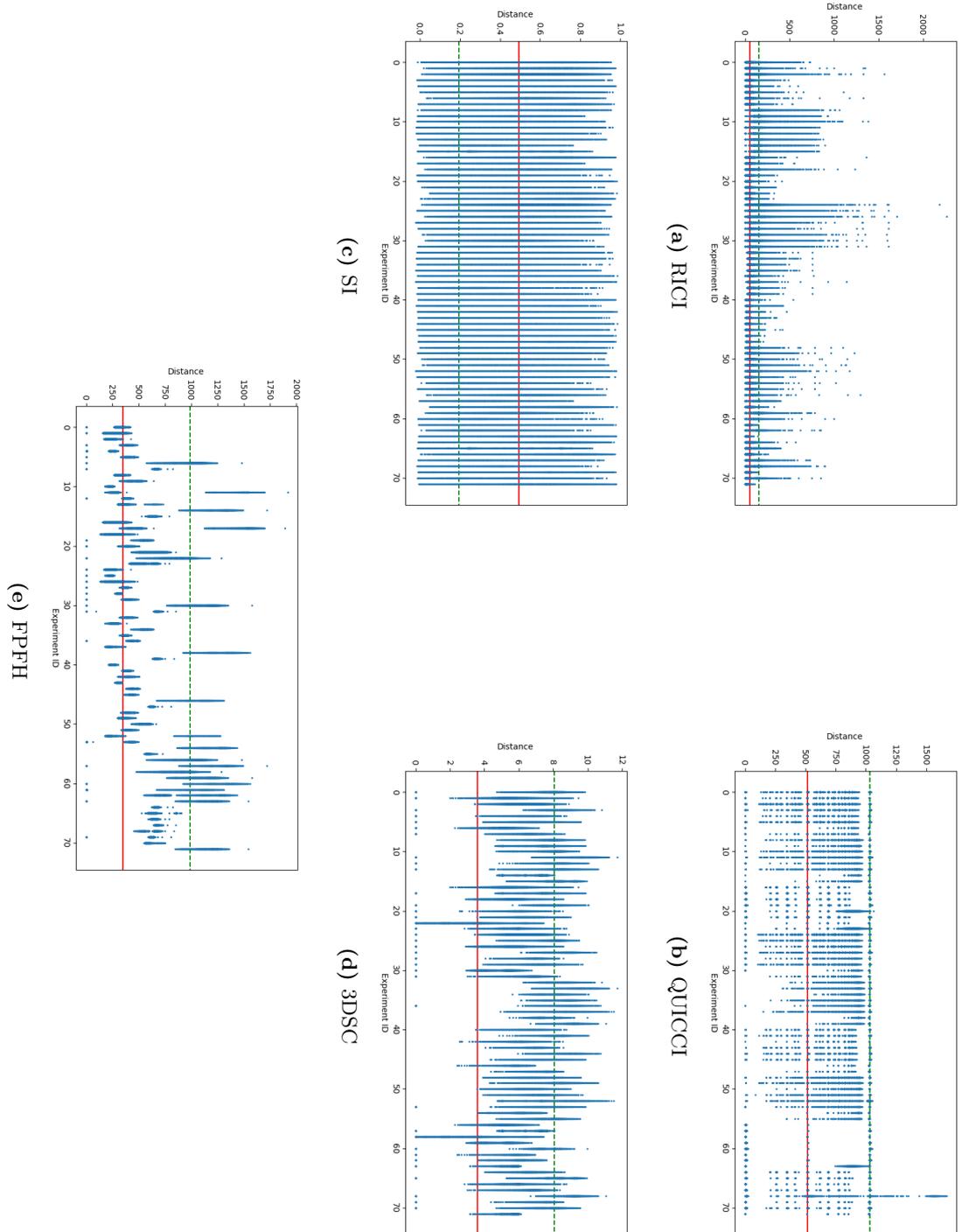
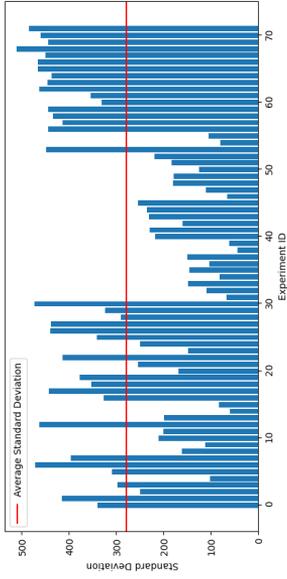
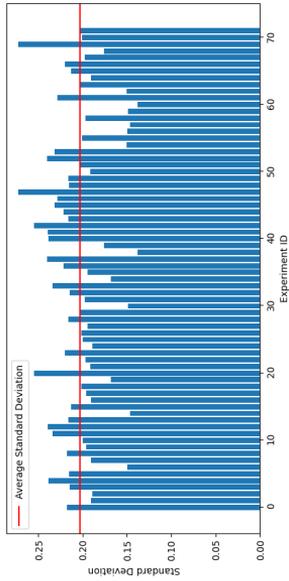


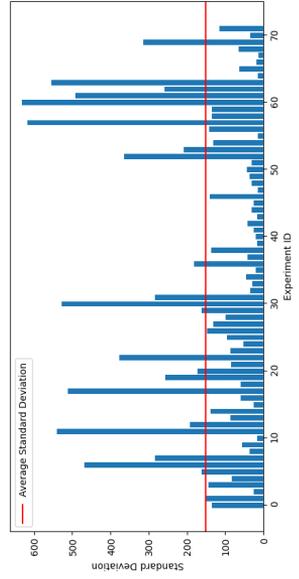
Figure 3.2.1: Floor tests for the shape descriptors. Red line = Average distance, Green line = 90th percentile



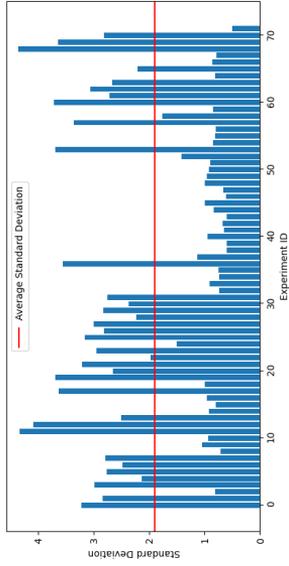
(a) RIC1: 70.4 avg



(b) QUCCI: 278.7 avg



(c) SI: 0.2 avg



(d) 3DSC: 1.9 avg

(e) FPFH: 151.8 avg

Figure 3.2.2: Standard Deviation distribution for the shape descriptors

same steps as when we tried to create a similarity score, we compare it to the floor and determine if the performance is good or poor.

3.2.2 Descriptor Parameters

As previously introduced, each shape descriptor has a set of parameters that can be modified in order to optimise the shape descriptors for different uses. The introduction papers and later papers have presented each of the shape descriptors' optimal parameters which we will base our experiments on. All the shape descriptors have at least some parameters in common, while others have a couple that are specific to them. What these parameters are and how their values were chosen will be further discussed here.

3.2.2.1 Support Radius

The Support Radius is the width and height from a given point, which defines the area a shape descriptor takes into consideration when performing calculations. It is also a parameters shared between all of the shape descriptors. Choosing the correct radius is important, as it needs to be able to pick up on smaller features of an object which makes that object unique as well as larger ones. In a perfect world, the radius would change from point to point in an object depending on the level of detail. For example on an aeroplane where a larger support radius can be used on the wings while a smaller one on the buttons inside the cockpit. This would ensure us that all the details which makes an object unique are taken into account. How to implement such a feature is a thesis in its own right, and would therefore be a significant contribution in future work. In light of this, a radius which can be used for all the objects in the dataset was chosen, as it ensures that the shape descriptors test equal parts of the objects. This choice depends on that all the objects in the dataset are of similar size. The radius was chosen based on the tests done in the 3DSC paper, where they performed extensive experimentation to figure out the optimal parameters [11]. For all the shape descriptors a support radius of 2.5 units was chosen. In Figure 3.2.3 you can see an approximation of the area this support radius would represent, as most objects in the dataset are the same size this would be the case for all objects.

3.2.2.2 Point Cloud Conversion Parameters

As the chosen dataset consists of triangle meshes, some of the shape descriptors need to have them converted into point clouds. This conversion has two parameters; *Point Cloud Sample Count* and *Random Seed*⁵. The sample count represents how many points the point cloud will have, while the seed enables us to create the exact same point cloud every time. To ensure that the objects using point clouds are not affected by noise,

⁵Chosen value for all tests: 4917133789385064

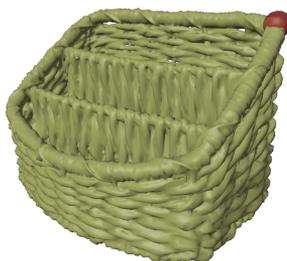


Figure 3.2.3: Red sphere representing the area taken into consideration with a 2.5 support radius

Parameter	Value
RICI/SI Resolution	64*64
QUICCI Resolution	63*64
3DSC Horizontal Slice Count	15
3DSC Vertical Slice Count	11
3DSC Layer Count	12
FPFH Bins Per Feature	11

Table 3.2.3: Bin sizes and resolutions of the shape descriptors

a sample count which is high enough to represent all the objects in the dataset needs to be chosen, but not too high as it affects the generation rate of the descriptors [13]. In the end a value of 200,000 was chosen, as it through our tests had showed to provide a good combination between the two requirements.

3.2.2.3 Resolution

All the shape descriptors also have parameters which define the size of their output. These are usually referred to as resolution or bin sizes. The ones chosen for this thesis are based on the testing environment in the QUICCI paper⁶ [14] and can be seen in table 3.2.3.

3.2.2.4 Support Angle

The SI shape descriptor has the support angle parameter, which represents the maximum angle between a given point's normal direction and the normals of points within the support radius. Based on the original spin image paper, this parameter was set to 60° [10].

⁶Which are further based on the parameters chosen in the shape descriptors' original papers

3.2.2.5 Minimum Support Radius and Point Density

The 3DSC shape descriptor has a minimum support radius, in addition to the maximum. The value of this was set to 0.1 units, which was also based on the experiments done in their paper. Following this, 3DSC's point density radius was set to 0.2 [11].

3.2.3 Implementation

Our last requirement was to make sure that it was easy for future researchers to utilise our work. In order to approach this, we decided to follow the modifiability software quality. Which refers to the degree of which a software system can be modified, adapted, and extended. This would enable future researchers to add their own shape descriptors and distance functions, as well as changing previous implementations. This requirement would also entail making the code understandable and creating a logical structure.

In order to make the code as understandable as possible, we made sure to use precise variable names and follow standard object oriented principles. With this anyone with some experience with object oriented programming would be able to read the code. We also structured the project by separating out larger parts of the software, such as creating descriptors and matching descriptors to each other using metadata. This would help future researchers to find the code they are looking for quicker, without going through one large file.

Our implementation was built on top of the already existing *libShapeDescriptor*⁷, which is implemented using C++ and CUDA. This library includes implementations for all the shape descriptors and distance functions, as well as tools for importing objects. CUDA provides GPU acceleration of computations, which speeds up the amount of time spent creating and comparing descriptors [76]. Additionally, C++ is a low-level language with great performance and efficiency. This lets us go through larger datasets at the time, without taking too much extra time, which leads to our results being more accurate. Our contributions to this library has been making a benchmarking tool that utilises the aforementioned shape descriptor implementations, and lets users send in datasets in order to generate performance data for the different shape descriptors.

The implementation also utilised the *libShapeDescriptors*'s features for ensuring replicability when converting triangle meshes to point clouds. This is done through defining a random seed, which ensures that the points in the point cloud always are put in the same spot when the same seed is used. This ensures that it's possible to generate the same results every time when using triangle meshes as the dataset.

The benchmarking tool was made to compare two large dataset folders, however it is also possible to directly compare two objects. The output, in the case of comparing two folders, will be a large JSON file containing all

⁷<https://github.com/bartvbl/libShapeDescriptor>

relevant data for evaluating the shape descriptors' performances. JSON is very versatile, which makes it easy to extract the data needed after the fact and create in-depth comparisons. The JSON includes data about what transformation has been tested, categories within each transformation, and all the objects. Each object has data about:

- The total average distance for each shape descriptor
- The standard deviation of distances for each shape descriptor
- The generation time and the time it took to compare the descriptors
- Vertex and face counts for the original and comparison objects

These data points will let us perform the necessary analysis based on our proposed method, as well as giving us some freedom in how we choose to visualise the data.

All tests were run on the same computer to ensure that all data are based on the same premises. Additionally, as the project is based on the RICI test environment [13] it also utilises the same random seed system, which leads to reproducible results. The computer uses the NVIDIA GeForce RTX 3090, AMD Ryzen 9 3900X 12-Core CPU, and has 32 GB of RAM.

3.3 Data Analysis

In order to gather any interesting conclusions from the performance data, it is important to define how to visualise it. This will help us to get a better insight into what the data represents. Additionally, as a part of a benchmark it is important to standardise how the shape descriptors are being ranked, which will help researchers see how well their new shape descriptors are performing. With this we define the following requirement for the data analysis process:

1. Create a visualisation for the performance data, which lets researchers easily see how well their shape descriptors performs.

First off we need to analyse the data from the performance data generation process. As previously mentioned, for each test we have data representing the total average distance, standard deviation, face and vertex counts, and generation and comparison time. We also have the previously calculated distance floor and standard deviation boundary for each shape descriptor. Additionally, some of the object transformations include multiple categories, while others only one. With this we need create a visualisation which conveys how the average distance and standard deviation for a test compares to the set floors, and also accommodates for different amounts categories.

In the case of when a transformation has multiple categories, there would potentially be over 1000 tests per category. If we wanted to visualise

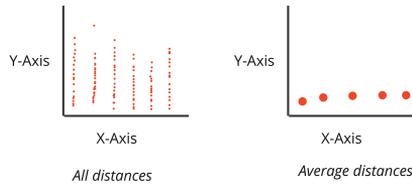


Figure 3.3.1: Examples of diagrams with all distances per category and average

all specific tests the diagram might seem very messy, and it could become hard to extract any conclusions from it. Therefore a decision had to be made for how we wanted to show the performance for each category. There were two possible options, show all the distances from the tests for a given category or calculate an average for each category. As seen in Figure 3.3.1, we see that by using only the average per category we get a much better understanding of how the performance changes over the different categories. However, we lose insight into the variation of distances, but this insight will be minimised by also showing the standard deviation as it gives a good indication of how the different the distances are.

When there is only one category only showing one average distance wouldn't really tell us much. Therefore it is possible to show all the different distances for the tests, but we need to find a good metric to sort the tests by. This can show how the distances for a shape descriptor changes based on the test. There are several good alternatives for this, such as the vertex count of the object tested, individually based on the distance, and based on the standard deviation. In the end we chose to sort individually based on the average distance for a test, as it would give a clear overview over all the tests and it enables us to approximate how the shape descriptors would normally perform in the given environment.

3.3.1 Implementation

The diagrams were created using the Python library Matplotlib. Matplotlib is a library for creating static, animated, and interactive diagrams, and additionally provides a lot of resources for customising these [77]. We had also experimented with using software such as Microsoft Excel and Google Sheets, which worked well with smaller datasets, however their performance and usability was tremendously impacted when the datasets increased in size. We therefore chose Matplotlib to produce diagrams as it provides tools for optimising performance, and because of its simple to use API [78]. Using Matplotlib also aids researchers as they can use our scripts to visualise their generated performance data, which helps when comparing their new shape descriptors.

3.4 Architecture Overview

An overview over the standardised benchmark can be seen in Figure 3.4.1⁸. This diagram shows how the data flows through the different processes of the architecture, and how the different parts communicate. Further discussion of the reasoning behind the architecture will be done in section 5.2. The different GitHub repositories for the parts can be found in appendix A.

⁸The benchmark is built upon code from the Autumn 2022 preparatory project [1]

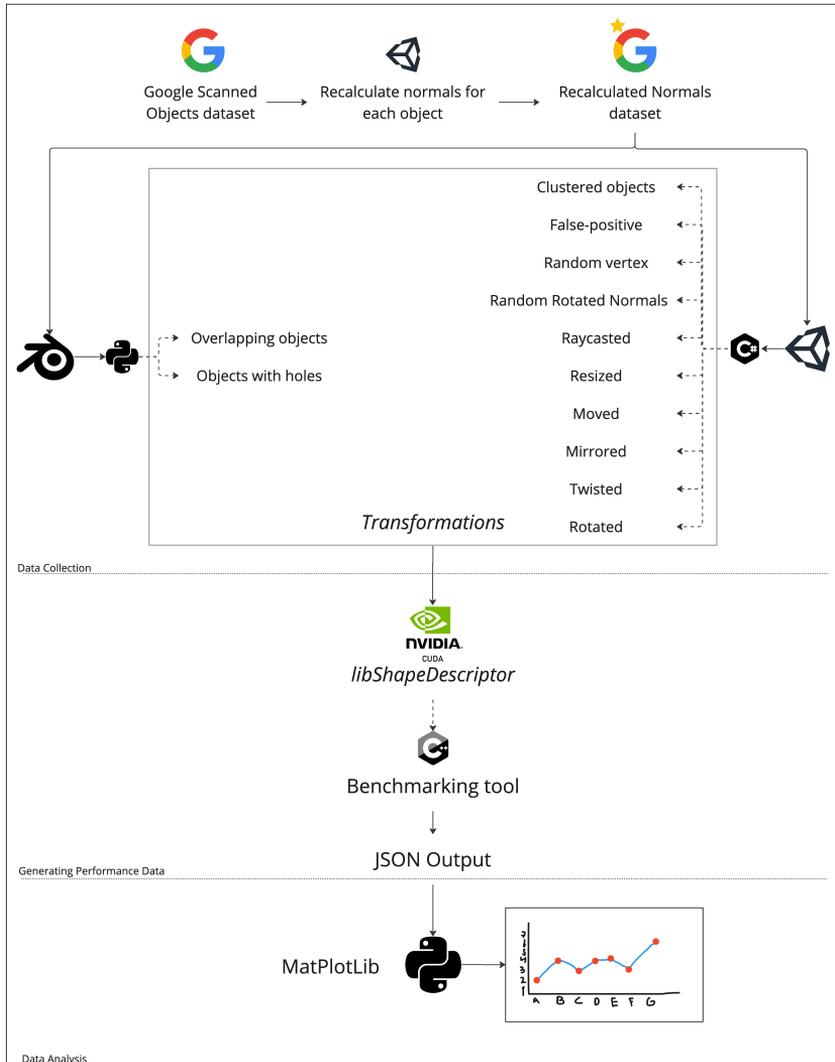


Figure 3.4.1: Overview of Benchmark architecture

RESULTS

In this chapter the results for all the tests done will be presented and explained¹. The results will also be further discussed in the next chapter. Each category contains $n=1193$ objects, and the amount of categories differ from transformation to transformation. The amount of time spent on running each test can be seen in Table 4.0.1. Some tests were run in parallel, which reduced the total amount of time spent on running all the tests.

As previously mentioned, each shape descriptors' results are represented using an average distance and an average standard deviation for a given category. These results are compared to the previously calculated average distances and standard deviations when two unequal objects were compared (values for each shape descriptor can be seen in Table 3.2.1 and Figure 3.2.2). These will be referred to as a shape descriptors poor matching floor and standard deviation boundary, respectively. If either the average distance or average standard deviation crosses their given boundary, it indicates a poor performance. However, if the opposite is true it indicates a good performance.

Each shape descriptor and distance function combination has their own unique output. This can be represented through ranges, which show what values are optimal for the combinations:

- RICI - CRD | QUICCI - WHD | 3DSC - Euclidean | FPFH - Euclidean:
 - Best: 0
 - No maximum value, but the higher it is the worse it is.

- SI - Pearson:
 - Best: 1

¹Results and code used to produce these graphs can be found in our Benchmark repository <https://github.com/masteroppgaven/Benchmark>

Test	Time (hours)
Rotated Objects	23
Resized Objects	23
Moved	13
Mirrored	6
Twisted Deformation	33
Rippled Deformation	38
Gaussian Vertex Displacement	48
Deviation and Rotation of Vertex Normals	31
Clustered Objects	103
Partial Overlap	59
Objects with Holes	52
Partial Surface Visibility	6

Table 4.0.1: Amount of time spent running each test, in hours

- Worst: 0 (would be -1, however since all values generated from the shape descriptors are positive it is not possible to go lower than 0)

The following diagrams will show how the shape descriptors performed in the different simulated real world scenarios.

4.1 Generation Times

A visualisation of the generation time for each shape descriptor can be seen in Figure 4.2.1 and their average generation time can be seen in Table 4.2.1. These results show that both RICI and QUICCI are considerably faster than the rest. Only SI is close to their performance, but is still four times slower. We also see that FPFH has a large spike at the end, when the vertex counts are higher. We got similar results to what van Blokland et al. got in his tests in the QUICCI paper [14].

4.2 Comparison Times

The average time used to compare the descriptors can be seen in Table 4.2.2². As expected the 3DSC shape descriptor uses the most time, as it has to perform more comparisons than the other shape descriptors. RICI and SI has similar results, and FPFH and QUICCI perform the quickest comparisons given their small descriptor sizes. van Blokland et al. also got the same results in his QUICCI paper [14].

²*Note:* These results are based on the time used when calculating on the CPU, as these show the differences more clearly. The results can be viewed at https://docs.google.com/spreadsheets/d/1mTGLzFWl7z1YiuJ_t7dx6D9xkwdpI4roUY_xLKNdD6c/edit?usp=sharing

Shape Descriptor	Average Generation Time (s)
RICI	0.09060
QUICCI	0.09341
SI	0.39163
3DSC	2.67576
FPFH	6.37923

Table 4.2.1: Average generation time for each descriptor

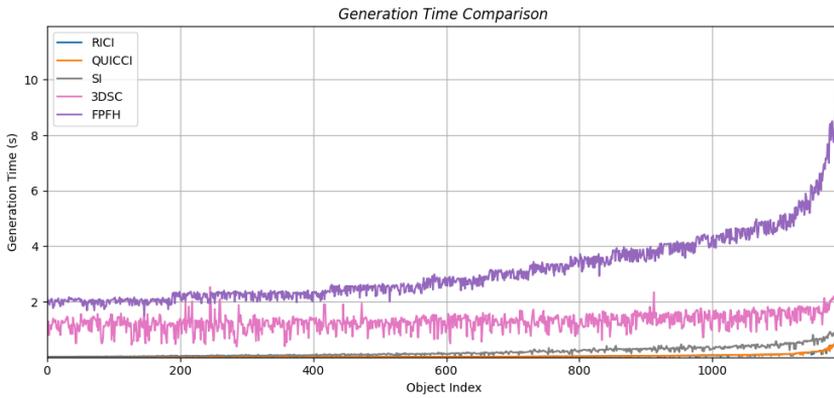


Figure 4.2.1: Generation time sorted on object's vertex count from low to high

Shape Descriptor	Average Comparison Time (s)
RICI	0.05757
QUICCI	0.01049
SI	0.05596
3DSC	1.49970
FPFH	0.01185

Table 4.2.2: Average comparison time for each descriptor

4.3 Basic Matching Performance

The Basic Matching Performance transformations will test the different shape descriptors abilities to identify objects which have had simple transformations applied to them. Such as, the objects being rotated or resized.

4.3.1 Rotated Objects

The results from the Rotated Objects tests can be seen in Figure 4.3.1. All the shape descriptors, except 3DSC, produces perfect results for objects that are rotated. We see that 3DSC is somewhat affected by how the object is rotated, but not by much. Additionally, another interesting observation is that when the object is rotated in all three axes the distance ends up being lower compared to when the object is only rotated in one axis.

4.3.2 Resized Objects

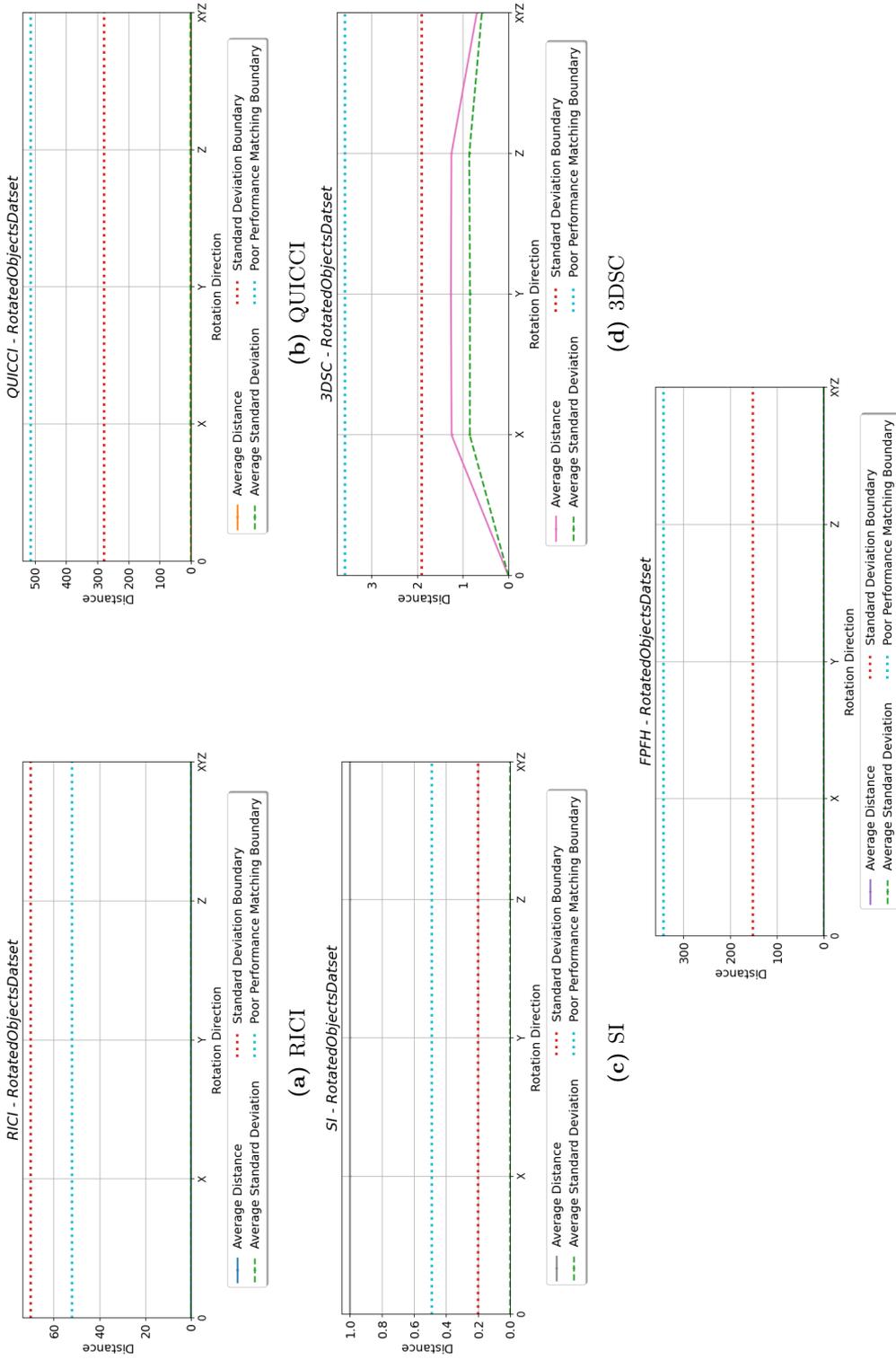
The results from the Resized Objects test can be seen in Figure 4.3.2. First off we see that all the shape descriptors behave relatively similarly to the varying degrees of size change. However with further analysis, we also see that the shape descriptors get quite different results when looking at the average distances for when the object are reduced to half their size compared to when they double in size. RICI, SI, and 3DSC's results indicate that an object half its original size is more equal in comparison to an object double its original size. On the other hand, QUICCI and FPFH's results indicate the opposite. Only RICI and SI manage to produce results that are consistently better than the poor matching floor, while the other descriptors seem to struggle with larger changes in size.

4.3.3 Moved Objects

As can be seen in Figure 4.3.3, moving an object in 3D space does not affect the performance of the shape descriptors. All of them have perfect scores.

4.3.4 Mirrored Objects

The results of the mirrored transformation test can be seen in Figure 4.3.4. The results are sorted individually on the average distance. We see here that RICI, SI, and FPFH manage to consistently produce average distances better than the set floor. While 3DSC and QUICCI seemed to struggle with this task, as most of their distances are above the floor. An interesting observation is that 3DSC manages to keep a few distances very low, but most of its results are considered poor. When observing the QUICCI results we also see that its distances are a lot more spread out than the others, which can be deduced by looking at its standard deviation values that are nearly uniformly above the standard deviation boundary.



(e) FPFH

Figure 4.3.1: Rotated Objects Results

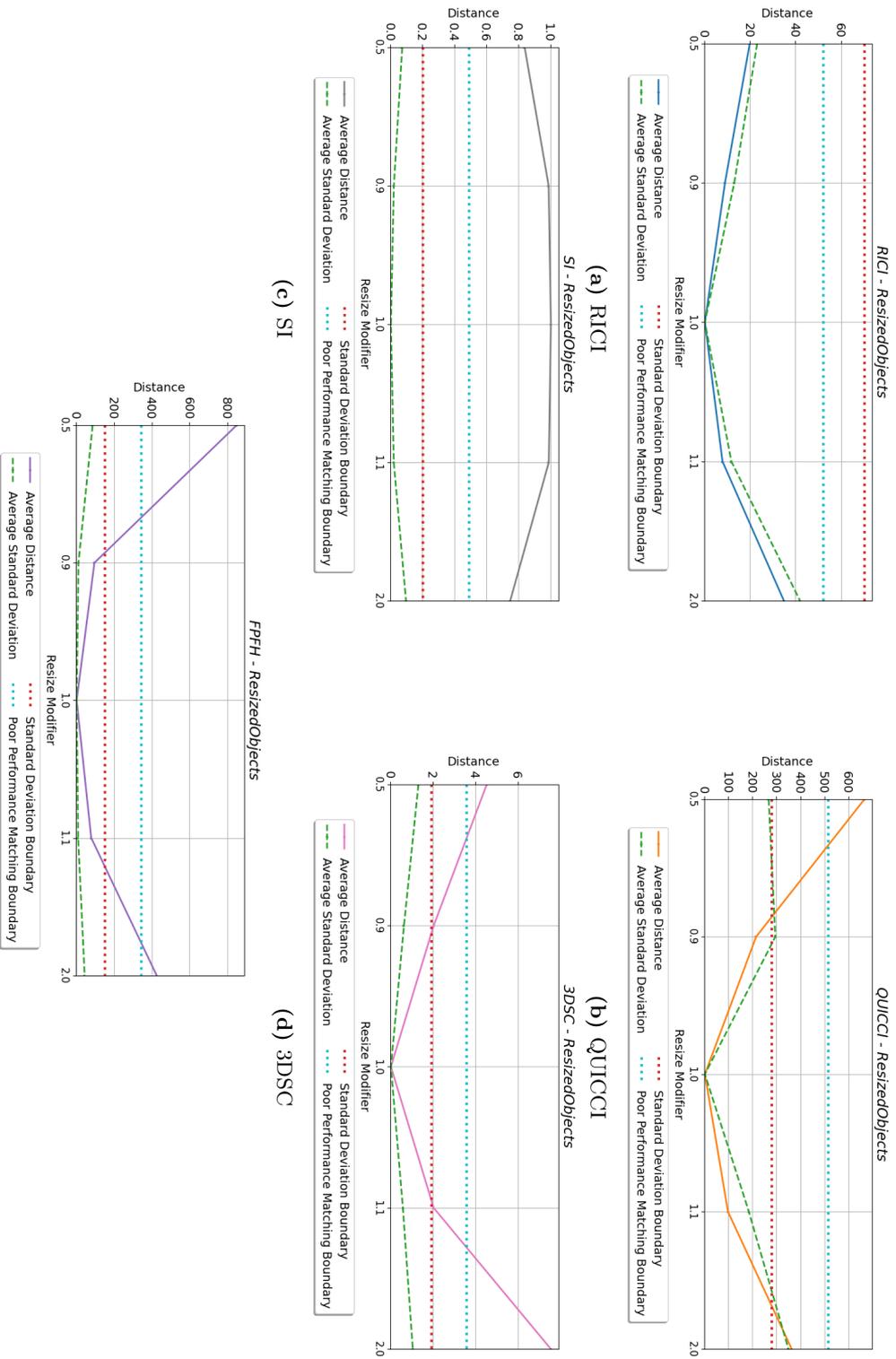
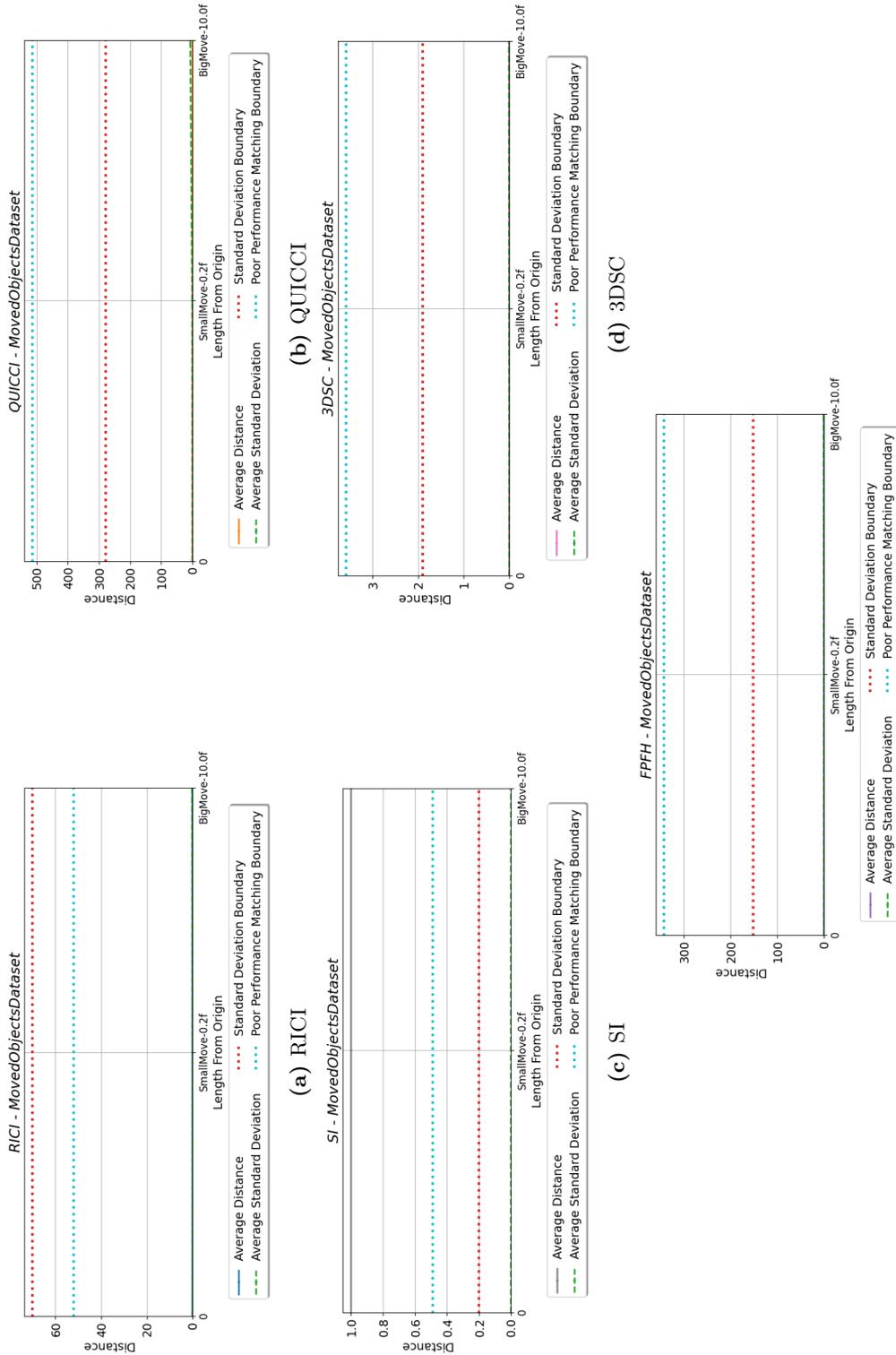


Figure 4.3.2: Resized Objects Results



(e) FPFH

Figure 4.3.3: Moved Results

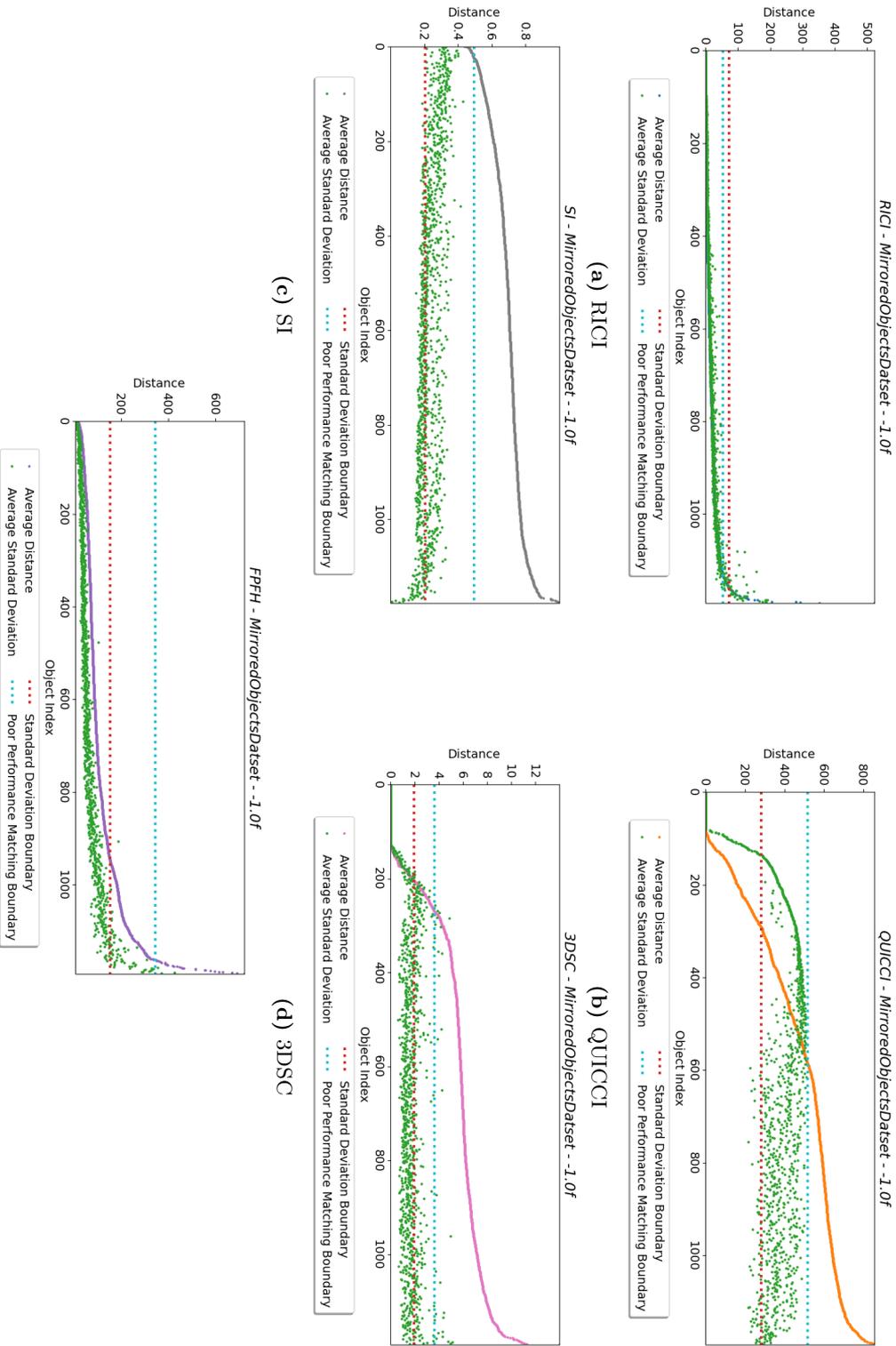


Figure 4.3.4: Mirrored Results

4.4 Deformation Resistance

The deformation resistance tests include the twisted and rippled transformations. These test the shape descriptors ability to recognise objects that have been deformed in some sort of way.

4.4.1 Twisted Deformation of Objects

The results from the Twisted Deformation of Objects test can be seen in Figure 4.4.1. The first thing to notice is that 3DSC is the only shape descriptor that passes its poor matching performance floor, but not before after the third category. FPFH and 3DSC's average distances also increase relatively fast, compared to the other shape descriptors which keep a more stable distance in the different categories. In addition, QUICCI's standard deviation gets proportionally close to its boundary.

4.4.2 Rippled Deformation of Objects

The results of the Rippled Deformation of Objects test can be seen in Figure 4.4.2. Here we see that only 3DSC goes above the set poor matching floor, but not before the most extreme variation of the rippled categories. The other shape descriptors manage to perform well for all the categories. However, we see that QUICCI's standard deviation crosses its boundary from category 0.02. This shows that the variation of distances for QUICCI are quite drastic with higher amounts of the rippled deformation, which might indicate that it is a bit more unsure than the other shape descriptors.

4.5 Noise Resistance

The Noise Resistance transformations test how the shape descriptors are affected by different types of noise. This includes Gaussian noise and changing the orientation of an object's normals.

4.5.1 Gaussian Vertex Displacement Along Normals

The results for the Gaussian Vertex Displacement Along Normals test can be seen in Figure 4.5.1. We can see that all the shape descriptors generally manage to keep a good performance throughout the different categories, however 3DSC goes outside the poor matching floor at the 0.003 category. We also see that no shape descriptor manages to keep a smooth average distance, and are all converging towards their own poor matching floor. Additionally, RICI and QUICCI's standard deviation ends up crossing their standard deviation boundary, while SI and 3DSC barely goes above. FPFH manages to keep its standard deviation steadily under the boundary in all the categories.

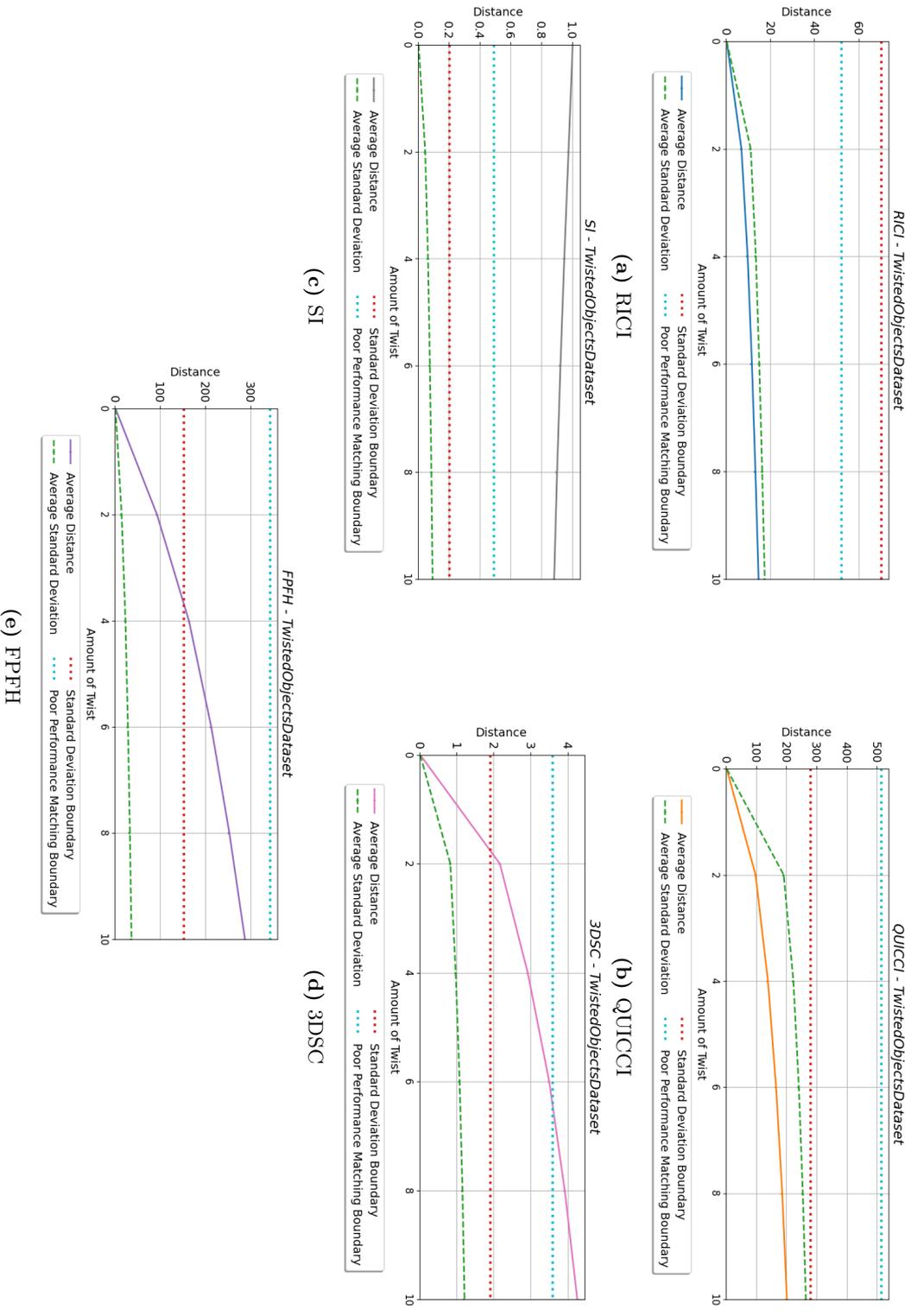
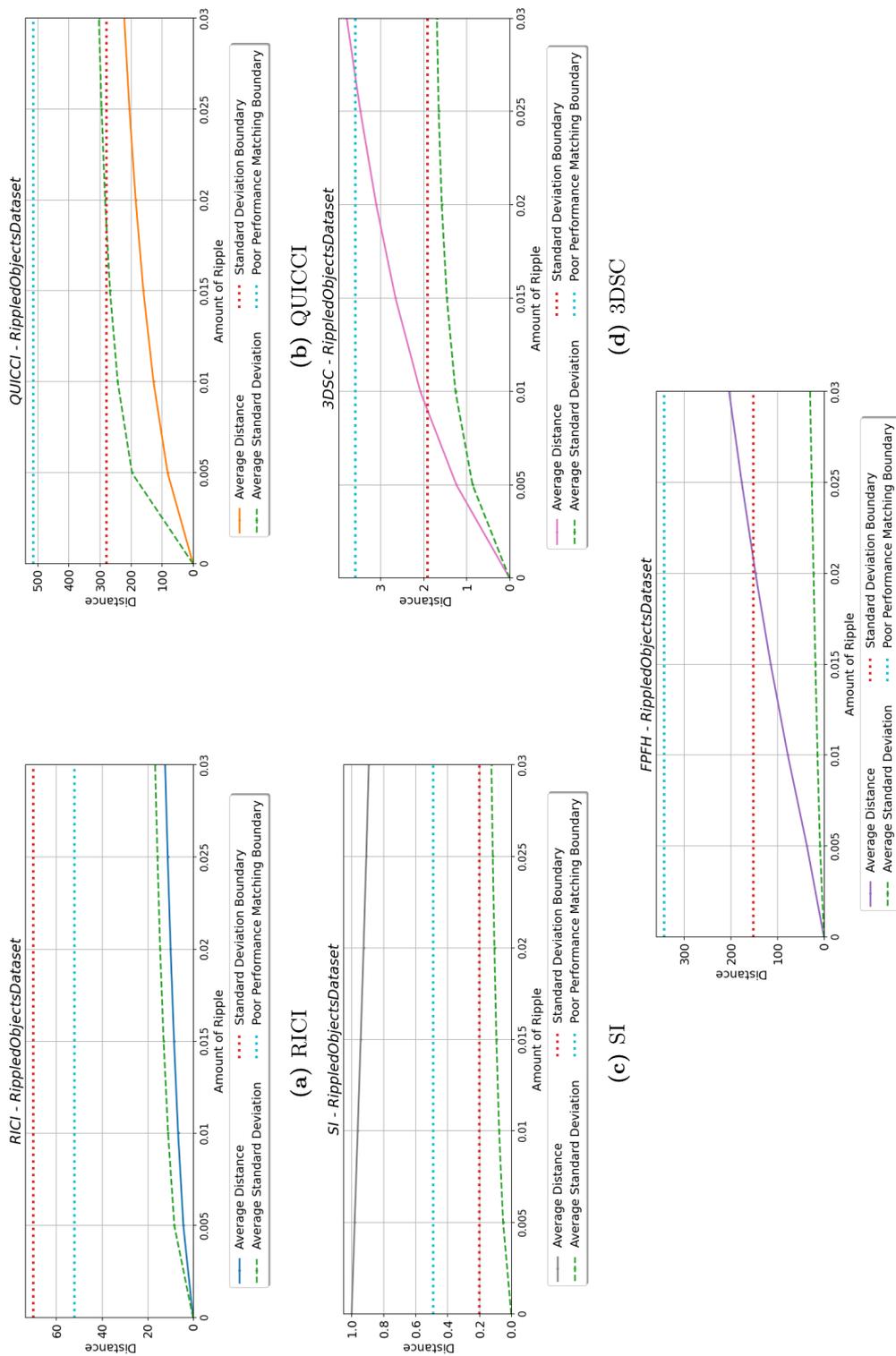


Figure 4.4.1: Twisted Deformation Results



(e) FPFH

Figure 4.4.2: Rippled Deformation Results

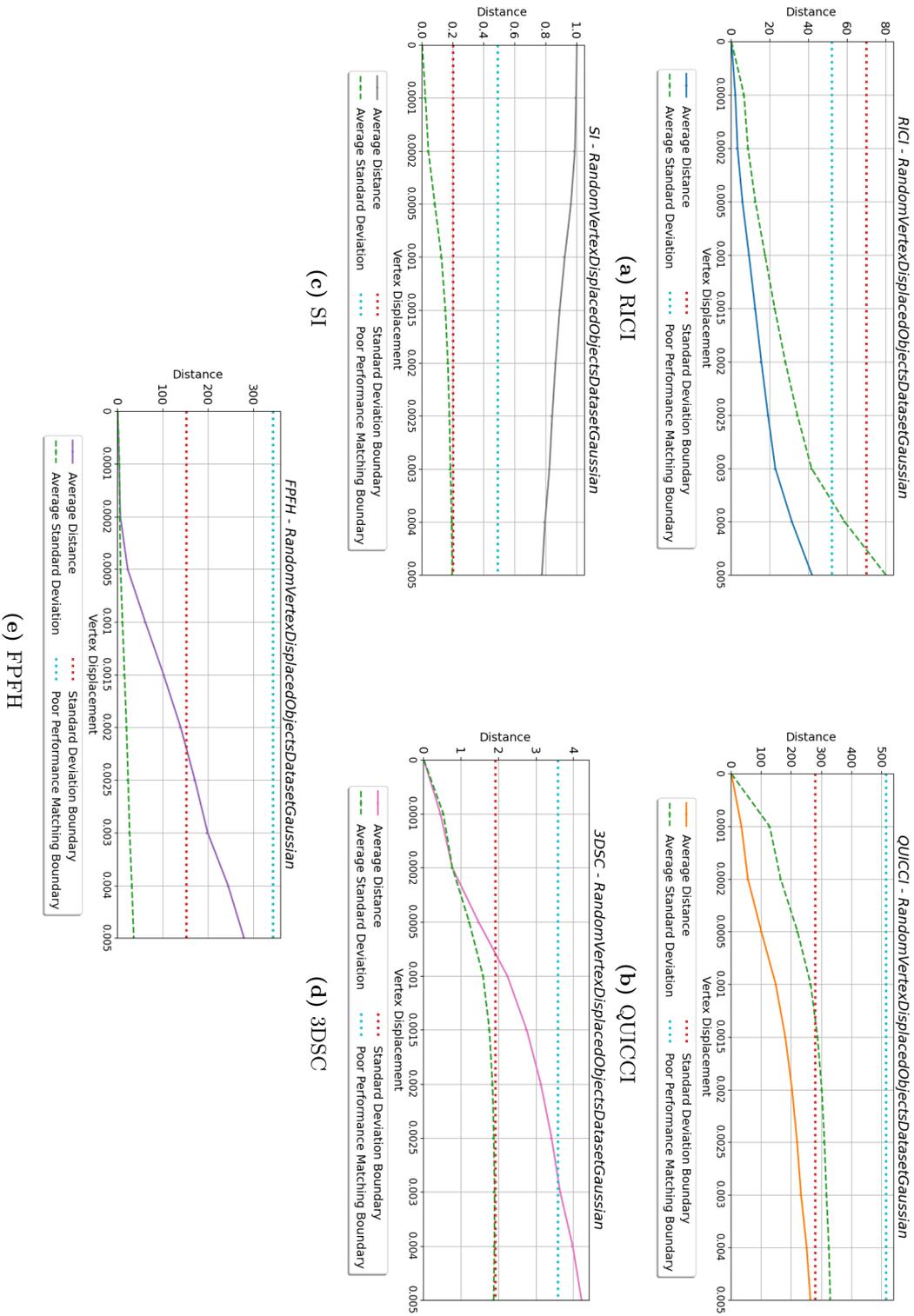


Figure 4.5.1: Gaussian Vertex Displacement Along Normals Results

4.5.2 Deviation and Rotation of Vertex Normals

The results the Deviation and Rotation of Vertex Normals tests can be seen in Figure 4.5.2. As we can observe in the diagrams, all the shape descriptors manage to keep inside of the poor matching floor when the normal deviation is under 20° . 3DSC goes outside the poor matching floor at 20° , while FPFH doesn't go above before 45° . The other shape descriptors manage to consistently keep inside of the floor.

4.6 Clutter Resistance

The Clutter Resistance transformations test the shape descriptors ability to identify an object in cluttered environments.

4.6.1 Clustered Objects

The results from the Clustered Objects test can be seen in Figure 4.6.1. We see that only QUICCI manages to keep a distance that is consistently under its poor matching floor, while on the other hand 3DSC and FPFH starts above it at the lowest amount of clustering. RICI and SI is under the floor at category 5, but both of them go outside on the next category. Another observation is that while the others' standard deviation stays steady throughout the test, RICI's increases together with the increasing average distance.

4.6.2 Ability to Match with Partial Overlap

The results from the objects with partial overlap can be seen in Figure 4.6.2. QUICCI is the only shape descriptor that seems to be resistant to this transformation. 3DSC and FPFH start above their respective poor matching floors, while SI goes under at the "45.1-55.0" category, and RICI goes over after the "35.1-45.0" category. An interesting observation is RICI's spike at the third to last category, through further investigation we found that this spike is caused only by object *977* which produced an average distance of 36,202. If we ignore this object the graph would follow a more natural growth pattern (figure 4.6.3). However, even with that removed the average still ends above the poor matching floor. As with the Clustering test, RICI's standard deviation increases together with an increasing average distance, while the others stay relatively flat. Another interesting observation is that both SI and FPFH's standard deviation decreases towards the highest amount of overlap.

4.7 Occlusions and Incomplete Surfaces

The Occlusions and Incomplete Surfaces transformations will test the shape descriptors abilities to recognise objects that have had some parts of it removed.

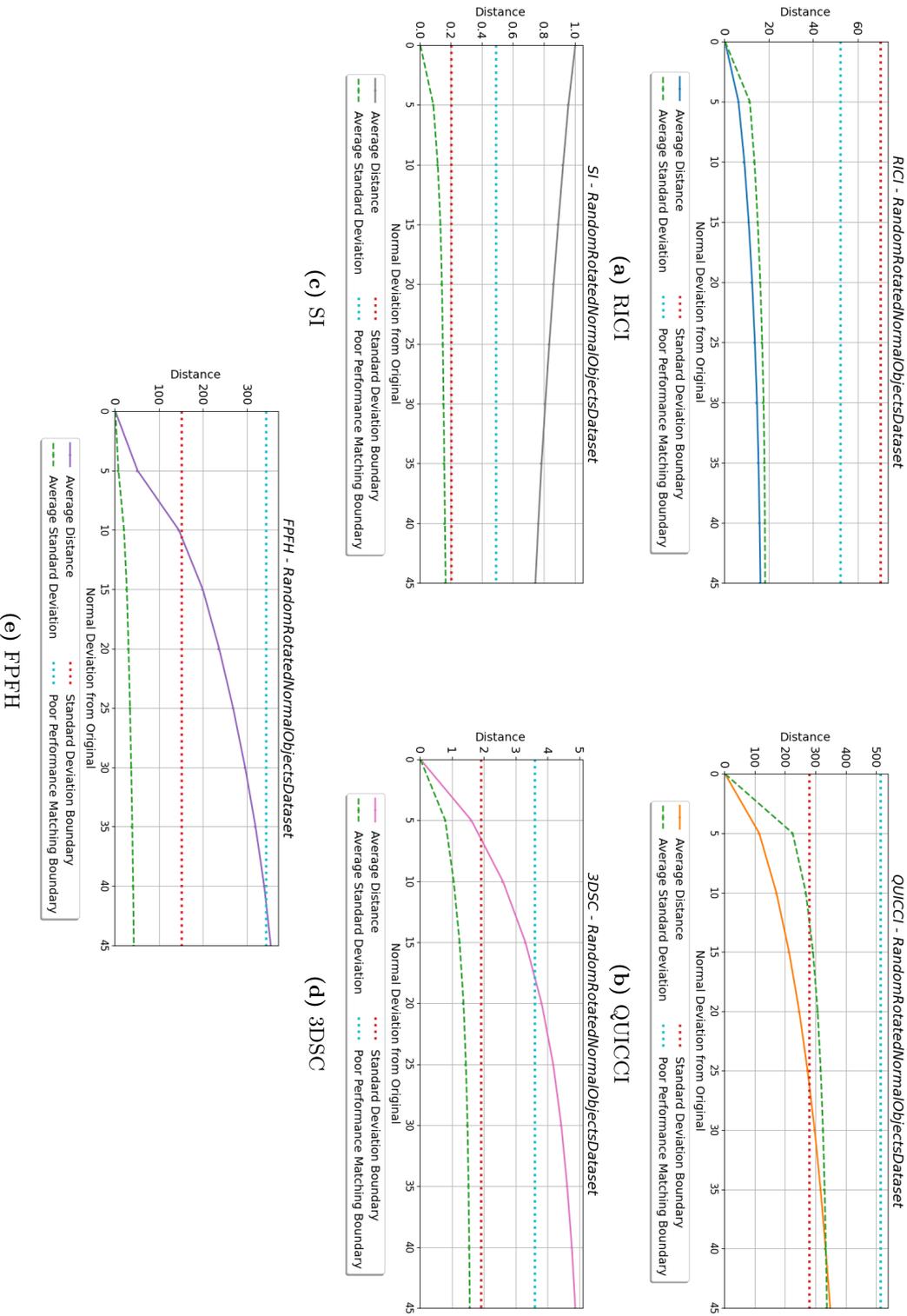


Figure 4.5.2: Deviation and Rotation of Vertex Normals Results

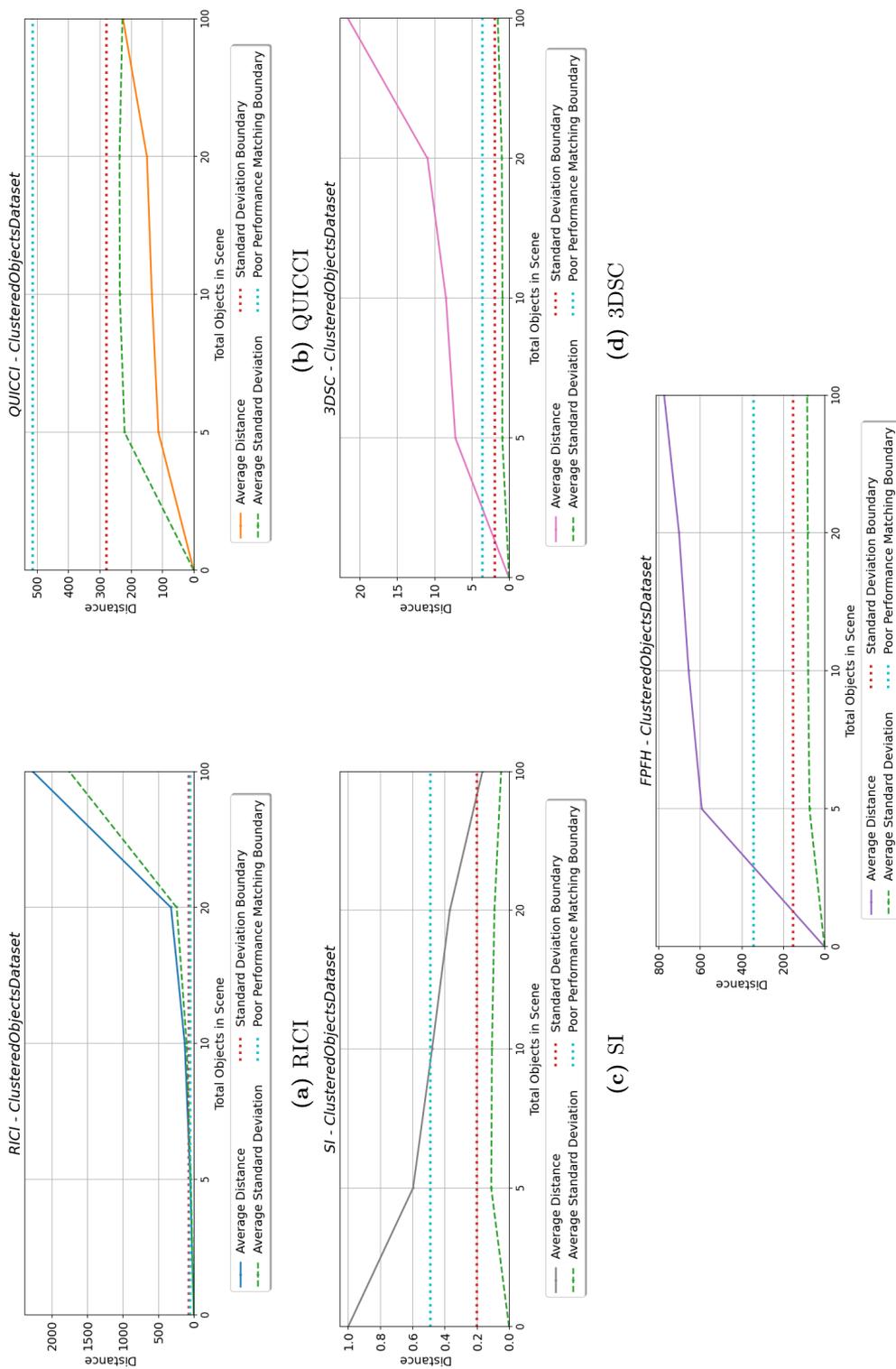
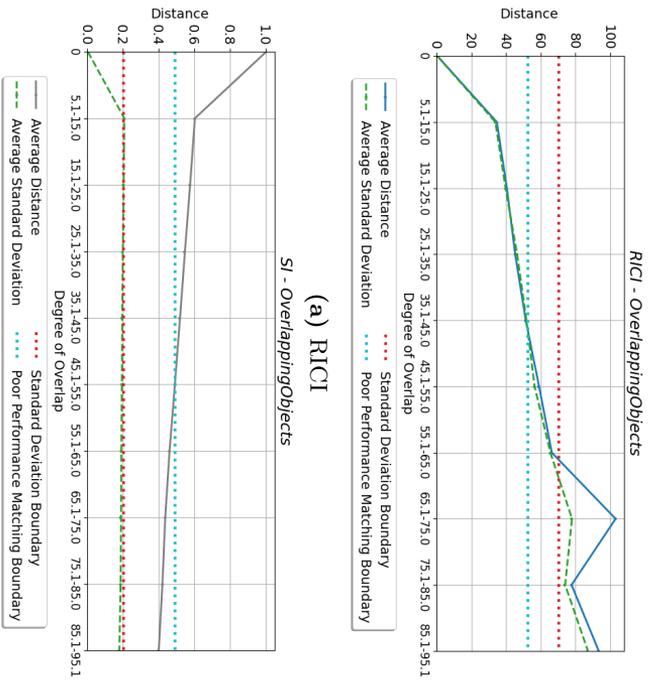
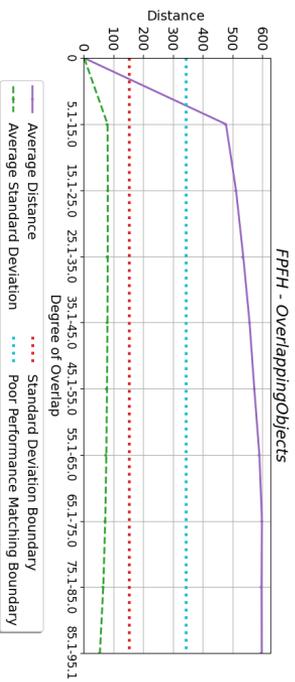


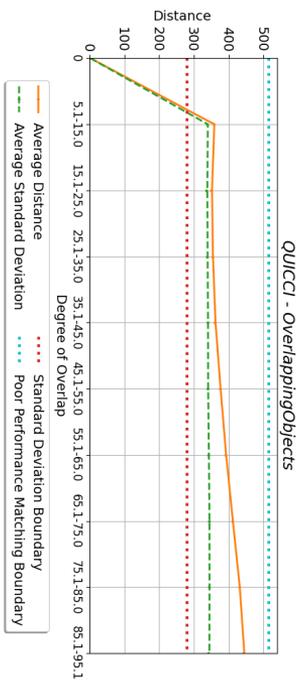
Figure 4.6.1: Clustered Objects Results



(c) SI



(e) PPFH



(d) 3DSC

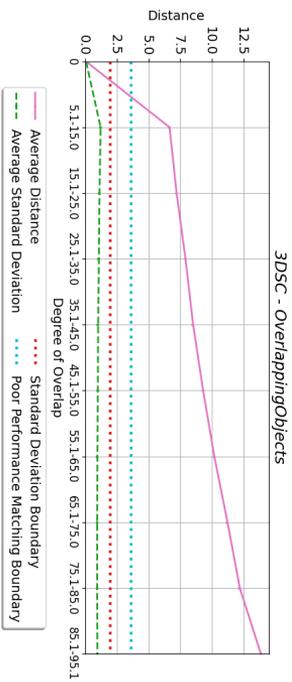


Figure 4.6.2: Ability to Match with Partial Overlap Objects Results

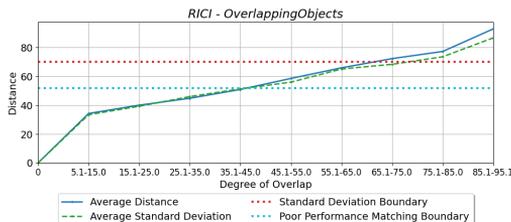


Figure 4.6.3: Ability to Match with Partial Overlap Objects results for RICI, when object 977’s result in category “65.1-75.0” is ignored

4.7.1 Objects with Holes

The results for the Objects with Holes transformation can be found in Figure 4.7.1. RICI and SI seem to be the best performers with this category, as they manage to both keep away the poor matching performance floor and their distances are not affected by the amount of holes in the object. 3DSC starts above the poor matching performance floor, while FPFH only reaches it in the “60.1-70.0” category. QUICCI also manages to keep under the poor matching performance floor, but doesn’t manage to keep a stable distance after the “40.1-50.0” category.

4.7.2 Partial Surface Visibility

The results from the Partial Surface Visibility test can be seen in Figure 4.7.2. The results are sorted individually on the test’s average distance. RICI and SI show clear signs of being the most resistant to this type of transformation, with most of their distances performing better than the set floor. SI’s standard deviations follow the standard deviation boundary. QUICCI, 3DSC, and FPFH all have distances that vary from under to above the floor, but 3DSC has the highest quantity of distances above the floor indicating a worse performance.

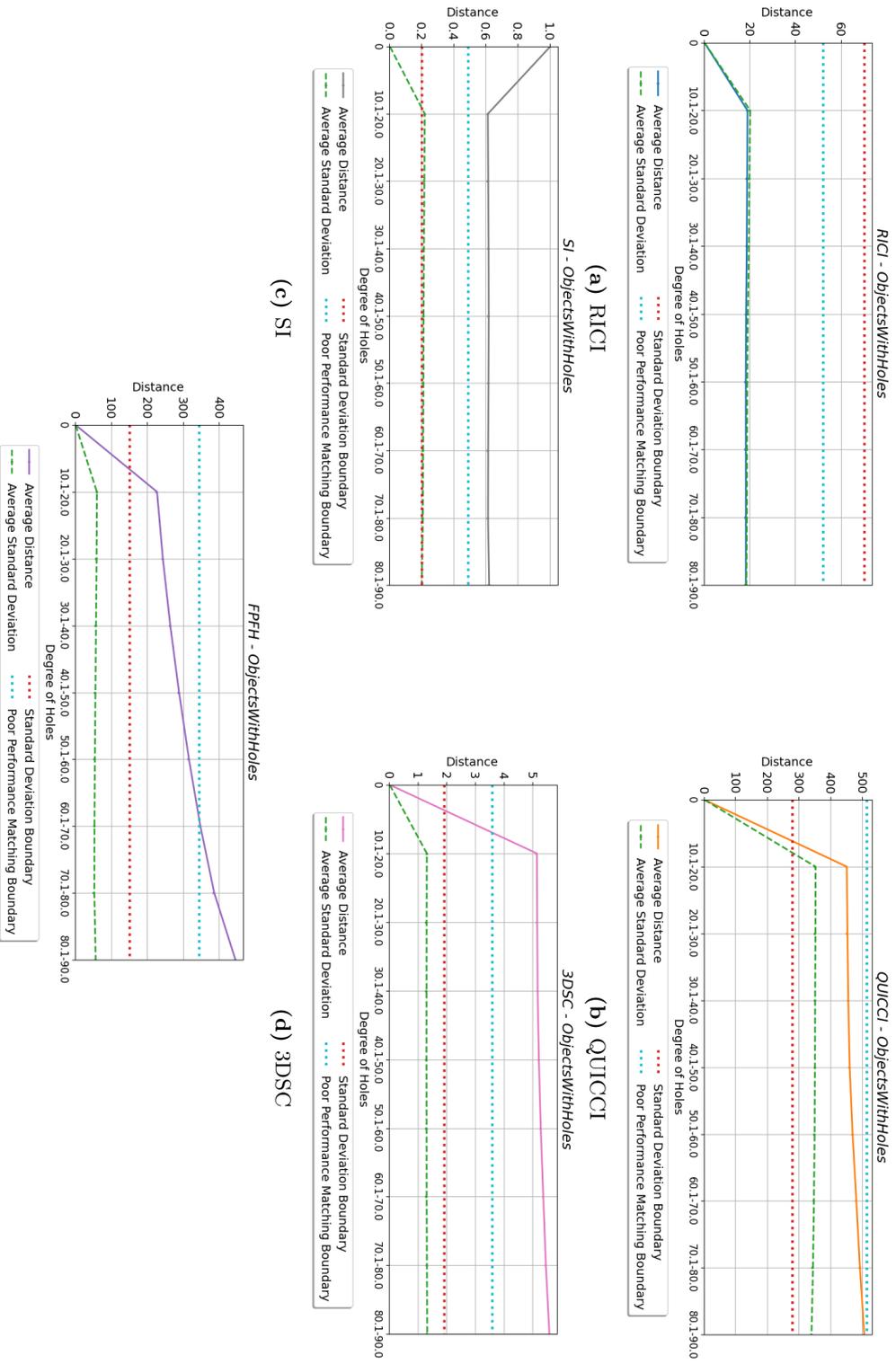
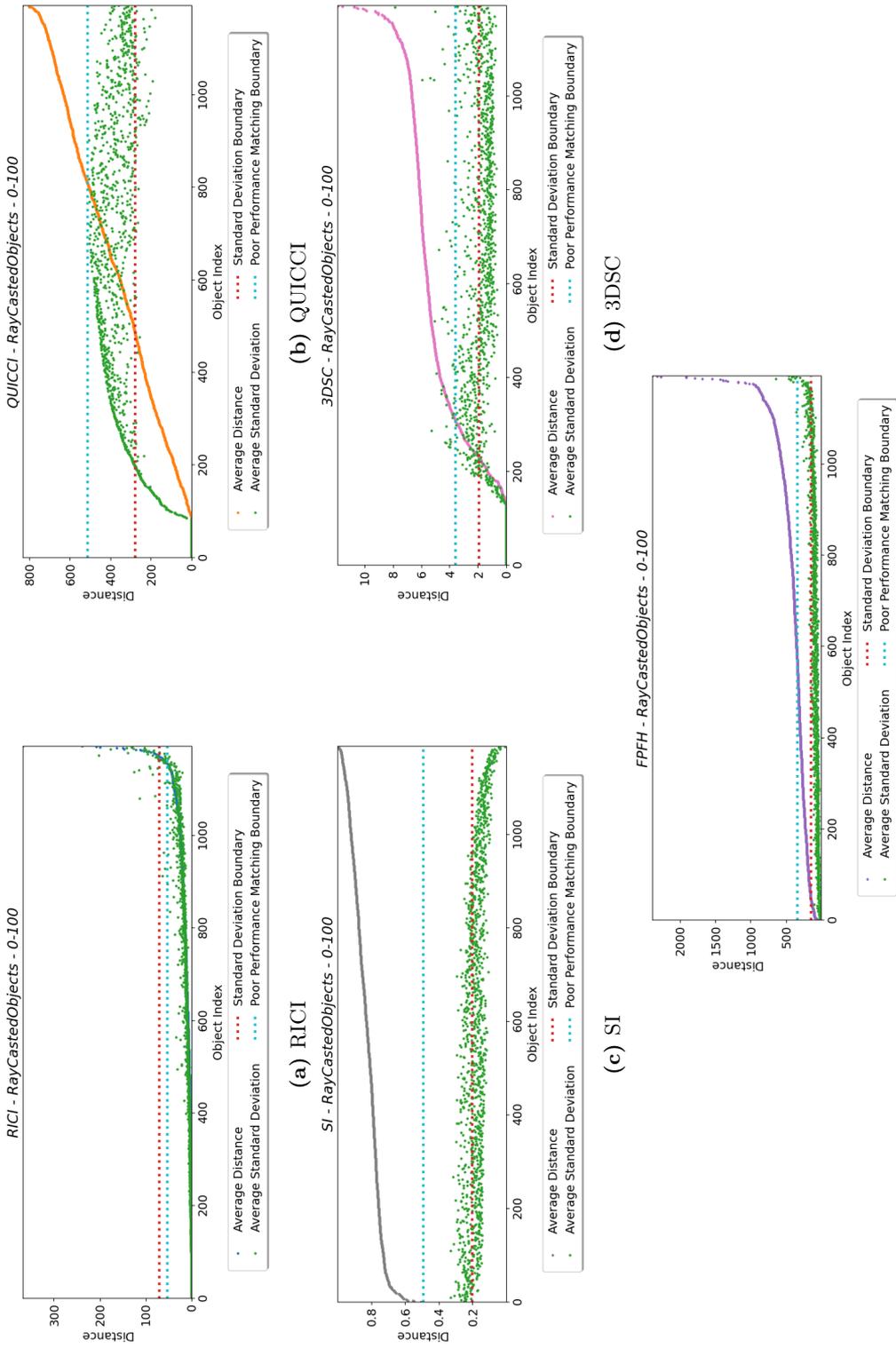


Figure 4.7.1: Objects With Holes Results



(e) FPFH

Figure 4.7.2: Partial Surface Visibility Results

DISCUSSION

This chapter will discuss what we found from our results, as well as discuss more technical parts of the project.

5.1 Evaluating the Shape Descriptors

As previously discussed, we can't compare the shape descriptors directly to each other using the method we proposed. Therefore we need to see how well they perform overall within each set of transformations and determine which of them perform good and which of them are lacking. This will be done using the results of each test, and examining how their average distances and standard deviations compares to the previously set floors.

5.1.1 Basic Matching Performance

The Basic Matching Performance transformations included the following tests: Rotated (Figure 4.3.1), Resized (Figure 4.3.2), Moved (Figure 4.3.3), and Mirrored Objects (Figure 4.3.4). We see that throughout these tests all the shape descriptors generally perform well, however 3DSC tends to stand out. In all the tests, except Moved Objects where they all got a perfect score, 3DSC is never in the top performers. This is clear in both the Mirrored results and Rotated Objects. In the mirrored results 3DSC for the most part ends up outside its poor matching floor. While the others, with exception of QUICCI, produces good results. Even though 3DSC's results in the Rotated Objects is considered good, all the others are not affected by the rotation. Since we are looking at how shape descriptors' object recognition abilities are affected by transformations, this indicates a poorer performance than the other shape descriptors. This fault might be caused by 3DSC's support volume having a free axis of rotation, which can have led to small areas of the object ending up in different bins than in the original. The two shape descriptors that stand out in a positive way in these tests are the RIC1 and SI shape descriptors. We see that for

all the tests they for the most part manage to stay within their respective poor matching and standard deviation floors. FPFH and QUICCI manages to produce equal results to RIC1 and SI in both the Rotated and Moved Objects test, but they both seem to struggle in the Resized Objects test. They have at least one category where they end up outside their poor matching floor. However as pointed out, their results indicate that an object half of its original size is less equal than an object double its original size. This is opposite to the other shape descriptors, which suggests that they are more sensitive to one direction of resizing than the shape descriptors.

5.1.2 Deformation Resistance

The Deformation Resistance transformations included the following tests: Twisted Deformation (Figure 4.4.1) and Rippled Deformation of Objects (Figure 4.4.2). The shape descriptors act similarly in the two tests. This is not unexpected as they both deform the objects, just using different methods. We see here that only 3DSC goes outside its poor matching floor, but not before some of the more extreme deformations (Category 8 in Twisted and 0.03 in Rippled). This shows that 3DSC performs worse than the others for this transformation. QUICCI's standard deviation moves above its boundary at category 0.02 in the Rippled Deformation test, which indicates that its results are a lot more varied than the others. This can be sub-optimal in situations where you depend on the results being more stable. Other than this RIC1, QUICCI, SI, and FPFH produce good results in both environments, even at the most extreme transformations.

5.1.3 Noise Resistance

The Noise Resistance transformations included the following tests: Gaussian Vertex Displacement Along Normals (Figure 4.5.1) and Deviation and Rotation of Vertex Normals (Figure 4.5.2). Both of these tests show that the shape descriptors are generally resistant to moderate amounts of these types of noise. However, on the more extreme levels we see that 3DSC in both tests goes above its poor matching floor and FPFH goes above its floor when the normal deviation reaches 45° . We also see that RIC1, QUICCI, and SI produce good average distances throughout the different tests. But RIC1 and QUICCI's standard deviations tend to go over their boundary. RIC1's standard deviation only goes above its boundary at the most extreme variation of the Gaussian Noise test, while QUICCI's standard deviation ends up above around half-way in both. These results might indicate that SI overall is the most resistant to these noise tests, however RIC1, QUICCI, and FPFH still have an acceptable performance when the conditions are not at their most extreme.

5.1.4 Clutter Resistance

The Clutter Resistance transformations included the following tests: Clustered Objects (Figure 4.6.1) and Ability to Match with Partial Overlap (Figure 4.6.2). Even though the motivation for creating many of these shape descriptors was to solve the problem of clutter, we see that it has a large effect on them. Only QUICCI produces results that do not go outside the poor matching performance floor. We chose to treat RICI's spike as an outlier (Figure 4.6.2a), as it's most likely something that went wrong when calculating as it is not present in the other tests. It should therefore not contribute towards RICI's overall performance. Both RICI and SI manage to keep inside of the poor matching floor when there is low amounts of objects in the scene and degree of overlap, but they start to struggle when these are increased. RICI's rapid increase in distance when the amount of objects in the scene increases might be explained by how it uses the squared sum of intersections around a given point. Which leads to it calculating higher values in environments with high amount of clutter. RICI's standard deviation also stands out with it increasing at the same pace as the the average distance, which indicates it having a larger variation in results when the clutter gets larger. This isn't present with the other shape descriptors.

5.1.5 Occlusions and Incomplete Surfaces

The Occlusions and Incomplete Surfaces transformations included the following tests: Objects with Holes (Figure 4.7.1) and Partial Surface Visibility (Figure 4.7.2). Here we see that in both tests 3DSC and FPFH ends up outside the poor matching floor, while the others keep a relatively stable distance in both tests. It is difficult to determine how well QUICCI performs in the Partial Surface Visibility test, as its distance are so varied. In Objects with Holes it looks like RICI and SI are unaffected by the amount of holes in the object and keeps a steady distance throughout. Additionally, they both prove to have good results in the Partial Surface Visibility test as well. Proving that they perform better than the other shape descriptors in these types of environments.

5.1.6 Overall

Looking at the results some shape descriptors seem to be more resistant to some of the scenarios than others. Both RICI and SI seem to provide consistently good performance in the different tests, while 3DSC and FPFH seems to provide a more varying degree of reliability. QUICCI has also been a good contender, but its outputs appear to be a lot more varying than the others. However, with its quick comparison and generation time it is definitely a good candidate for many tasks.

5.2 Project Architecture

There were many different ways we could have split the architecture of the project. It could have been one large project with the same programming language and libraries, which would have made communication between the different parts a bit easier to implement. Or we could have merged the dataset transformations into the benchmarking tool, which would have enabled us to generate new simulations on-demand if we for example wanted to merge together two transformations. However, by choosing to split the project into the Data Collection, Performance Data Generation, and Data Analysis parts we were able to freely choose the technologies that fit for each task, without adhering to specific technology requirements for the entire project. For example we could choose Unity and utilise its features to dynamically create transformations, without having to create the rest of the project within a game engine. This made it easier to develop and optimise each component independently, without affecting the others. Additionally, separating the components also allowed for easier testing and debugging, as issues could be isolated to a specific component rather than having to search through the entire project. There is however no correct answer to what the best architecture would look like, and it might be necessary to change ours in the future if new requirements are presented.

CONCLUSIONS

We originally created three objectives which would be used in order to answer our research question: *To what extent is the object recognition abilities of shape descriptors affected when simulating real-world scenarios?*. These objectives were:

1. Identify the criteria by which a shape descriptor should be evaluated.
2. Develop a method to measure these criteria in an objective manner.
3. Evaluate the performance of popular shape descriptors based on the identified criteria.

These objectives have defined the structure of how we have attempted to answer the question.

Through reading previous papers and related works, we found that there was a lack of research on what factors make the performance of a shape descriptor good or bad, and a lack of a standard method for evaluating their performance. This motivated the creation of a new benchmark, which tested shape descriptor's performance in a diverse set of real world scenarios. The benchmark also utilised our new method to fairly test their performance. By comparing two completely unrelated objects we calculate a poor matching floor and standard deviation boundary for each shape descriptor and distance function combination, and use these to identify if a shape descriptor's performance in a given environment is good or not.

The benchmark consisted of three separated processes: Data Collection, Performance Data Generation, and Data Analysis. All of the different parts were developed with replicability and modifiability in mind, which lets researchers generate the same results as us and build upon our code to add their own shape descriptors and transformations. This will enable researchers to use this benchmark as a standard in the future, which will legitimise improvements for a given shape descriptor.

We have evaluated the performance of five shape descriptors: Spin Image (SI), 3D Shape Context (3DSC), Fast Point Feature Histogram

(FPFH), Radial Intersection Count Image (RICI), and Quick Intersection Count Change Image (QUICCI). These shape descriptors were tested in five different environments: Basic Matching Performance, Deformation Resistance, Noise Resistance, Clutter Resistance, and Occlusions and Incomplete Surfaces. Which represents a wide range of different real-world scenarios which shape descriptors might face. We found that both SI and RICI provide overall consistent good performance, while the others lack reliability for some of the different transformations. Through these results our original research questions has been answered. Additionally, they will allow future researchers and developers to take educated choices for which shape descriptors to choose for their research or applications.

With our results and research contributions we have expanded the knowledge of how shape descriptors perform across a broader range of scenarios. Our testing environments have been more closely modelled after real-world scenarios than those used in previous studies. Furthermore, we have developed transformations to isolate the variables that may impact the results of the tests more effectively. For instance, we've subdivided cluster resistance into separate categories: partial overlapping of objects and clustering of objects. These contributions have enabled more precise and relevant analysis in our research. This is a field in research where there is still a lot to figure out. We hope that our thesis and benchmark project will aid future researchers in developing new shape descriptors, which hopefully further improve the field of computer vision.

6.1 Future Work

As discussed in different parts of the thesis, there are some areas which we think can contribute to improved results. These are features we didn't have time to implement ourselves or they were out of our scope, because of their complexity. First off, looking into methods on how to directly compare the different shape descriptor and distance function outputs. An approach to this could be to figure out how to normalise their values. This feature would greatly benefit the benchmark, as one wouldn't need to speculate on which shape descriptors had the best performance.

Following this a method that changes the support radius parameter of a shape descriptor based on the level of detail in the current scene. An important factor here is that the support radius has to always be the same for the same point in an object, as if otherwise an equal object might not get recognised by the shape descriptor. This could increase the object recognition abilities of the different shape descriptors, as both smaller and larger features of an object would be guaranteed to be taken into account.

Another area that would greatly benefit the benchmark, is to further research how much the calculated poor matching floor is affected by the size of the objects. As our objects were mostly the same size, hand-picking some objects to calculate the floor was a viable option. However, if you are working with a more diverse set of objects some more research could

be needed.

In the end, an even more diverse and larger dataset in addition to implementing more shape descriptors into the benchmark would benefit the results substantially. As the benchmark has been made with modifiability in mind, adding these features shouldn't be a complex task. This would give researchers more data to analyse, which would further help them in their research.

REFERENCES

- [1] H. Gunnarsli and J. Brooks. *Autumn 2022 Project Summary*. Dec. 2022.
- [2] John Novatnack and Ko Nishino. “Scaledependent/invariant local 3D shape descriptors for fully automatic registration of multiple sets of range images”. In: *European conference on computer vision*. Springer, 2008, pp. 440–453.
- [3] Sotiris Malassiotis and Michael G Strintzis. “Snapshots: A novel local surface descriptor and matching algorithm for robust 3D surface alignment”. In: *IEEE Transactions on pattern analysis and machine intelligence* 29.7 (2007), pp. 1285–1290.
- [4] Sameh M Yamany and Aly A Farag. “Surface signatures: an orientation independent freeform surface representation scheme for the purpose of objects registration and matching”. In: *IEEE transactions on pattern analysis and machine intelligence* 24.8 (2002), pp. 1105–1120.
- [5] Maks Ovsjanikov and et al. “Exploration of continuous variability in collections of 3D shapes”. In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), p. 33.
- [6] Ruizhen Hu, Lubin Fan and Ligang Liu. “Cosegmentation of 3d shapes via subspace clustering”. In: *Computer graphics forum* 31.5 (2012), pp. 1703–1713.
- [7] Zizhao Wu and et al. “Unsupervised cosegmentation of 3D shapes via affinity aggregation spectral clustering”. In: *Computers & Graphics* 37.6 (2013), pp. 628–637.
- [8] Cong Feng, Andrei C. Jalba and Alexandru C. Telea. “A Descriptor for Voxel Shapes Based on the Skeleton Cut Space”. In: *Eurographics Workshop on 3D Object Retrieval*. Ed. by A. Ferreira, A. Giachetti and D. Giorgi. The Eurographics Association, 2016. ISBN: 978-3-03868-004-8. DOI: [10.2312/3dor.20161082](https://doi.org/10.2312/3dor.20161082).

- [9] Daniela Craciun, Guillaume Levieux and Matthieu Montes. “Shape Similarity System driven by Digital Elevation Models for Nonrigid Shape Retrieval”. In: *Eurographics Workshop on 3D Object Retrieval*. Ed. by Ioannis Pratikakis, Florent Dupont and Maks Ovsjanikov. The Eurographics Association, 2017. ISBN: 978-3-03868-030-7. DOI: [10.2312/3dor.20171051](https://doi.org/10.2312/3dor.20171051).
- [10] A.E. Johnson and M. Hebert. “Using spin images for efficient object recognition in cluttered 3D scenes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.5 (1999), pp. 433–449. DOI: [10.1109/34.765655](https://doi.org/10.1109/34.765655).
- [11] Andrea Frome et al. “Recognizing Objects in Range Data Using Regional Point Descriptors”. In: *Computer Vision - ECCV 2004*. Ed. by Tomáš Pajdla and Jiří Matas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 224–237. ISBN: 978-3-540-24672-5.
- [12] Radu Bogdan Rusu, Nico Blodow and Michael Beetz. “Fast Point Feature Histograms (FPFH) for 3D registration”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 3212–3217. DOI: [10.1109/ROBOT.2009.5152473](https://doi.org/10.1109/ROBOT.2009.5152473).
- [13] Bart Iver van Blokland and Theoharis Theoharis. “Radial intersection count image: A clutter resistant 3D shape descriptor”. In: *Computers & Graphics* 91 (2020), pp. 118–128.
- [14] Bart Iver van Blokland and Theoharis Theoharis. “An indexing scheme and descriptor for 3D object retrieval based on local shape querying”. In: *Computers & Graphics* 92 (2020), pp. 55–66. ISSN: 0097-8493. DOI: <https://doi.org/10.1016/j.cag.2020.09.001>. URL: <https://www.sciencedirect.com/science/article/pii/S009784932030128X>.
- [15] Adam Finkelstein. COS 426 Lecture. 2005. URL: <https://www.cs.princeton.edu/courses/archive/spr05/cos426/lectures/12-reps.pdf>.
- [16] Tiange Xiang et al. “Walk in the cloud: Learning curves for point clouds shape analysis”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 915–924.
- [17] Steve Marschner. CS 4620 Lecture. 2014. URL: <https://www.cs.cornell.edu/courses/cs4620/2014fa/lectures/02trimesh1.pdf>.
- [18] Safe Software. *Vertex Normals*. URL: https://docs.safe.com/fme/html/FME_Desktop_Documentation/FME_ReadersWriters/!FME_Geometry/Vertex_Normals.htm.
- [19] Unity Technologies. *Anatomy of a Mesh*. 2022. URL: <https://docs.unity.cn/560/Documentation/Manual/AnatomyofaMesh.html>.

- [20] William R. Sherman and Alan B. Craig. “Chapter 6 - Presenting the Virtual World”. In: *Understanding Virtual Reality (Second Edition)*. Ed. by William R. Sherman and Alan B. Craig. Second Edition. The Morgan Kaufmann Series in Computer Graphics. Boston: Morgan Kaufmann, 2018, pp. 398–536. ISBN: 978-0-12-800965-9. DOI: <https://doi.org/10.1016/B978-0-12-800965-9.00006-4>. URL: <https://www.sciencedirect.com/science/article/pii/B9780128009659000064>.
- [21] Juana Valeria Hurtado and Abhinav Valada. “Chapter 12 - Semantic scene segmentation for robotics”. In: *Deep Learning for Robot Perception and Cognition*. Ed. by Alexandros Iosifidis and Anastasios Tefas. Academic Press, 2022, pp. 279–311. ISBN: 978-0-323-85787-1. DOI: <https://doi.org/10.1016/B978-0-32-385787-1.00017-8>. URL: <https://www.sciencedirect.com/science/article/pii/B9780323857871000178>.
- [22] Jing Wang, Juan Zhang and Qingtong Xu. “Research on 3D laser scanning technology based on point cloud data acquisition”. In: *2014 International Conference on Audio, Language and Image Processing*. 2014, pp. 631–634. DOI: [10.1109/ICALIP.2014.7009871](https://doi.org/10.1109/ICALIP.2014.7009871).
- [23] Matthew Berger et al. “A Survey of Surface Reconstruction from Point Clouds”. In: *Computer Graphics Forum* 36.1 (2017), pp. 301–329.
- [24] Dirk Hahnel, Wolfram Burgard and Sebastian Thrun. “Learning compact 3D models of indoor and outdoor environments with a mobile robot”. In: *Robotics and Autonomous Systems* 44.1 (2003), pp. 15–27.
- [25] Jacopo Aleotti and Stefano Caselli. “A 3D Shape Segmentation Approach for Robot Grasping by Parts”. In: *Robotics and Autonomous Systems* 60.3 (2012), pp. 358–366.
- [26] Michael Kazhdan and Hugues Hoppe. “Screened Poisson Surface Reconstruction”. In: *ACM Transactions on Graphics (TOG)* 32.3 (2013), p. 29.
- [27] Fausto Bernardini et al. “The ball-pivoting algorithm for surface reconstruction”. In: *IEEE transactions on visualization and computer graphics* 5.4 (1999), pp. 349–359.
- [28] Herbert Edelsbrunner, David G Kirkpatrick and Raimund Seidel. “A shape theory for general point sets in the plane”. In: *Journal of the ACM (JACM)* 30.3 (1983), pp. 668–677.
- [29] The Spatial Team. “The Main Benefits and Disadvantages of Point-Cloud Modeling”. In: (17th Dec. 2019). URL: <https://blog.spatial.com/the-main-benefits-and-disadvantages-of-point-cloud-modeling> (visited on 21/05/2023).
- [30] Andrew E Johnson. “Spin-images: a representation for 3-D surface matching”. In: (1997).

- [31] S. Belongie, J. Malik and J. Puzicha. “Shape matching and object recognition using shape contexts”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.4 (2002), pp. 509–522. DOI: [10.1109/34.993558](https://doi.org/10.1109/34.993558).
- [32] Radu Bogdan Rusu et al. “Aligning point cloud views using persistent feature histograms”. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, pp. 3384–3391. DOI: [10.1109/IROS.2008.4650967](https://doi.org/10.1109/IROS.2008.4650967).
- [33] *Fast Point Feature Histograms (FPFH) descriptors*. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/fpfh_estimation.html (visited on 10/04/2023).
- [34] Yulan Guo et al. “A Comprehensive Performance Evaluation of 3D Local Feature Descriptors”. In: *International Journal of Computer Vision volume 116* (2016), pp. 66–89. ISSN: 1573-1405. DOI: <https://doi.org/10.1007/s11263-015-0824-y>. URL: <https://link.springer.com/article/10.1007/s11263-015-0824-y>.
- [35] Manolis Savva et al. “SHREC’17 Track Large-Scale 3D Shape Retrieval from ShapeNet Core55”. In: *Proceedings of the Eurographics Workshop on 3D Object Retrieval (2017)*. Vol. 10. 2017.
- [36] Aitor Aldoma et al. “Tutorial: Point Cloud Library: Three-Dimensional Object Recognition and 6 DOF Pose Estimation”. In: *IEEE Robotics & Automation Magazine* 19.3 (2012), pp. 80–91. DOI: [10.1109/MRA.2012.2206675](https://doi.org/10.1109/MRA.2012.2206675).
- [37] Cuemath. *Euclidean Distance Formula*. URL: <https://www.cuemath.com/euclidean-distance-formula/> (visited on 11/04/2023).
- [38] Nitya Raut. *What is Hamming Distance?* 2020. URL: <https://www.tutorialspoint.com/what-is-hamming-distance> (visited on 11/04/2023).
- [39] Shaun Turney. *Pearson Correlation Coefficient (r) | Guide & Examples*. 2022. URL: <https://www.scribbr.com/statistics/pearson-correlation-coefficient/> (visited on 12/04/2023).
- [40] Zhizhong Han et al. “Mesh Convolutional Restricted Boltzmann Machines for Unsupervised Learning of Features With Structure Preservation on 3-D Meshes”. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (2017), pp. 2268–2281. DOI: [10.1109/TNNLS.2016.2582532](https://doi.org/10.1109/TNNLS.2016.2582532).
- [41] Anastasia Ioannidou et al. “Deep Learning Advances in Computer Vision with 3D Data: A Survey”. In: *ACM Comput. Surv.* 50.2 (Apr. 2017). ISSN: 0360-0300. DOI: [10.1145/3042064](https://doi.org/10.1145/3042064). URL: <https://doi.org/10.1145/3042064>.
- [42] Mikaela Angelina Uy et al. “Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1588–1597.

- [43] Xianfang Sun et al. “Noise in 3D laser range scanner data”. In: *2008 IEEE International Conference on Shape Modeling and Applications*. 2008, pp. 37–45. DOI: [10.1109/SMI.2008.4547945](https://doi.org/10.1109/SMI.2008.4547945).
- [44] Pedram Oskouie, Burcin Becerik-Gerber and Lucio Soibelman. “Automated measurement of highway retaining wall displacements using terrestrial laser scanners”. In: *Automation in Construction* 65 (2016), pp. 86–101. ISSN: 0926-5805. DOI: <https://doi.org/10.1016/j.autcon.2015.12.023>. URL: <https://www.sciencedirect.com/science/article/pii/S092658051500271X>.
- [45] Xianfang Sun et al. “Noise analysis and synthesis for 3D laser depth scanners”. In: *Graphical Models* 71.2 (2009). IEEE International Conference on Shape Modeling and Applications 2008, pp. 34–48. ISSN: 1524-0703. DOI: <https://doi.org/10.1016/j.gmod.2008.12.002>. URL: <https://www.sciencedirect.com/science/article/pii/S1524070308000337>.
- [46] Marie-Julie Rakotosaona et al. “POINTCLEANNET: Learning to denoise and remove outliers from dense point clouds”. In: *Computer Graphics Forum*. Vol. 39. 1. Wiley Online Library. 2020, pp. 185–203.
- [47] Sebastian Koch et al. “ABC: A Big CAD Model Dataset For Geometric Deep Learning”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.
- [48] Angel X. Chang et al. *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.
- [49] Laura Downs et al. *Google Scanned Objects: A High-Quality Dataset of 3D Scanned Household Items*. 2022. arXiv: [2204.11918](https://arxiv.org/abs/2204.11918) [cs.RO].
- [50] George A. Miller. “WordNet: A Lexical Database for English”. In: *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*. 1992. URL: <https://aclanthology.org/H92-1116>.
- [51] Yu Xiang, Roozbeh Mottaghi and Silvio Savarese. “Beyond PASCAL: A benchmark for 3D object detection in the wild”. In: (2014).
- [52] Klaus Greff et al. *Kubric: A scalable dataset generator*. 2022. arXiv: [2203.03570](https://arxiv.org/abs/2203.03570) [cs.CV].
- [53] Huang Huang et al. *Mechanical Search on Shelves using Lateral Access X-RAY*. 2020. arXiv: [2011.11696](https://arxiv.org/abs/2011.11696) [cs.RO].
- [54] Y. Wang, Y. Li and J. Wang. “3D Modeling and Computer Graphics in Virtual Reality”. In: *Journal of Physics: Conference Series* 1007.1 (2018), p. 012010. DOI: [10.1088/1742-6596/1007/1/012010](https://doi.org/10.1088/1742-6596/1007/1/012010).

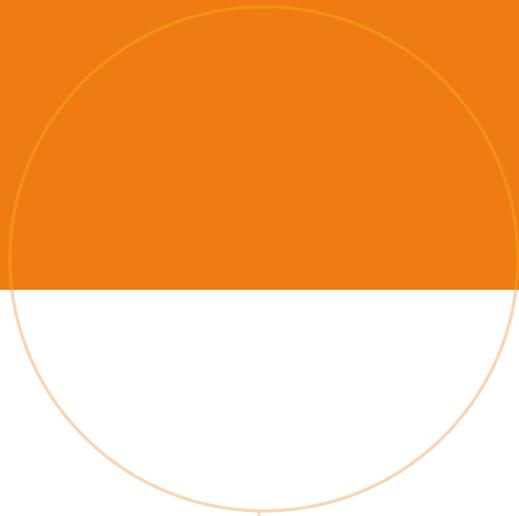
- [55] S. Kumar, S. Kumar and S. K. Jena. “Mixed Reality Meets Procedural Content Generation in Video Games”. In: *2021 International Conference on Computer Communication and Informatics (ICCCI)*. 2021, pp. 1–5. DOI: [10.1109/CCINF053287.2021.9458584](https://doi.org/10.1109/CCINF053287.2021.9458584).
- [56] Jessica V. *Procedural Generation*. https://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html. 2018.
- [57] Aristid Lindenmayer. “Mathematical models for cellular interactions in development I. Filaments with one-sided inputs”. In: *Journal of Theoretical Biology* 18.3 (1968), pp. 280–299. ISSN: 0022-5193. DOI: [https://doi.org/10.1016/0022-5193\(68\)90079-9](https://doi.org/10.1016/0022-5193(68)90079-9). URL: <https://www.sciencedirect.com/science/article/pii/0022519368900799>.
- [58] Pascal Müller et al. “Procedural Modeling of Buildings”. In: *ACM Trans. Graph.* 25 (July 2006), pp. 614–623. DOI: [10.1145/1141911.1141931](https://doi.org/10.1145/1141911.1141931).
- [59] Jiajun Wu et al. “Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling”. In: *CoRR* abs/1610.07584 (2016). arXiv: [1610.07584](https://arxiv.org/abs/1610.07584). URL: <http://arxiv.org/abs/1610.07584>.
- [60] Roger D Peng. “Reproducible research in computational science”. In: *Science* 334.6060 (2011), pp. 1226–1227.
- [61] Hans E Plesser. “Reproducibility vs. replicability: a brief history of a confused terminology”. In: *Frontiers in neuroinformatics* 11 (2018), p. 76.
- [62] Victoria Stodden. “The case for replication in computational science”. In: *The Princeton guide to ecology*. Princeton University Press, 2012, pp. 244–249.
- [63] Gary King. “Replication, replication”. In: *PS: Political Science & Politics* 28.3 (1995), pp. 444–452.
- [64] Radu Bogdan Rusu et al. “Towards 3D Point Cloud Based Object Maps for Household Environments”. In: *Robotics and Autonomous Systems* 56.11 (2008), pp. 927–941.
- [65] Zhenyu Jiang et al. “Synergies between affordance and geometry: 6-dof grasp detection via implicit representations”. In: *CoRR* abs/2104.01542 (2021).
- [66] *iGibson challenge 2021*. 2021. URL: <http://svl.stanford.edu/igibson/challenge.html>.
- [67] Andrey Kurenkov et al. “Semantic and geometric modeling with neural message passing in 3d scene graphs for hierarchical mechanical search”. In: *CoRR* abs/2012.04060 (2020).
- [68] Qianqian Wang et al. “Ibrnet: Learning multi-view image-based rendering”. In: *CoRR* abs/2102.13090 (2021).

- [69] Shaoqing Ren et al. “Faster R-CNN: towards real-time object detection with region proposal networks”. In: *CoRR* abs/1506.01497 (2015).
- [70] Nathan Morrical et al. “Nvisii: A scriptable tool for photorealistic image generation”. In: *CoRR* abs/2105.13962 (2021).
- [71] *Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/Manual/UnityManual.html> (visited on 25/04/2023).
- [72] *Blender 3.5 Reference Manual*. 2023. URL: <https://docs.blender.org/manual/en/latest/> (visited on 25/04/2023).
- [73] *Mesh and Object Deformers for Unity 3D*. <https://assetstore.unity.com/packages/tools-and-object-deformers-for-unity-3d-81427#description>. Version 2.1.1. Accessed: 2023-04-19. July 2019. URL: <https://wixarexperience.com/>.
- [74] Tudor Barbu. “Variational Image Denoising Approach with Diffusion Porous Media Flow”. In: *Abstract and Applied Analysis* 2013 (Jan. 2013), e856876. DOI: [10.1155/2013/856876](https://doi.org/10.1155/2013/856876). URL: <https://www.hindawi.com/journals/aaa/2013/856876/>.
- [75] *Boolean Modifier — Blender Manual*. Accessed: 2023-04-18. 2023. URL: <https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/booleans.html>.
- [76] *CUDA Zone*. URL: <https://developer.nvidia.com/cuda-zone> (visited on 25/04/2023).
- [77] *Matplotlib: Visualization with Python*. URL: <https://matplotlib.org/> (visited on 25/04/2023).
- [78] *Performance*. URL: <https://matplotlib.org/stable/users/explain/performance.html> (visited on 25/04/2023).

APPENDICES

GITHUB REPOSITORIES

- Haakon Gunnarsli & Jonathan Brooks - *Benchmark* <https://github.com/masteroppgaven/Benchmark>
 - The code and data used to generate the diagrams for the benchmark. In addition, all the pictures generated can be found in full size.
- Bart Iver van Blokland - *libShapeDescriptor* <https://github.com/bartvbl/libShapeDescriptor>
 - A library consisting of implementations for the RIC1, QUICCI, SI, 3DSC, and FPFH descriptors. Both GPU and CPU.
- Haakon Gunnarsli & Jonathan Brooks - *libShapeDescriptor* <https://github.com/masteroppgaven/libShapeDescriptor>
 - A fork of the original libShapeDescriptor project, which includes a benchmark tool that compares the different shape descriptors to one another.
- Haakon Gunnarsli & Jonathan Brooks - *DatasetGenerator* <https://github.com/masteroppgaven/DatasetGenerator>
 - A project consisting of both Unity and Blender scripts, that modify objects in order to create comprehensive datasets.



 **NTNU**

Norwegian University of
Science and Technology