UNIVERSITY OF LINCOLN

Surface Reconstruction of Point Clouds

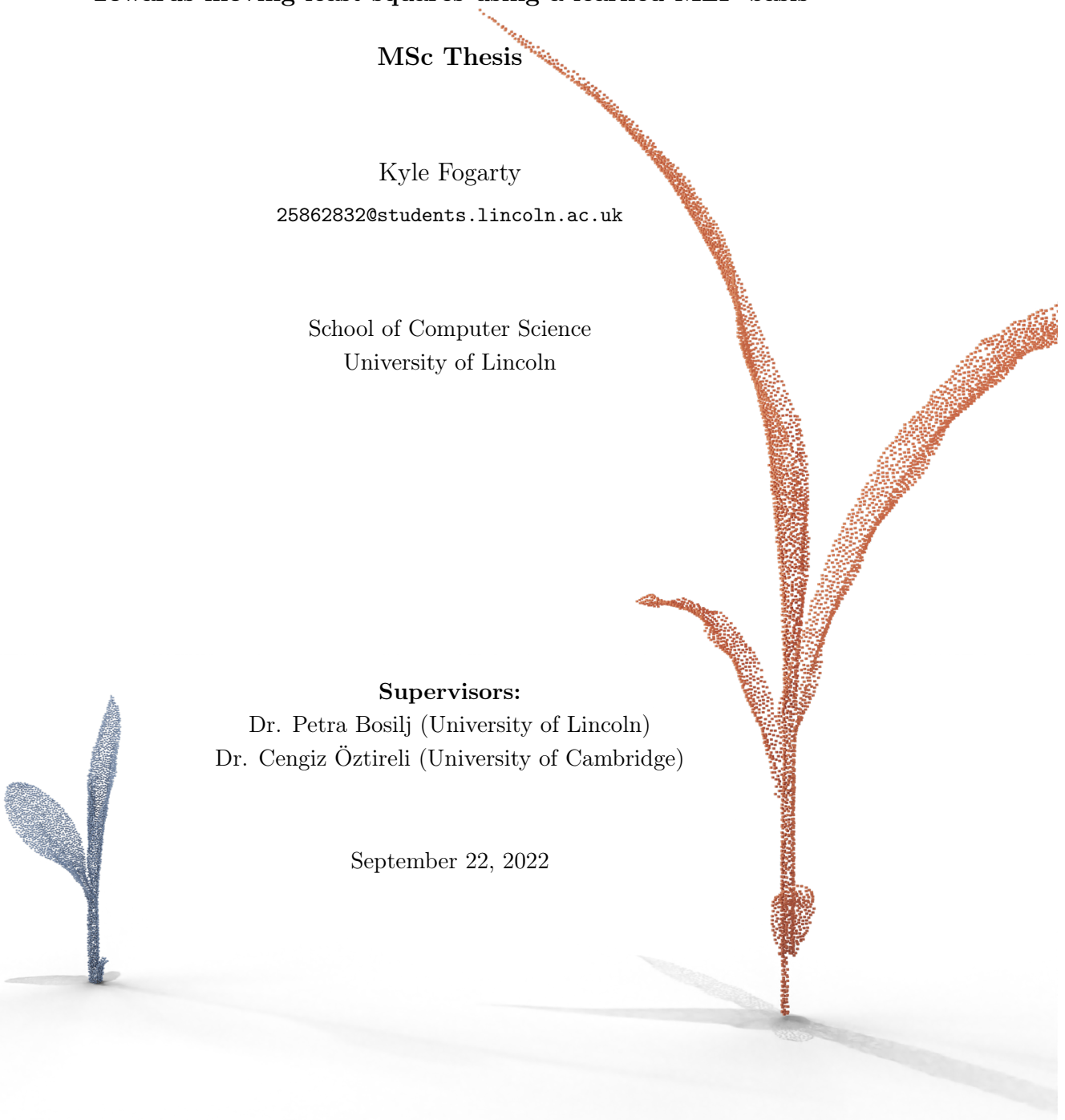
Towards moving least squares using a learned MLP basis

MSc Thesis

Kyle Fogarty

25862832@students.lincoln.ac.uk

School of Computer Science
University of Lincoln



Supervisors:
Dr. Petra Bosilj (University of Lincoln)
Dr. Cengiz Öztireli (University of Cambridge)

September 22, 2022

Acknowledgements

I would like to thank my supervisors Dr Petra Bosilj and Dr Cengiz Öztireli for their guidance and support throughout this project; I look forward to working with you both over the next three years of my PhD.

Financial Support

This work was supported by the Engineering and Physical Sciences Research Council [EP/S023917/1], as part of the Center for Doctoral Training in Agri-food robotics (Agri-FoRwArdS).

Abstract

The ability to accurately perceive and model natural structures is a central consideration of many application, including robotic. In this project we propose a new neural based approach to reconstruction: starting with the moving least squares reconstruction approach, we make use of a multi-layer perception to develop a set of learned basis functions for *optimal* reconstruction. To enable the development of the novel approach, we first construct a novel 2D point-set dataset we refer to as Shapeset. We then construct the `PyPointset` library, which implements three different, existing, surface reconstruction methods to act as benchmarks. Finally, we present our experimentation with the novel neural approach, highlighting strengths and current weaknesses, and provide an outlook to future work on this topic.

Contents

Chapter 1

Project Introduction

1.1 Introduction

Over the past three decades, point clouds have received increased attention as a representation of 3D geometry in the fields of computer graphics, computer vision, and robotics [1, 2, 3]. Driven by the advances in scanner technologies, such as Light Detection and Ranging (LiDAR) [4], point clouds are now relatively easy and fast to obtain. 3D scanning technologies are even making their way into main-stream commercial products, like smartphones and tablets.

While point clouds are typically the geometry representation of raw sensor data, they are not always the best representation for down-stream perception and graphics tasks. Classically, there are four methods of representing surfaces, which we illustrate in Fig. 1.1. While point-sets, meshes, and voxel-grids provide different forms of discrete representations for geometry, there exists a growing interest in using continuous implicit functions to representation geometry.

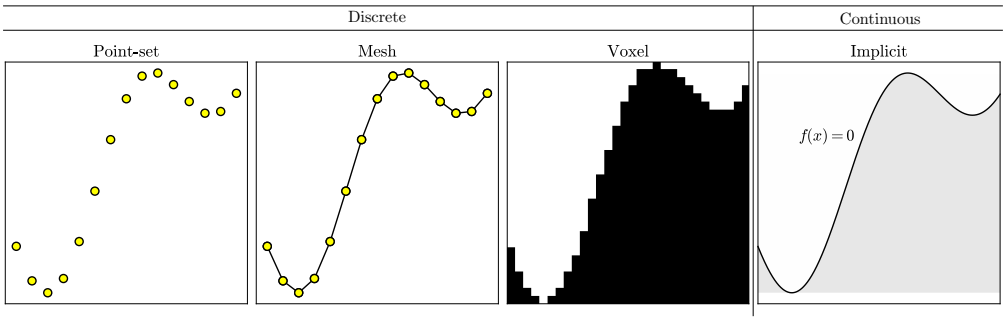


Figure 1.1: Shows existing representations of scanned 3D data. Typically, point-sets are obtained from raw sensors. Mesh and voxel representations are discrete but can be directly visualised. Implicit representations are continuous and have flexible topology, but require further processing to be visualised.

While point-sets are lightweight and flexible, they carry no information about an objects topology. Moreover, implicit surface representations have the benefit of

flexible topology, meaning the space of shapes they can represent is large compared with other representations. Once an implicit surface is defined, algorithms like March Cubes [5] allow for the efficient transformation of implicit surfaces into meshes or voxel grids, the discretisation step required to rendered objects. Implicit surface representations also benefit from high memory efficient because only the parameters of the implicit function are required to be stored for future reconstructions.

1.2 The Surface Reconstruction Problem

In this project, we consider the problem of obtaining an implicit representation of a surface given a point-set representation; we will refer to this problem as *surface reconstruction*. This idea is captured in Fig. 1.2, where we aim to go from a discrete point-set shown on the right to an implicit function representation in the center. The implicit function can then be easily meshed using existing tools to produce the rendered mesh surface on the left.

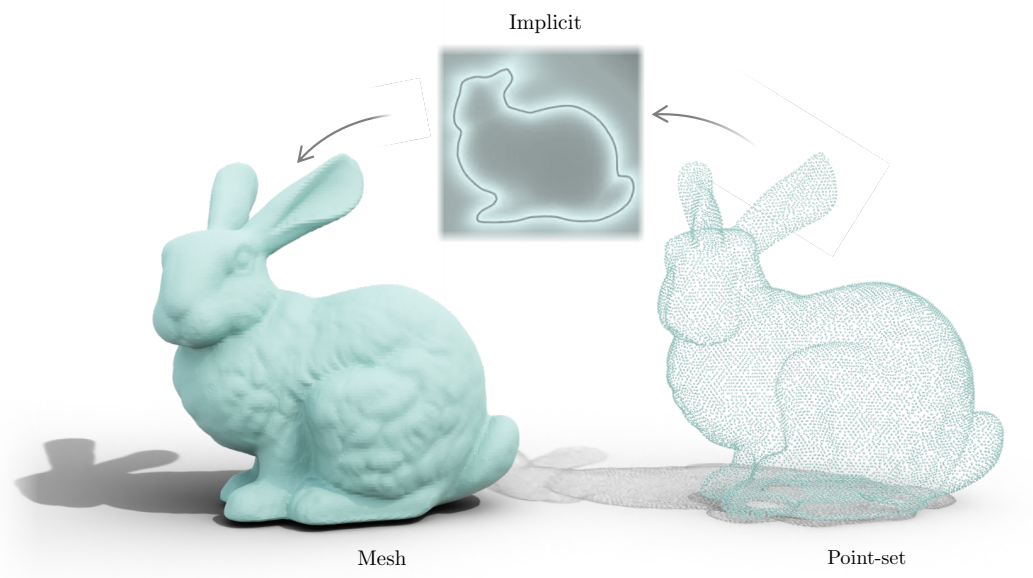


Figure 1.2: Scene shows two renders of the the Stanford Bunny [6], one of the most commonly used test models in computer graphics. The render on the left has a surface defined by a mesh. The render on the right shows a discretised point-set representation of the surface. The aim of point-set surface reconstruction is to produce algorithms that can take the input on the right and produce an implicit representation, which can then be used to generate the mesh on the left.

The problem of surface reconstruction is mathematically ill-posed: there exists an infinite number of surfaces that could have been generated by a given point-set,

especially if there exists noise in the acquisition. The problem is regularised by imposing so-called priors, constraints that reduce the space of possible solutions. In this project, we investigate a novel method of surface reconstruction that fuses a well known *local* surface smoothness prior with data-driven methods.

1.3 Project Aims & Objectives

The aim of this project is to conduct preliminary work on a novel neural approach to point-set surface reconstruction based on the moving least squares approximation framework. The long-term aim is to develop an approximation framework that is capable of producing high-fidelity reconstruct of 3D natural structures. However, in this project we restrict our attention to reconstruction of 2D surfaces from point-sets.

To achieve our aim, we have three objectives we seek to be completed within this project:

1. Explore and present the underlying theory of both surface reconstruction and the novel reconstruction approach, namely the areas: differential geometry, least squares function approximation, and MLP neural networks.
2. Explore current state-of-the-art approaches to surface reconstruction, and implement several approaches to act as benchmark methods.
3. Implement the novel approach to surface reconstruction in 2D and evaluate it against the benchmark approaches from the previous objective.

1.4 Contributions

The principle contributions of this project, which we present in this thesis are:

1. The development of a novel 2D point-set dataset and dataset creation pipeline called the Shapeset dataset (Chapter 5).
2. The development of a 2D point-set reconstruction library `PyPointset`, which implements three different approaches to point-set reconstruction (Chapter 6).
3. The preliminary development of our novel surface reconstruction approach (Chapter 7).

1.5 Report Structure

The thesis is split into 8 Chapters. In Chapter 2, we provide a brief introduction to differential geometry and implicit surfaces; these topics theoretically underpin the project. Then, in Chapter 3 we introduce two forms of function approximators: we study the least squares function approximation, with a particular emphasis on the moving least squares framework. Then, we also introduce the multi-layer perceptron (MLP) neural network and the method by which MLPs are *trained*. In Chapter 4, we provide the reader with a detailed literature review of surface reconstruction methods, focusing on methods that use either smoothness priors or data-driven priors to regularise the reconstruction problem. In Chapter 5, we present the development of a novel 2D point-sets dataset we call Shapeset. In Chapter 6, we then present the work conducted on building the `PyPointset` reconstruction library which provides three benchmark methods for 2D point-set reconstruction. In Chapter 7, we present the initial work conducted on implementing a novel surface reconstruction method in both 1 and 2 dimensions. Finally, in Chapter 8 we present our final conclusions about the project and provide the reader with a plan for future work.

1.6 Assumed Knowledge

While every effort has been made to ensure this thesis is self-contained, some knowledge must be assumed as known. In terms of mathematics, its is expected that the reader has a confident knowledge of *multivariate calculus* and *linear algebra* such as presented in [7] and [8], respectively.

1.7 A Brief Note On Nomenclature

As the methods we develop in this project will mainly aim at dealing with the 2D case of reconstruction, we use the word *point-set* to refer to any scattered set of data-points in $\mathbb{R}^n$, including $n = 2$, and the word *point cloud* to referred explicitly to a point-set in 3D.

Chapter 2

Differential Geometry

In this Chapter, we provide the reader with the preliminary theoretical knowledge needed to understand the ensuing Chapters within this report. In Section 2.1, we briefly introduce the concept of a manifold surface, before introducing some of the useful *differential properties* that can be defined under the differential geometry framework.

Differential geometry is the mathematical framework used to describe surfaces and their differential properties. While the field of differential geometry is vast, we concentrate on only a few core ideas within this Chapter. The central notion we wish to describe with differential geometry is how we can parameterise a surface such that we can employ ideas from multivariate calculus *on the surface*. In this section, we give a brief overview of the concept of a *manifold* surface, the object that allows us to employ idea from calculus. Then using this definition, we will introduce the definition of the *tangent plane* and the *surface normal*.

2.1 Manifold Surfaces

The definition of manifold relies critically on the definition of homeomorphism, therefore, we first introduce the definition of a homeomorphic mapping in Def. 2.1.

Definition 2.1: (Homeomorphism [9]).
A function $f : X \to Y$ is a homeomorphism if it satisfies the three following criteria: (1) f is a bijection, (2) f is continuous, and (3) f^{-1} is continuous.

The bijective property of a homeomorphism ensures that each point in $x \in X$ has a $y \in Y$ such that $f(x) = y$ and $f^{-1}(y) = x$. Given the definition of a homeomorphic function, we can then define a manifold surface.

Definition 2.2: (Manifold Surface [10]).
A manifold surface $\mathcal{M}$ is one that is locally homeomorphic to Euclidean space: that is, for every $x \in \mathcal{M}$, there exists an $r > 0$ such that an open ball, $B_x(r)$, centered at x with radius r intersected with $\mathcal{M}$ is homeomorphic to an open disc in $\mathbb{R}^n$.

The geometric intuition of a manifold surface is shown in Fig. 2.1: for every point on the torus we can define a function that maps from the surface to $\mathbb{R}^2$. These mappings are sometimes referred to as charts, with the collection of all charts that form the full manifold surface, then referred to as an atlas.

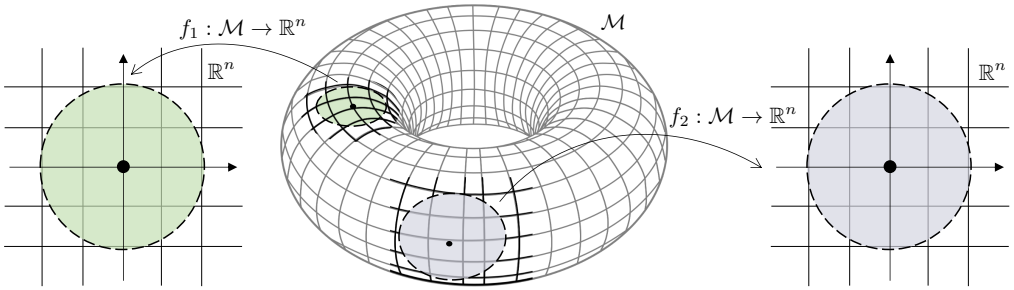


Figure 2.1: Illustration of the concept of a manifold: there exists a homeomorphic mapping for every point on the manifold $\mathcal{M}$ in $\mathbb{R}^n$.

The power of viewing surfaces as manifolds is that we can always define a local coordinate systems which resembles Euclidean space. Then, the established instruments of calculus can be enacted through the mapping, from from Euclidean space onto the surface.

2.1.1 Differential Properties of Manifolds

A manifold that maps from a *surface* to $\mathbb{R}^n$ is typically referred to as n-manifold. Suppose we have a 2-manifold embedded in 3 dimensional space (like the torus in Fig. 2.1), then as there exists a homeomorphic mapping we can form a vector valued function $\boldsymbol{p}(u, v)$ that maps from $\mathbb{R}^2$ to a local neighborhood on the surface. Local basis vectors $\boldsymbol{p}_u$ and $\boldsymbol{p}_v$ can then be defined via:

$$\boldsymbol{p}_u = \frac{\partial}{\partial u}\boldsymbol{p}(u, v) \quad \text{and} \quad \boldsymbol{p}_v = \frac{\partial}{\partial v}\boldsymbol{p}(u, v),$$

which operate in the Euclidean space. This idea is illustrated in Fig. 2.2. Assuming that $\boldsymbol{p}_u$ and $\boldsymbol{p}_v$ are not collinear, the surface normal on the manifold can then be defined via:

$$\hat{\boldsymbol{n}}(u,v) = \frac{\boldsymbol{p}_u \times \boldsymbol{p}_v}{\|\boldsymbol{p}_u \times \boldsymbol{p}_v\|}. \tag{2.1}$$

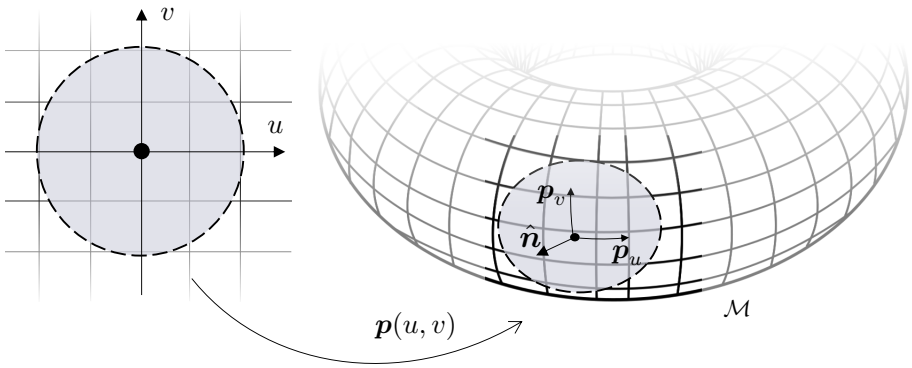


Figure 2.2: Illustration of the mapping from $\mathbb{R}^2$ to the manifold which can be used to define the local surface normal.

The salient point from this section is that under the assumption the surface we are interested in is manifold, we can always construct a local coordinate system such that the surface resembles Euclidean space. Then, from the mapping between Euclidean space and the surface, differential properties can be defined.

2.2 Implicit Surface

In Chapter 1, we briefly mentioned that our aim is to produce an implicit function representations of the surface given a point-set. Implicit functions typically have the domain of the the embedding space of the surface (i.e., $\mathbb{R}^2$ in this project) and codomain of $\mathbb{R}$. Then, the implicit surface is then defined as one of the level-sets of the implicit function, which we define more precisely in Def. 2.3.

Definition 2.3: (Implicit Surface Definition).
Let $\mathcal{S}$ denote an implicit surface, then this surface represents the 0-isocontour of some unknown implicit function $f(\boldsymbol{x})$, such that:

$$\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) = 0\}, \tag{2.2}$$

with the condition that $\nabla f(\boldsymbol{x}) \neq \mathbf{0}$ for all $\boldsymbol{x} \in \mathcal{S}$.

The final condition of non-zero gradient is important as it is the condition that ensures the implicit surface is manifold. Using the *implicit function theorem* [11], we can state that for each point of the implicit function that has a value of zero

but does not have zero gradient (i.e., $f(\boldsymbol{x}) = 0$ and $\nabla f(\boldsymbol{x}) \neq 0$), then locally to that point the 0 level-set looks like a graph which implies that globally the surface is manifold.

2.2.1 Signed Distance Function

A common choice for the implicit function $f(x)$ is a signed distance function (SDF). A signed distance function produces a scalar field, $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$, such that for each point $\boldsymbol{x} \in \mathbb{R}^n$, the value of the function $f(\boldsymbol{x})$ gives the signed geodesic distance to the nearest point on the surface $\mathcal{S}$. This idea is illustrated for the simple example of the unit circle in Fig. 2.3, where we have used the convention of inside the surface having negative sign and outside the surface having positive sign.

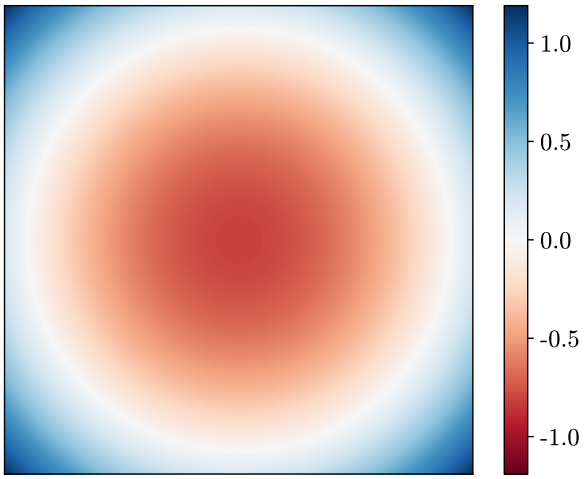


Figure 2.3: Illustration of a signed distance function for the unit circle. Points outside of the unit circle are positive in sign (blue) and points within the unit circle are negative in sign (red). The surface of the circle is defined by the white region where the function is exactly zero.

In this work, we mostly consider implicit surfaces defined by signed distance functions of the point-sets, and so we provide a more formal definition that will be used throughout this project.

Definition 2.4: (Signed Distance Function).
Let Ω denote the region of occupied space, with boundary $\partial\Omega$. As $\Omega \subset \mathbb{R}^n$, a metric space can be defined by $\mathbb{R}^n$ equipped with the ℓ_2 vector norm. Then a signed distance function, $f(\boldsymbol{x})$, is defined by the piece-wise definition:

$$f(\boldsymbol{x}) = \begin{cases} -d(\boldsymbol{x}, \partial\Omega) & \text{if } \boldsymbol{x} \in \Omega \\ d(\boldsymbol{x}, \partial\Omega) & \text{if } \boldsymbol{x} \in \bar{\Omega} \end{cases} \tag{2.3}$$

where,

$$d(\boldsymbol{x}, \partial\Omega) = \inf_{\boldsymbol{\omega} \in \partial\Omega} \|\boldsymbol{x} - \boldsymbol{\omega}\|, \tag{2.4}$$

and inf refers to the infimum over the set of boundary points.

Chapter 3

Function Approximation

Approximating functions is central to robust point-set surface reconstruction. In Section 3.1, we provide the reader with an introduction to the ordinary least squares and the moving least squares function approximation frameworks. Then, in Section 3.2 we provide a rigorous introduction to neural networks as function approximates and discuss the effect of layer width and depth on expressivity.

3.1 Least Squares Approximation

Core to implicit surface reconstruction is the technique used to approximate the implicit function that represents the surface. The least squares approach is a very common framework for function approximation when the system of equations is *over-determined*[1], in-part due to being extremely well studied. The least squares approach also has the advantage of having a number of provable properties, including the equivalence to the statistical technique of Maximum Likelihood Estimation (MLE) with a Gaussian likelihood model (i.e., the use of least squares implies a Gaussian model for observation noise) [12].

In this section, we provide an introduction to the moving least squares framework. Moving least squares (MLS) is central to the approach taken to surface reconstruction in this project. While least squares is a very common function approximation framework, we motivate MLS by first introducing ordinary least squares (OLS). Throughout this section we will make use of a test problem, which we introduce in section 3.1.1.

[1] A system of equations is over-determined if there exists more unique equations than there are unknowns.

3.1.1 The Test Problem

For simplicity, we will work in 1D. In general, we will suppose that we have a set $\mathcal{S}$ of n data-pairs, which are samples of some unknown function $f(x)$, such that,

$$\mathcal{S} = \{(x_i, y_i) \mid y_i = f(x_i) \text{ for } i = 1, 2, .., n\}. \tag{3.1}$$

In this test example we will use the function $f(x) = \sin(8x) + \sin(6.4x)$, and we will let $x_i = \frac{i}{10}$ for $i = \{0, 1, ..., 10\}$. These data-points are plotted in Fig. 3.1. Given $\mathcal{S}$, our aim is to recover an approximation to $f(x)$, which we denote $\hat{f}(x)$.

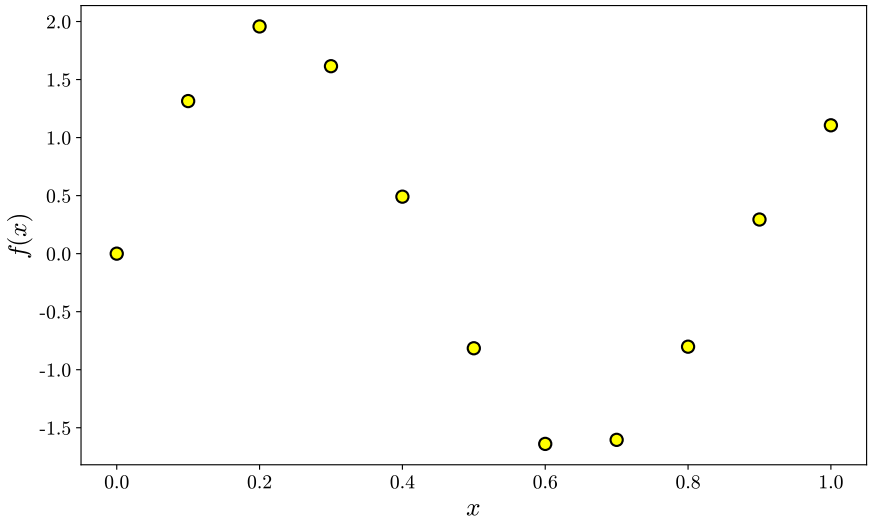


Figure 3.1: Shows a dataset of 11 samples from the function $f(x) = \sin(8x) + \sin(6.4x)$.

3.1.2 A Note On Basis Functions

The least squares approximation framework provides a criterion for determining coefficients of some functional, but the form of the functional must be specified. The choice of the functional to be optimised is very important in determining the resulting approximation, we will show this in the following sections. It is common for $\hat{f}(x)$ to be written as a linear combination of *basis functions*, such that:

$$\hat{f}(x) = \sum_i \alpha_i \phi_i(x), \tag{3.2}$$

where α_i are constants to be determined and $\phi_i : \mathbb{R}^n \to \mathbb{R}$ are the basis functions. The choice of basis function defines the so called function approximation space.

A common class of basis functions are the algebraic polynomials such that our approximation is represented by

$$\hat{f}(x) = \alpha_0 + \sum_{n=1}^{N} \alpha_n x^n, \tag{3.3}$$

for some $N \in \mathbb{N}$. The algebraic polynomials are a good candidate for basis functions as Taylor's theorem shows that any suitably smooth function can be expressed locally in terms of a linear combination of polynomials [13]. However, we note there exist many other families of basis function which have different applications depending upon the problem domain.

3.1.3 Ordinary Least Squares

Ordinary least squares is the most basic form of least squares approximation. Originally proposed by Legendre in 1805 [14], it continues to be a corner stone technique in function approximation. The optimisation function is given in Def. 3.1 below.

Definition 3.1: (Ordinary Least Squares).

Let $\hat{f}(x)$ be the functional, then OLS seeks to optimise the parameters of $\hat{f}(x)$ with respect to the objective function:

$$\sum_{\mathcal{S}} \|\hat{f}(x_i) - y_i\|_2^2. \tag{3.4}$$

OLS & The Test Problem

To implement OLS for the test problem described in Section 3.1.1, we begin by selecting the function space (i.e., the functional $\hat{f}(x)$). To use the algebraic polynomials defined in Eq. (3.3), we must select the number of polynomial terms N. As an example, let $N = 1$; then the approximation function is given by $\hat{f}(x) = \alpha_0 + \alpha_1 x$, where we aim to recover the coefficients α_0 and α_1 from the scattered data points $\mathcal{S}$.

Formulating the optimisation problem, we wish to minimise the objective function E given by:

$$E = \sum_{(x_i,y_i)\in\mathcal{S}} \left[\hat{f}(x_i) - y_i\right]^2 = \sum_{(x_i,y_i)\in\mathcal{S}} \left[\alpha_0 + \alpha_1 x_i - y_i\right]^2, \tag{3.5}$$

which can be viewed as a function of α_0 and α_1. Differentiating E with respect to α_0 and α_1 and equating the resulting expressions to zero allows for the coefficients α_0 and α_1 to be determined. The derivative:

$$\frac{\partial E}{\partial \alpha_0} = 2 \sum_{(x_i,y_i)\in\mathcal{S}} [\alpha_0 + \alpha_1 x_i - y_i] = 0, \tag{3.6}$$

yields the equation,

$$n\alpha_0 + \alpha_1 \sum_{(x_i,y_i)\in\mathcal{S}} x_i = \sum_{(x_i,y_i)\in\mathcal{S}} y_i, \tag{3.7}$$

where n defines the number of data-pairs in the set $\mathcal{S}$. Similarly,

$$\frac{\partial E}{\partial \alpha_1} = 2 \sum_{(x_i,y_i)\in\mathcal{S}} [\alpha_0 + \alpha_1 x_i - y_i]\, x_i = 0, \tag{3.8}$$

yields the equation,

$$\alpha_0 \sum_{(x_i,y_i)\in\mathcal{S}} x_i + \alpha_1 \sum_{(x_i,y_i)\in\mathcal{S}} x_i^2 = \sum_{(x_i,y_i)\in\mathcal{S}} x_i y_i. \tag{3.9}$$

To efficiently solve Eq. (3.7) and Eq. (3.9), to find the coefficients α_0 and α_1, it is helpful to employ linear algebra. Rewriting Eq. (3.7) and Eq. (3.9) in matrix vector form

$$\begin{bmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \end{bmatrix}, \tag{3.10}$$

the coefficients can be easily found by computing the matrix inverse, such that

$$\begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \end{bmatrix}. \tag{3.11}$$

Having found the coefficients α_1 and α_2, the approximation defined by $\hat{f}(x) = \alpha_0 + \alpha_1 x$ can be plotted; this is shown in Fig. 3.2 in orange. The same procedure can be repeated for any finite number of polynomial terms. Fig. 3.2 also shows the OLS approximation for $N = 2$ in blue , $N = 3$ in green, and $N = 4$ in purple. We note that as we increase the number of polynomial terms (i.e., the number of degrees of freedom), the residual error between the approximation and the data appears to decreases.

3.1.4 Moving Least Squares

One of the problems of the OLS approach is it performs global optimisation; the approximation coefficients are assumed to be independent of the spatial location.

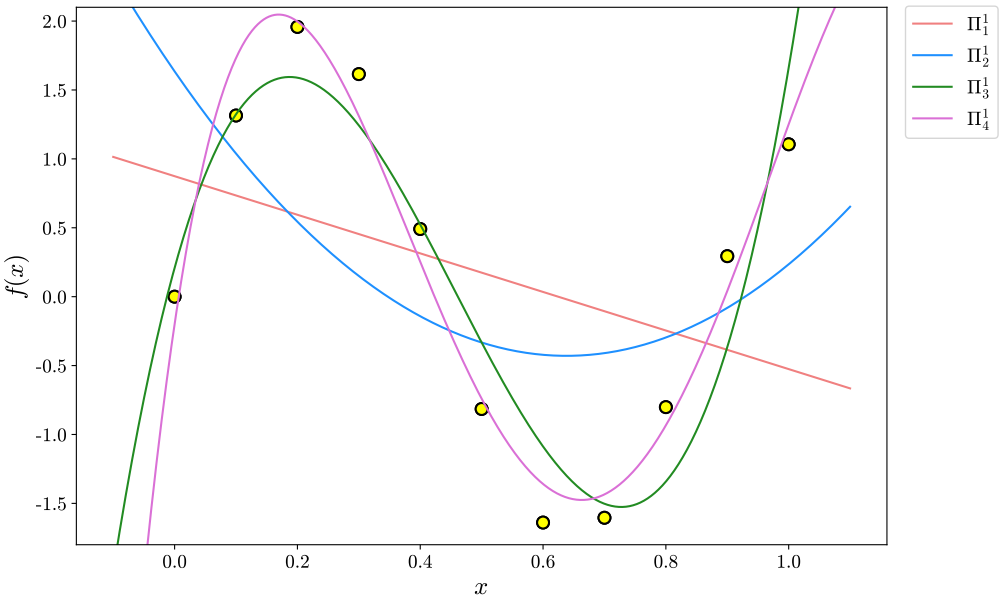


Figure 3.2: Plot shows the OLS approximation to the test problem described in section 3.1.1 using 4 different sets of basis functions. The compact polynomial notation used in the legend Π_n^d defines the n^{th} degree polynomial in d dimensions, that was used as the functional basis.

This, in-part, explains why higher degrees of freedom appear to lead to higher quality reconstructions under OLS. The core idea of MLS is to remove the requirement for the coefficients to be constant over the entire domain and instead let them vary, with the optimisation problem localised via a weight function. We give the definition of MLS optimisation below.

Definition 3.2: (Moving Least Squares).
Let $\hat{f}(x)$ be the functional to be optimised, then for each x in the approximation domain, the objective function is given by:

$$\sum_i \|\hat{f}(x_i) - y_i\|_2^2 w(x, x_i), \tag{3.12}$$

where $w(x, x_i)$ is the weighting function. In this project, we will exclusively consider $w(x, x_i)$ to be the stationary un-normalised Gaussian defined by

$$w(x, x_i) = \exp\left(-\frac{\|x - x_i\|_2^2}{\sigma^2}\right). \tag{3.13}$$

A very important consideration of the above definition is the choice of the variance, σ^2, of the Gaussian weighting term. The variance effectively controls the

spread of the weighting distribution, with higher variances giving a wider spatial weighting (see Fig. 3.3).

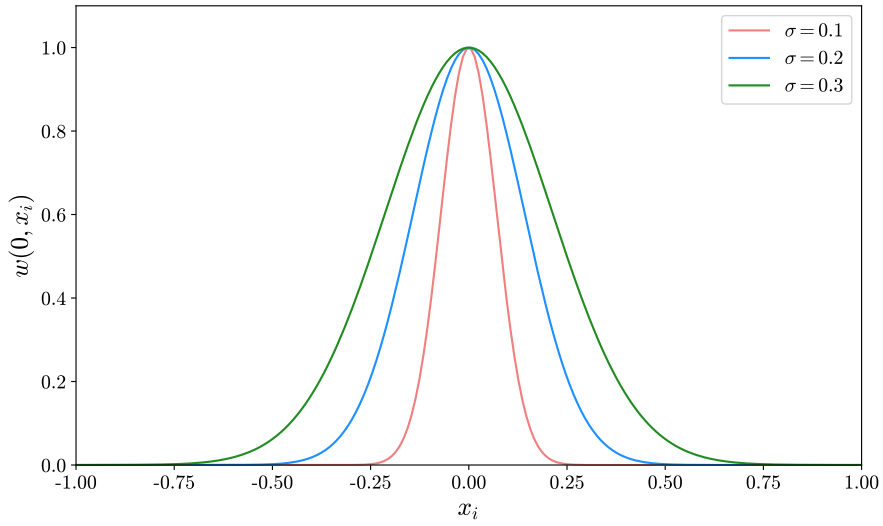


Figure 3.3: Shows three weighting terms $w(x, x_i)$ centered at 0, with different values of σ. Note that the larger the value of σ, the larger the spread of the weight distribution.

Vector Space Approach

Having introduced the MLS objective function, we now show how MLS can be implemented efficiently using linear algebra. Using matrix-vector notation, the functional represented by a linear combination of basis functions can be expressed in the form

$$\hat{f}(x) = \boldsymbol{b}(x)^T \boldsymbol{\alpha}(x) \tag{3.14}$$

where $\boldsymbol{b}(x) = [b_0(x), b_1(x), \cdots, b_m(x)]$ and $\boldsymbol{\alpha}(x) = [\alpha_0(x), \alpha_1(x), \cdots, \alpha_m(x)]^T$. The objective function defined in Eq. (3.12), at a set-point x, can then be written in matrix-vector form such that we minimise

$$J = (B\boldsymbol{\alpha}(x) - \boldsymbol{y})^T W (B\boldsymbol{\alpha}(x) - \boldsymbol{y}) \tag{3.15}$$

where B is defined as:

$$B = [\boldsymbol{b}(x_1), \boldsymbol{b}(x_2), ..., \boldsymbol{b}(x_n)]^T, \tag{3.16}$$

which expands to give the matrix definition,

$$B = \begin{bmatrix} b_0(x_1) & b_1(x_1) & \cdots & b_m(x_1) \\ b_0(x_2) & b_1(x_2) & \cdots & b_m(x_2) \\ \vdots & \vdots & & \vdots \\ b_0(x_n) & b_1(x_n) & \cdots & b_m(x_n) \end{bmatrix}. \tag{3.17}$$

The vector $\boldsymbol{y}$ is defined as,

$$\boldsymbol{y} = [y_1, y_2, \ldots, y_n]^T, \tag{3.18}$$

and $W(x)$ is defined as,

$$W(x) = \begin{bmatrix} \omega(x - x_1) & 0 & \cdots & 0 \\ 0 & \omega(x - x_2) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \omega(x - x_n) \end{bmatrix}. \tag{3.19}$$

Then letting $A(x) = B^T W(x) B$ and $C(x) = B^T W(x)$, the matrix-vector objective function in Eq. (3.15) can then be differentiated with respect to the coefficient vector $\boldsymbol{\alpha}(x)$ such that,

$$\frac{\partial J}{\partial \boldsymbol{\alpha}} = A(x)\boldsymbol{\alpha}(x) - B(x)\boldsymbol{y} = 0, \tag{3.20}$$

which rearranges to give the matrix equation,

$$A(x)\boldsymbol{\alpha}(x) = C(x)\boldsymbol{y}. \tag{3.21}$$

Then assuming that the matrix $A(x)$ is not singular, the coefficients can be efficiently found via inverting the matrix such that

$$\boldsymbol{\alpha}(x) = [A(x)]^{-1} C(x)\boldsymbol{y}. \tag{3.22}$$

Hence, the approximation at each x is defined by,

$$\hat{f}(x) = \boldsymbol{b}(x)^T [A(x)]^{-1} C(x)\boldsymbol{y}. \tag{3.23}$$

MLS & The Test Problem

We now present the MLS approximation to the test problem stated in Section 3.1.1. Again, like OLS, we opt for a simple linear functional of the form $\hat{f}(x) = \alpha_0(x) + \alpha_1(x)x$, where $\alpha_0(x)$ and $\alpha_1(x)$ are now also functions of the spatial variable x. Solving Eq. (3.23) with four different values of σ^2, we yield

four approximations which are presented in Fig. 3.4. As MLS blends together Gaussian weighting terms, even know the functional was of a linear form, the resulting approximation can be highly non-linear; this is different from the OLS case where a linear function guarantees the approximation will also be linear.

The asymptotic behaviour of the σ^2 parameter is clear from Fig. 3.4, where small values of σ^2 lead to the interpolation of the data-points, in this case with piece-wise linear function. Moreover, larger values of σ^2 effectively reduces the MLS framework to that of OLS presented in Section 3.1.3. Clearly, there then exists an optimal value of σ^2 which produces the best approximation in terms of the smallest error; hand tuning this parameter, $\sigma^2 = 0.08$ appears to yield the best result.

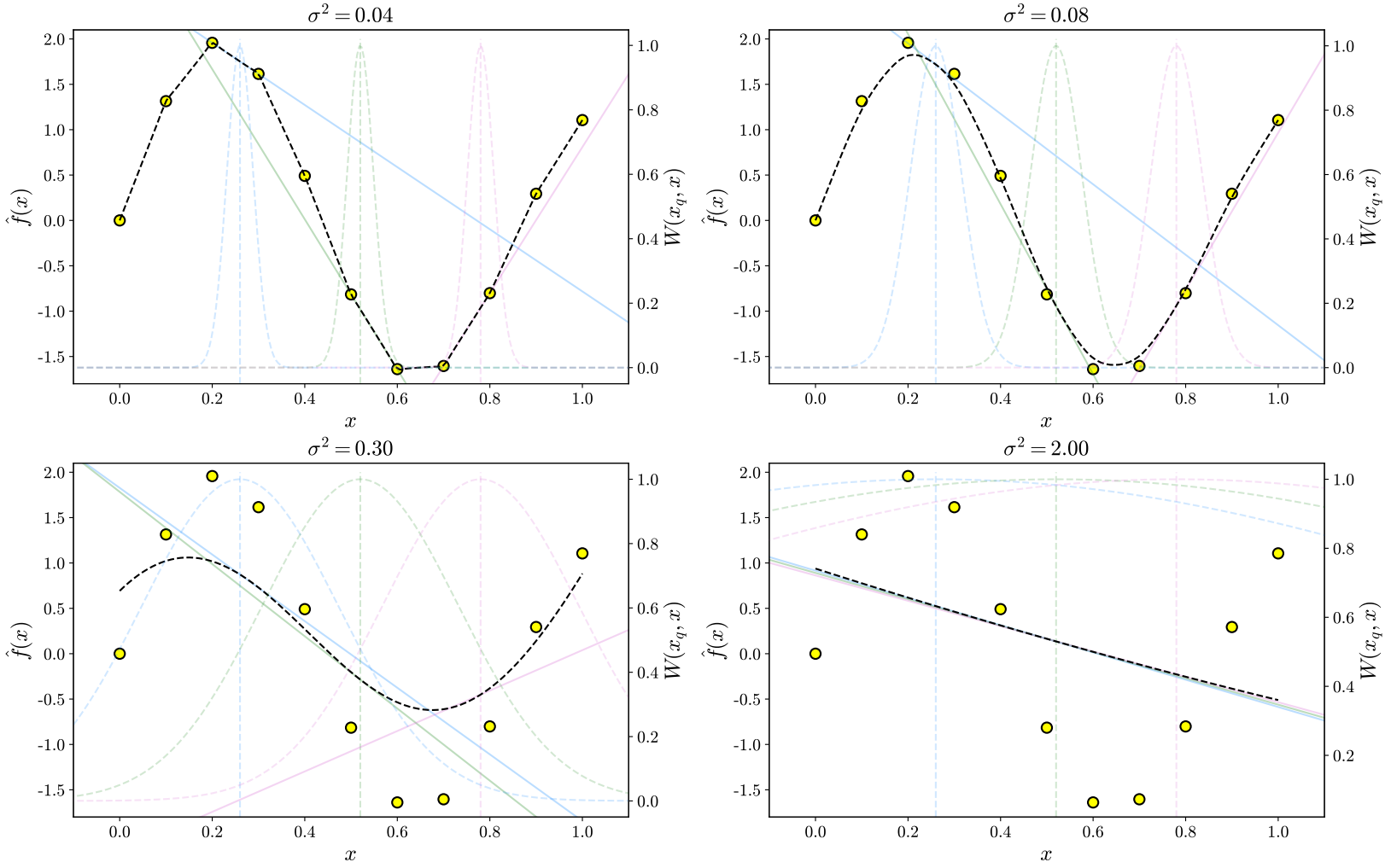


Figure 3.4: Shows approximations under the MLS framework, with varying spatial weighting variance. The weight function is plotted at three locations on the domain, and the linear approximation at those location, treating $\boldsymbol{\alpha}$ as constant, is also plotted. We note the asymptotic behaviour of the approximation: As $\sigma \to 0$, the function interpolates the points with piece-wise linear regions. Similarly, as $\sigma \to \infty$, the MLS framework reduces to OLS and a linear fit is obtained.

3.2 Neural Networks for Function Approximation

The final topic we consider in this Chapter is neural networks for function approximation. This Chapter introduces the multi-layer perception (MLP), a fully-connected feed-forward neural network, and introduces the framework by which they are trained. The concept of an MLP for function approximation is very similar to the ideas of function approximation introduced in the previous Section, however, where the function space in OLS was defined by a handful of parameters, MLPs typical have very large numbers of parameters (e.g., tens of thousands) which are *learned* via training. This large parameter space can give rise to expressive function approximators, and in the later part of this Section we will seek to develop a heuristic approach to understand expressivity via network architecture selection. The work in this Section builds upon the work originally presented in [15].

3.2.1 Introduction to Neural Networks

An MLP consists of neurons that are arranged into layers, the particular arrangement of neurons in these layers is what we referred to as the *network architecture*. Each of the neurons is modelled as activation functions $\sigma : \mathbb{R} \to \mathbb{R}$. Previous literature provides may candidates for these activation function [16, 17, 18] (for a comprehensive review see [19]), depending on the type of problem being tackled. In this project we will make use of the rectified linear activation unit (ReLU) which is defined by

$$\sigma(z) = \text{ReLU}(z) = \begin{cases} x & x > 0, \\ 0 & x \leq 0, \end{cases} = \max(0, x). \tag{3.24}$$

The ReLU activation function has a number of nice properties: The derivative is simple to compute being either 0 or 1, depending upon the value of x. The ReLU activation function has been extensively studied in-terms of its expressivity and the functions spaces it can represent, these ideas will be explored in Section 3.2.4.

3.2.2 Evaluating the MLP

Before we continue we must first introduce some notation. Throughout this section we make use of the notation introduced in [20], other than we opt for a zero-based counting system. Let L represent the number of layers within our network, then we index each layer with a zero-based layer number, l, such that $l \in \{0, 1, ..., L-1\}$. Similarly, let N_l represent the number of neurons at layer l, then we index each neuron of the layer with $n_l \in \{0, 1, .., N_l - 1\}$. This notation allows us to write the input to the j^{th} neuron at layer l as $z_j^{[l]}$ and the output

of the j^{th} neuron at layer l as $a_j^{[l]}$. We relate the input and output using the previously defined activation function such that:

$$a_j^{[l]} = \begin{cases} x_j & \text{for } j = 0, .., n_l \quad \text{if } l = 0 \\ \sigma\left(z_j^{[l]}\right) & \text{for } j = 0, .., n_l \quad \text{if } l = 1, .., L-1. \end{cases} \tag{3.25}$$

Note that the input at layer 0 is given by the input vector $\boldsymbol{x}$. An example MLP is shown in Fig. 3.5 which consists of one input layer of two neurons, one output layer of one neuron and two hidden layers each with three neurons.

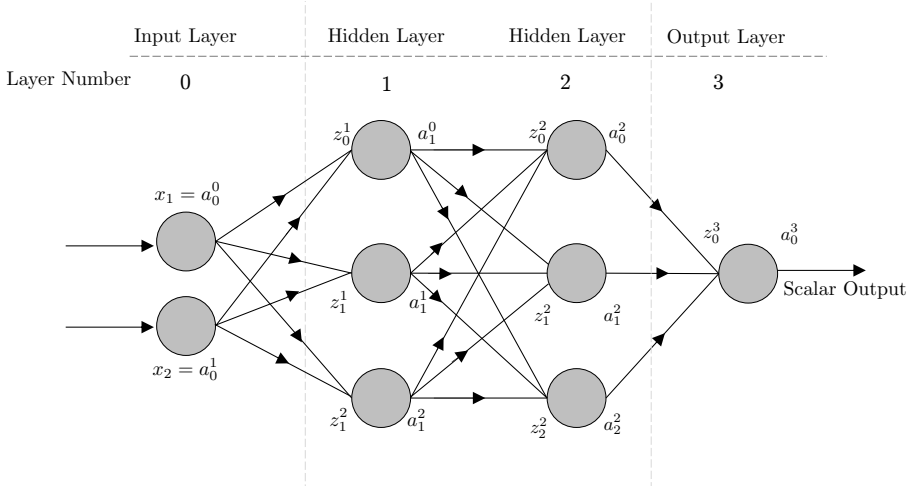


Figure 3.5: Fully connected MLP with 4 layers: one input layer, one output layer and two hidden layers. The input layer consists of two inputs $\boldsymbol{x} = (x_1, x_2)$ and the output is a single scalar value. $z_j^{[l]}$ and $a_j^{[l]}$ represent the input and output of the j^{th} neuron on the l^{th} layer, respectively.

Of course, as can be seen from Fig. 3.5, if a layer has more than one neuron within it, the next layer will have more than one scalar input terminating at its neurons and so we must introduce a way of combining these inputs into a single scalar value. We form the inputs of $z_j^{[l]}$ as a biased linear combination of the outputs in layer $l-1$ such that:

$$z_j^{[l]} = \sum_{k=0}^{n_{l-1}} \left(w_{jk}^{[l]} a_k^{[l-1]} \right) + b_j^{[l]} \quad \text{for} \quad l = 1, 2, .., L-1 \quad j = 0, ..., N_l - 1. \tag{3.26}$$

where $w_{jk}^{[l]}$ are the weights and $b_j^{[l]}$ is the biases at layer l. The weights and biases are then initialised. In this work we use a random initialisation at the beginning of training. While the network is trained, the weights and biases are adjusted, and once training has concluded they remain fixed thereafter. To compact the notation, we may write $\mathbf{b}^{[l]} \in \mathbb{R}^{n_l}$ as the vector of biases and $W^{[l]} \in \mathbb{R}^{[n_l]} \times \mathbb{R}^{[n_{l-1}]}$

as the matrix of weights at layer l, such that Eq. (3.26) may be written as

$$\mathbf{z}^{[l]} = W^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}. \tag{3.27}$$

where $\mathbf{z}^{[l]} \in \mathbb{R}^{n_l}$ is the vector of inputs at layer l. Similarly, by forming a vector of outputs $\mathbf{a}^{[l]} \in \mathbb{R}^{n_l}$ we may rewrite Eq. (3.25) using Eq. (3.27) to find:

$$\mathbf{a}^{[l]} = \begin{cases} \boldsymbol{x} & \text{if } l = 0 \\ \sigma\left(W^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}\right) & \text{if } l = 1, .., L-1. \end{cases} \tag{3.28}$$

where $\sigma(\cdot)$ is Eq. (3.28) is understood to act on each component of its vector input independently. This is referred to as the *feed-forward* algorithm; given the weights and biases this allows us to propagate an input $\boldsymbol{x}$ through the network and find its output. The weights and biases are found by *training* the network, which we will discuss in the next subsection.

3.2.3 Training the MLP

In this Section we will look how we determine the *optimum* weights and biases for our network through training. Training a neural network requires we define a loss function, similar to how we defined objective functions for OLS and MLS. We must also define the method of optimisation, which is typically a gradient descent based method. Finally, we use the back-propagation algorithm to update the weights and biases within the network.

The Loss Function

In this project, we consider only the *supervised* paradigm of machine learning; by supervised training, we mean that we provide the network with a set of training examples, $\boldsymbol{x}^i \in \{\boldsymbol{x}^1, \boldsymbol{x}^2, ..., \boldsymbol{x}^N\}$, and corresponding labels $\mathbf{y}^i \in \{\mathbf{y}^1, \mathbf{y}^2, ..., \mathbf{y}^N\}$ and we aim to minimise some loss function. As the MLPs we consider in this project are working in coordinate space, a sensible first choice for our cost function is the squared ℓ_2 norm

$$C = \frac{1}{N}\sum_{i=i}^{N} \frac{1}{2}||\mathbf{y}^i - \mathbf{a}^{[L-1]}(\boldsymbol{x}^i)||_2^2 \tag{3.29}$$

where N is the number of training examples. By noting that the argument of the ℓ_2 norm is a function of all the weights and biases of the network we may write

$$C = C(\mathbf{P}), \tag{3.30}$$

where $\mathbf{P} = \left(W^{[1]}, .., W^{[L-1]}, \mathbf{b}^{[1]}, ..., \mathbf{b}^{[L-1]}\right)$ is a vector of all the weights and biases in the network. It is our aim to find $\mathbf{P}$ such that $C(\mathbf{P})$ is minimised.

Stochastic Gradient Descent

In order to solve this optimisation problem we will make use of the stochastic gradient decent (SGD) algorithm; we first introduce the gradient decent algorithm and then we will show how it can be modified for efficiency.

The initialisation of the parameters $\mathbf{P}$, defines some cost $C(\mathbf{P})$. Our aim is to construct a new vector $\mathbf{P}+\Delta\mathbf{P}$ such that $C(\mathbf{P}+\Delta\mathbf{P})$ is minimised. By performing a first order Taylor expansion we find

$$C(\mathbf{P}+\Delta\mathbf{P}) \approx C(\mathbf{P}) + (\nabla C(\mathbf{P})) \cdot \Delta\mathbf{P} \tag{3.31}$$

where ∇ is the gradient operator that acts over each component of $C(\mathbf{P})$. Then via the *Cauchy-Schwartz inequality* we find that an appropriate $\Delta\mathbf{P}$ is given by

$$\Delta\mathbf{P} = -\eta\nabla C(\mathbf{P}) \tag{3.32}$$

where η is the learning rate, which affects the rate of convergence. From Eq. (3.32) we yield our update rule for $\mathbf{P}$ as

$$\mathbf{P}^{i+1} = \mathbf{P}^i - \eta\nabla C(\mathbf{P}^i) \tag{3.33}$$

where i refers to the iteration number; this is the *gradient decent algorithm*. Computing $\nabla C(\mathbf{P})$ at every iteration, for every element in the training set, can be costly and so we introduce *stochastic gradient decent* [21]; by selecting a single training point at random the update rule becomes:

$$\mathbf{P}^{i+1} = \mathbf{P}^i - \eta\nabla C_{\boldsymbol{x}^i}(\mathbf{P}^i), \tag{3.34}$$

where,

$$C_{\boldsymbol{x}^i}(\mathbf{P}^i) = \frac{1}{2}||\mathbf{y}^i - \mathbf{a}^{[L-1]}(\boldsymbol{x}^i)||_2^2. \tag{3.35}$$

While this significantly reduces the computational expense, SGD is still able to converge to local minima in the loss space, providing approximation of the best parameters $\mathbf{P}$.

Back-propagation algorithm

In order to use the SGD algorithm introduced in the previous section we must introduce a method of evaluating $\nabla C_{\boldsymbol{x}^i}$, the derivative of the cost function with respect to each of the weights and biases; we achieve this using the *back-propagation algorithm*. We begin by defining a quantity called the error

$$\delta_j^{[l]} = \frac{\partial C_{\boldsymbol{x}^i}}{\partial z_j^{[l]}} \quad \text{for} \quad 0 \leq j \leq N_l - 1 \quad \text{and} \quad 1 \leq l \leq L-1, \tag{3.36}$$

that is the derivative of the loss function at data point i with respect to the z^{th} input to the j^{th} neuron at layer l. Using the definition of the activation function from Eq. (3.25), for $l = L-1$, and differentiating with respect to $z_j^{[L-1]}$ we can shows that,

$$\frac{\partial a_j^{[L-1]}}{\partial z_j^{[L-1]}} = \sigma'(z_j^{[L-1]}). \tag{3.37}$$

Now by considering the derivative of the cost function given in Eq. (3.29), for a fixed training point, with respect to $a_j^{[L-1]}$ we may show,

$$\frac{\partial C_{\boldsymbol{x}^i}}{\partial a_j^{[L-1]}} = \frac{\partial}{\partial a_j^{[L-1]}} \sum_{k=0}^{N_l-1} \frac{1}{2} \left(y_k - a_k^{[L-1]}(x_k) \right)^2 = -(y_j - a_j^{[L-1]}). \tag{3.38}$$

Then, by re-expressing the error using the chain rule, we find

$$\delta_j^{[L-1]} = \frac{\partial C_{\boldsymbol{x}^i}}{\partial z_j^{[l]}} = \frac{\partial C_{\boldsymbol{x}^i}}{\partial a_j^{[L-1]}} \frac{\partial a_j^{[L-1]}}{\partial z_j^{[L-1]}} = (a_j^{[L-1]} - y_j)\sigma'(z_j^{[L-1]}), \tag{3.39}$$

and hence

$$\boldsymbol{\delta}^{[L-1]} = (\mathbf{a}^{[L-1]} - \mathbf{y}) \circ \sigma'(\mathbf{z}^{[L-1]}), \tag{3.40}$$

where $\circ$ represents a component-wise product. Now by making use of the feed-forward algorithm in Eq. (3.28) we can show:

$$\boldsymbol{\delta}^{[l]} = \sigma'(\mathbf{z}^{[l]}) \circ (W^{[l+1]})^T \boldsymbol{\delta}^{[l+1]} \quad \text{for} \quad l = 1, 2, .., L-2. \tag{3.41}$$

Although we do not present the formal proof here[2], it is then possible to show the components of $\nabla C_{\boldsymbol{x}^i}$ may be written in terms of the quantity $\delta^{[l]}$ via

$$\frac{\partial C_{\boldsymbol{x}^i}}{\partial b_j^{[l]}} = \delta_j^{[l]}, \quad \text{and} \quad \frac{\partial C_{\boldsymbol{x}^i}}{\partial W_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]},$$

for $l = 1, 2, .., L-2$.

3.2.4 Neural Network Approximation Power

Having introduced neural networks and the framework necessary to train them to obtain the optimal weights and biases, this final Section will consider what functions a neural network can represent and introduce some heuristic principles for architecture selection. We begin with the universal function approximation theorem, which underpins the theoretical use of neural networks. We then consider the effect of layer width and layer depth on the function space produced.

[2] A formal proof is presented in [20] on page 12.

Universal Function Approximation

One motivating reason for using neural networks for function approximation is that they can represent any continuous function $f(\boldsymbol{x})$; this result, first introduced in the 1980's, is known as the *universal function approximation theorem* [22] and we present it below.

Theorem 3.1: (Universal Function Approximation).
Let $f(\boldsymbol{x})$ be a continuous function with compact support on the domain $\mathbb{R}^n$. Then, for all $\epsilon > 0$ there exists an $M \in \mathbb{N}$ such that

$$|\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})| < \epsilon \tag{3.42}$$

where $\hat{f}(\boldsymbol{x})$ is given by

$$\hat{f}(x) = W^{[2]}\sigma\left(W^{[1]}\boldsymbol{x} + \mathbf{b}^{[1]}\right) \tag{3.43}$$

where $W^{[1]} \in \mathbb{R}^{n\times m}$ and $W^{[2]} \in \mathbb{R}^{m\times 1}$, and all other notation is as presented in section 3.2.2.

While a detailed proof of Theorem 3.1 is beyond the scope of this report (the interested reader can find more detail in [22]), we provide the reader with a sketch proof of the theorem to highlight the intuitive idea.

The definition of the universal function approximator, as presented in Theorem 3.1, uses a single hidden layer. Using a single hidden layer leads to a piece-wise linear function definition as the output; this can be seen by observing that ReLU activation functions are, in a sense, piece-wise linear functions and so a piece-wise linear transformation of a linear combinations of inputs, leads to a piece-wise linear output. Furthermore, the number of linear regions scales approximately linearly with the number of hidden units (i.e., number of linear regions is $\mathcal{O}(N)$, where N is the number of hidden units); this behaviour can be seen in the three example networks plotted in Fig. 3.6. The proof of the universal function approximation follows naturally from this observation: as the number of hidden nodes asymptotically tends to infinity, the length of linear regions tend towards zero, leading to point-wise convergence to any suitably smooth function.

With advances in hardware accelerators, neural networks have seen a trend of increasing layers, in so-called *deep-learning* [23]. A natural question, given that the universal function approximation theorem uses a single layer neural network, is why go deeper? In short, the answer is that deep neural networks can outperform shallow ones in their expressivity [24]: to understand why, we introduce the so-called *saw-tooth* network [25]. Consider the following recursively defined

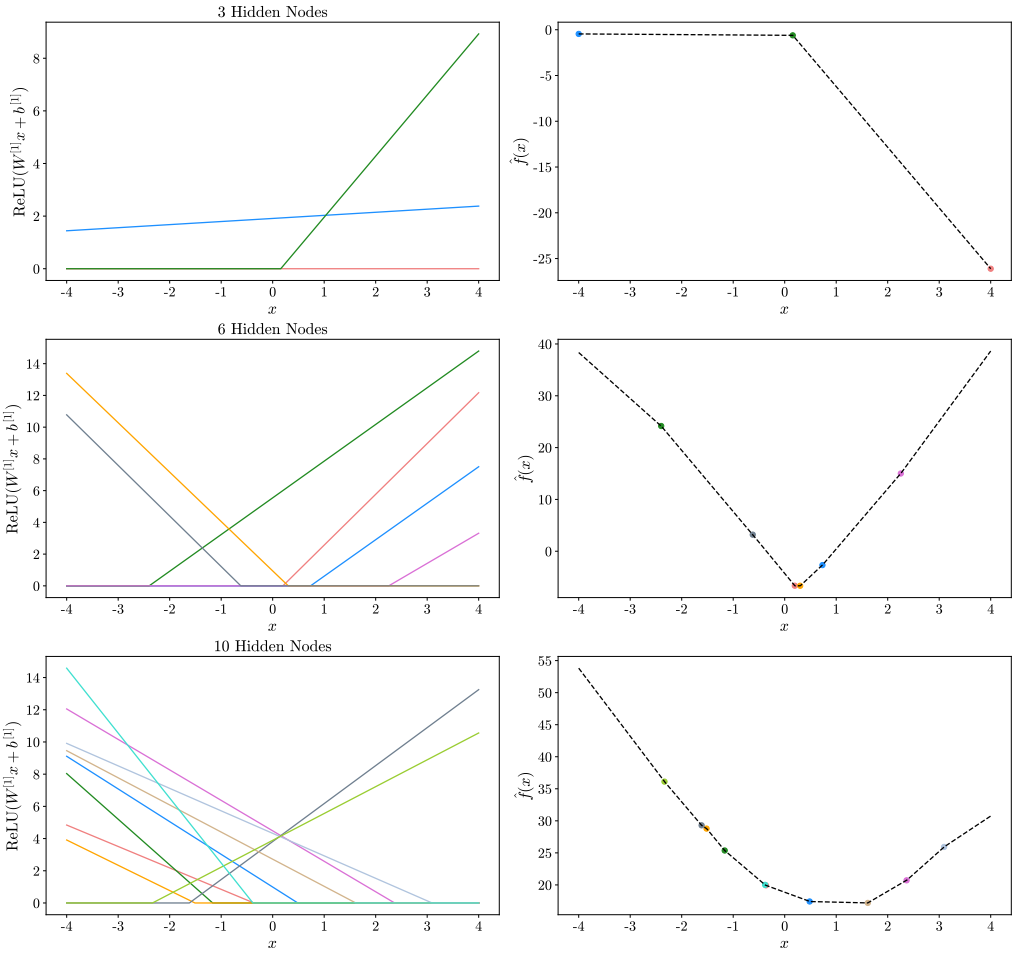


Figure 3.6: Figure shows the output of three neural networks with a single hidden layer but varying hidden nodes. We note that the output function is made of piecewise linear regions.

function

$$f_l(x) = \begin{cases} 2|f_{l-1}(x)| - 2 & l > 0 \\ x & l = 0 \end{cases} \tag{3.44}$$

which can be expressed in-terms of a network with two nodes in each layer l such that

$$f_l(x) = \begin{cases} 2\max(0, f_{l-1}(x)) - 1 + 2\max(0, -f_{l-1}(x)) - 1 & l > 0, \\ x & l = 0. \end{cases} \tag{3.45}$$

We can then consider the effect of changing the number of layers, l, on the function output; the results are plotted in Fig. 3.7. When there is 0 layers, the output of

the network is linear as expected and when there is 1 layer we have two linear regions. Increasing the number of layers l then demonstrates an exponential increase in the number of linear regions, such that the number of linear regions is $\mathcal{O}(2^l)$.

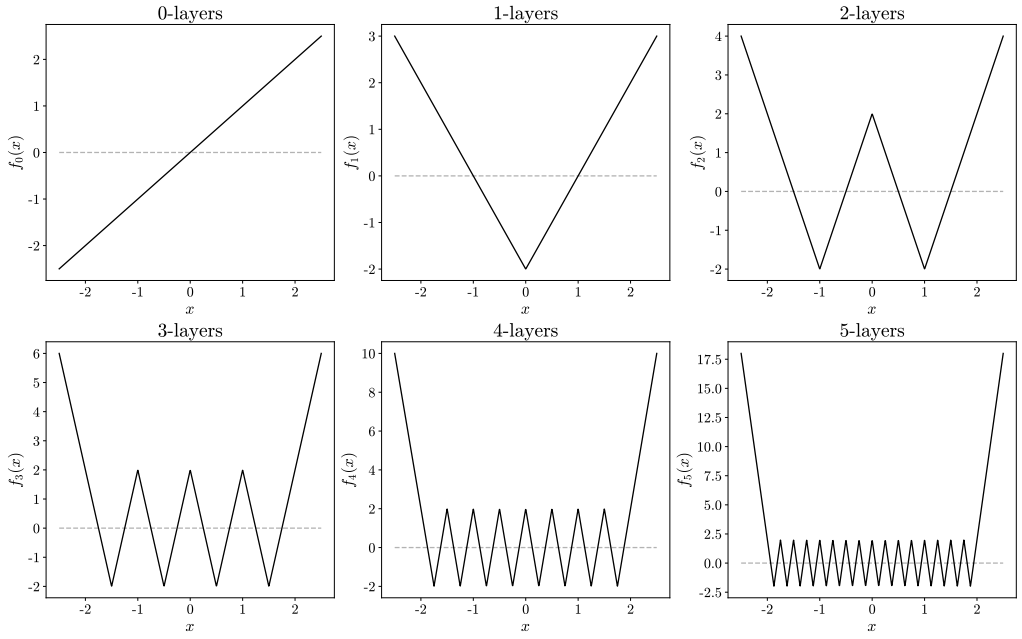


Figure 3.7: Figure shows the output of the sawtooth network, defined by Eq. (3.45), at differing numbers of layers l.

From the underpinning ideas of the universal function approximator, clearly having more linear regions in the output is better for representing higher complexity function with smaller error. While the analysis we present here is simplistic in nature, it provides a key insight into how ReLU network architectures change the resulting approximation space. In particular, fewer network parameters are needed to learn complex function when nodes are stacked into deeper layer compared with shallow and wide networks. We will use this heuristic in Chapter 7, when we implement an MLP to represent basis functions of the MLS surface reconstruction.

Chapter 4

Literature Review

As previously discussed, surface reconstruction from point-sets is mathematically ill-posed: there exists an infinite number of surfaces that can pass through, or near, a given set of scattered data-points. To make progress, the problem is regularised by imposing some form of *prior* as a constraint. Over the past three decades there have been numerous priors suggested in literature like surface smoothness, volume smoothness, and global regularity; see [26] for a comprehensives survey of reconstruction priors.

In this chapter we will concentrate on the surface smoothness class of prior: that is, methods that constrain the optimisation of the reconstructed surface to satisfy a certain level of continuity in the differential properties of the surface on a local scale, while ensuring that the surface is a close fit to the point-set. The prior of surface smoothness can be broken down into two sub-classes: *local smoothness* and *global smoothness*. In Sections 4.1 and 4.2, we provide a comprehensive review of methods that fall into these two categories, respectively. Then, in Section 4.3, we move attention away from hand-crafted smoothness priors to provide a discourse on methods that leverage data-driven techniques. Finally, in Section 4.4 we conclude with a comparative summary of the techniques considered within this chapter. The work in this Chapter is based on, and constitutes an extended version of, the previously conducted literature review [27].

4.1 Local Smoothness

Local smoothness strives for smoothness only in close proximity to each point within the point-set. The local surface smoothness prior tends to smooths out noise in the acquired point-set, but typically cannot handle severe artifacts or highly non-uniform sampling.

4.1.1 Tangent Projection Methods

The pioneering method introduced in [28], laid the groundwork for many subsequent methods that imposed local smoothness priors. The method approximated the surface as a signed distance function $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}$ by projecting each point $\boldsymbol{x} \in \mathbb{R}^n$ onto the tangent plane of the closest point within the point-set. Construction of the tangent plane for each point of the point-set requires that orientated surface normals be also supplied at each point, which are typically approximated using statistical techniques like principle component analysis (PCA) [29]. While this approach is efficient and simple to implement, without any form of smoothing in the approximation scheme the method is sensitive to errors in the surface normal estimations. We illustrate this idea in Fig. 4.1, by reconstructing points sampled from the unit circle but with varying amounts of Gaussian additive noise on the surface normal vector: Gaussian noise with variance of 0.25 results in a saw-tooth effect in the implicit surface, while variance of 0.5 results in a very poor surface reconstruction.

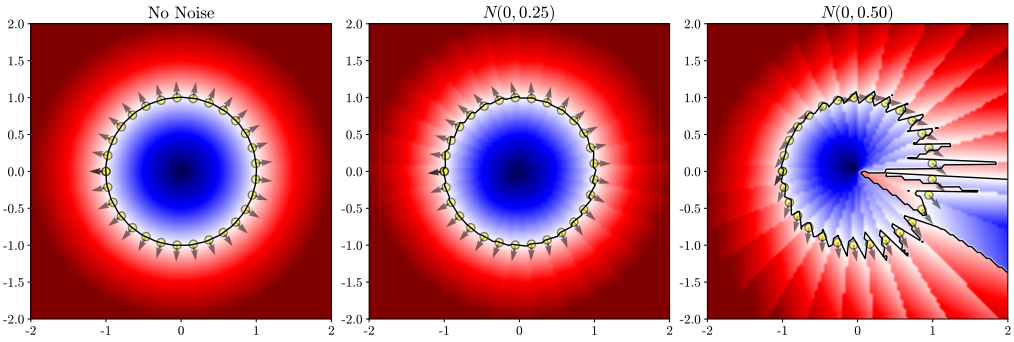


Figure 4.1: Shows the tangent plane surface reconstruction method for the unit circle with varying variances of additive Gaussian noise on the surface normal estimation.

Furthermore, under real scanning condition where the point-set sampling pattern is unlikely to be uniform, the definition of closest point on the manifold surface can also become ill-defined, leading to noisy reconstruction outputs. We illustrate this in Fig. 4.2, where we present the construction of the unit circle with both uniform and non-uniform sampling patterns. Under a uniform sampling distribution of twenty points, the surface generated is relatively smooth, however, under non-uniform sampling the resulting reconstruction contains sharp edges.

Subsequent methods based on local surface smoothness prior have since focused on addressing such issues, mainly by incorporating techniques that help to smooth-out noise in acquired point-sets.

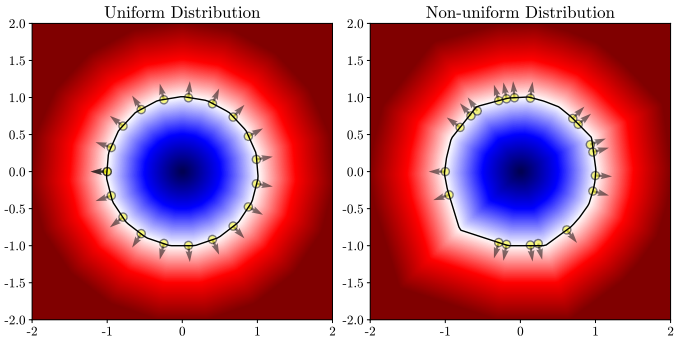


Figure 4.2: Shows the tangent plane surface reconstruction method for the unit circle under different sampling distributions. When the sampling distribution is non-uniform, the resulting distribution is non-optimal.

4.1.2 Moving Least Squares (MLS)

The MLS approach is a prominent method for function approximation of discrete sets of points, which have associated scalar quantities [30]; we provide a detailed introduction to this approximation technique in the previous Chapter (see Section 3.1.4). The initial pioneering work on using the MLS framework for the reconstruction of manifolds was conducted by Levin [31], with Alexa *et al* being the first to explicitly use MLS for applications in the field of computer graphics [32].

The MLS framework is appealing for surface reconstruction of point-sets because under mild conditions it carries provable properties. Given a densely enough sampled point-set, it can be proven that the approximation produced is a good approximation to a signed distance function, and that the reconstructed surface is geometrically and isotropically close to the sampled surface [33]. We note that theoretical guarantees, like those provided in [33], are one of the main attractors for employing MLS for surface reconstruction.

While all MLS approaches to surface reconstruction fundamentally implement the local optimisation problem introduced in Section 3.1.4, there exists many variants in the specific approach. In the next subsections we will explore several variants of MLS. We will begin with the original Levin projection method [31]; while it has become uncommon for this approach to be used in practice for surface reconstruction, it serves as an excellent pedagogy tool for introducing the underpinning idea of MLS for manifold reconstruction. We will then consider several methods that employ MLS for manifold reconstruction, including current state-of-the-art approaches.

Levin Projection Method

The initial approach to using MLS for manifold reconstruction [31] was broken into two stages:

1. First a *query* point is selected around which the approximation will be made, and a local reference plane is fitted to the point-set within a local neighborhood of the *query* point. The reference plane provides a local coordinate system on which the approximation can be constructed. Then the scalar values $s(\mathbf{p})$ for each point are constructed as the smallest distance from the point to the local reference plane. Under this formulation it is easy to see that the scalar values associated with each point represent nothing more than the height field to the reference plane.

2. Secondly, the approximation $\hat{f}(\boldsymbol{x})$ is found via solving the local optimisation problem presented in Def. 3.2, such that the coefficients of a linear combination of polynomial basis are found.

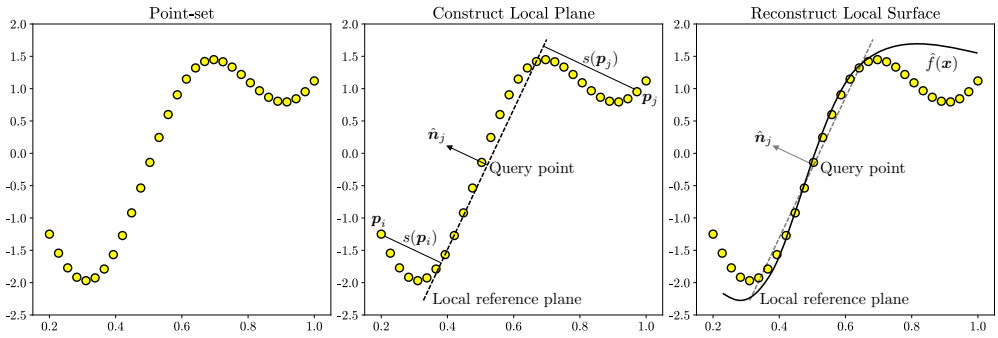


Figure 4.3: Illustrates the Levin projection method for surface reconstruction at a single point. Given a point-set, a local reference plane is constructed around a query point: this defines a height field for all other neighboring points $s(\boldsymbol{p})$. Then MLS optimisation is then carried out to find the coefficients of a parameterised function $\hat{f}(\boldsymbol{x})$.

The Levin projection method for a single point is illustrated in Fig. 4.3. The above procedure is then repeated for different query points, and the surface is formally defined as the subset of all points in $\mathbb{R}^n$ that project onto themselves.

The Levin projection approach has two main limitations: firstly, the construction of a local reference plane becomes problematic when the point cloud is sparse or when the local neighborhood is near sharp features (e.g., a corner). Secondly, this approach is relatively expensive to compute, essentially requiring the solution to two optimisation problems, the plane fitting and the local MLS optimisation, at

each step.

Scalar Field Approximation

It was quickly observed that the Levin projection method could be vastly simplified via omission of the local reference plane fitting [34], and instead associating the point-set with the zero level-set of the scalar field (i.e., $s(\boldsymbol{p}_i) = 0 \;\; \forall \, \boldsymbol{p}_i \in \mathcal{P}$).

Of course, taking this approach requires further constraints be placed on the optimisation problem to avoid the trivial solution ($\hat{f}(\boldsymbol{x}) = 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^n$). If the point-set is oriented, then conditions can be placed on the gradient of the approximated function $\nabla \hat{f}(\mathbf{x})$ such that it is aligned with the surface normal's; this was the approach taken in [35] and [34]. Alternatively, the optimisation can be constrained via the Eikonal equation where the norm of the gradient of the implicit function is constrained to be 1 over all of space. In the context of surface reconstruction, inclusion of the Eikonal term has been used for the MLS optimisation of planes [36] and sphere [37] for surfaces reconstruction.

Implicit MLS

Another variant of the MLS approach is called Implicit MLS (IMLS)[38], which reduces the function approximation space of $\hat{f}(\mathbf{x})$ to tangent planes. Assuming the point-set is oriented, then the signed distance function can be written in closed form as

$$\hat{f}(\mathbf{x}) := \frac{\sum_{\mathbf{p}_i \in \mathcal{P}} \langle \mathbf{x} - \mathbf{p}_i, \mathbf{n}_i \rangle \cdot \omega \left(\|\mathbf{x} - \mathbf{p}_i\| \right)}{\sum_{\mathbf{p}_i \in \mathcal{P}} \omega \left(\|\mathbf{x} - \mathbf{p}_i\| \right)}, \tag{4.1}$$

where $\omega(\cdot)$ is the MLS weighting function and $\langle \cdot, \cdot \rangle$ represents the vector inner-product. IMLS is an attractive approach to surface reconstructions due to the simplicity of its formulation. However, without a global optimization step, it was found that this approach suffers from expanding and shrinking of the surface away from the input points [37]. Furthermore, with the approximation space only constructed via tangent planes, IMLS can struggle with representing high frequency features, tending to produce overly smooth approximations. In Section 6.1.1, we will explore the IMLS approach in more detail and provide a full derivation of Eq. (4.1).

Robust IMLS

One of the limitations of employing the MLS framework for surface reconstruction is that it assumes that points are sampled from a smooth surface, which inherently makes the framework sensitive to outliers and increases the difficulty in recovering high frequency spatial details. In [39], Öztireli introduces the RIMLS method, which uses concepts from the field of robust statistics [40] to overcome these challenges. The method builds upon the IMLS approach but is able to discount outliers by iteratively reweighting points based on their spatial and normal residual errors. The authors showed that their approach could produce far better reconstructions of sharp features, like corners and edges, when compared with the original IMLS formulation; this leads to more faithful surface approximations. While the approach is over a decade old, it still provides a baseline for state-of-the-art in a number of recent papers like [41].

4.2 Global Smoothness

In contrast to the local smoothness prior, global smoothness seeks higher order smoothness, large-scale smoothness, or both. Higher order smoothness refers to the smoothness in the differential properties of the geometry, like the tangent planes or curvature, while large-scale refers to the spatial extend to which smoothness is enforced [1].

4.2.1 Radial Basis Function

One well known method of global scattered data interpolation is radial basis functions. First introduced in [42], radial basis functions for surface reconstruction are constructed via finding a signed field defined via RBFs whose zero level set represents the surface. Globally supported basis function $\phi : \mathbb{R}^+ \to \mathbb{R}$ are used to find an implicit function:

$$\Phi(\boldsymbol{x}) = p(\boldsymbol{x}) + \sum_{\boldsymbol{p}_i \in \mathcal{P}} \lambda_i \phi(\|\boldsymbol{x} - \boldsymbol{p}_i\|), \tag{4.2}$$

where $p(\boldsymbol{x})$ denotes a low-degree polynomial, and the basis functions are centered at the points $\boldsymbol{p}_i$ within the point-set. Some common RBFs include thin-plate splines $\phi(x) = x^2 \log(x)$, Gaussian $\phi(x) = \exp(-cx^2)$, and multiquadric $\phi(x) = \sqrt{x^2 + c^2}$, which are plotted in Fig. 4.4 for $c = 1$.

Like the scalar field approximation methods mentioned in Section 4.1, RBFs are typically used to regress SDFs. Again, to prevent the trivial solution ($\Phi(\boldsymbol{x}) = 0 \ \forall \boldsymbol{x} \in \mathbb{R}$), value constraints are enforced for off surface points, where $\Phi(\boldsymbol{x}_i + \epsilon \boldsymbol{n}_i) = \epsilon$ for some small value ϵ.

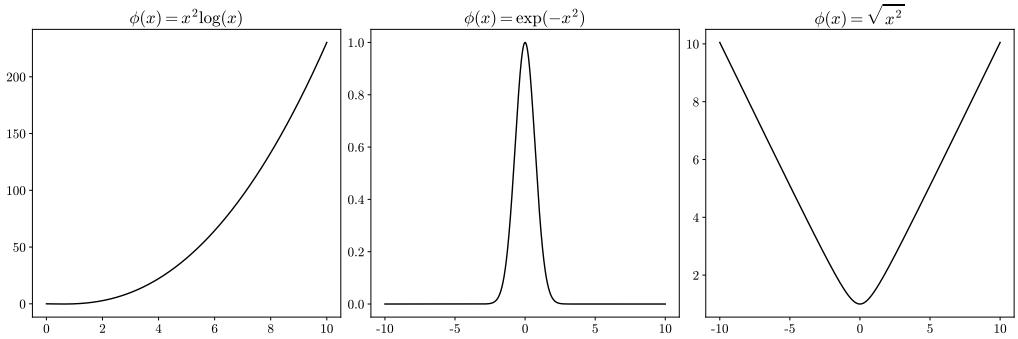


Figure 4.4: Shows common kernel functions used in RBF regression: left is the thin-plate splines $\phi(x) = x^2 \log(x)$, center is the Gaussian $\phi(x) = \exp(-cx^2)$, and right is the multiquadric $\phi(x) = \sqrt{x^2 + c^2}$, with $c = 1$

The real advantage of using the global smoothness prior, like RBF regression, is that the reconstructed surface produced must be globally smooth and hence watertight[1].

4.2.2 Poisson Reconstruction

Until this point, each of the implicit function we have considered has approximated a signed distance function, however, this is not the only choice. There are another class of implicit functions that use the idea of an indicator function to encode the shape boundary; in this work we define an indicator function by the piece-wise definition

$$\Phi(\boldsymbol{x}) = \begin{cases} 1 & x \in \Omega, \\ 0 & x \in \bar{\Omega}, \end{cases} \tag{4.3}$$

where Ω is a set that represents the occupied area of space. Reconstruction of $\Phi(\boldsymbol{x})$, given only a point-set is of course ill-posed; the problem is further constrained by ensuring that the gradient of the implicit function and the surface normals of the point-set align (see Fig. 4.5). This leads to a quadratic error optimisation function

$$\underset{\Phi}{\operatorname{argmin}} \int \|\nabla\Phi(\boldsymbol{x}) - \hat{\boldsymbol{n}}(\boldsymbol{x})\|_2^2 \, d\boldsymbol{x}. \tag{4.4}$$

where $\hat{\boldsymbol{n}}(\boldsymbol{x})$ is the unit surface normal estimate at $\boldsymbol{x}$. The solution to Eq. (4.4) is described by the solution to a Poisson problem, applying calculus of variation, the problem is minimised when:

[1] Watertight is a term used in computer graphics to mean that the surface is closed without holes.

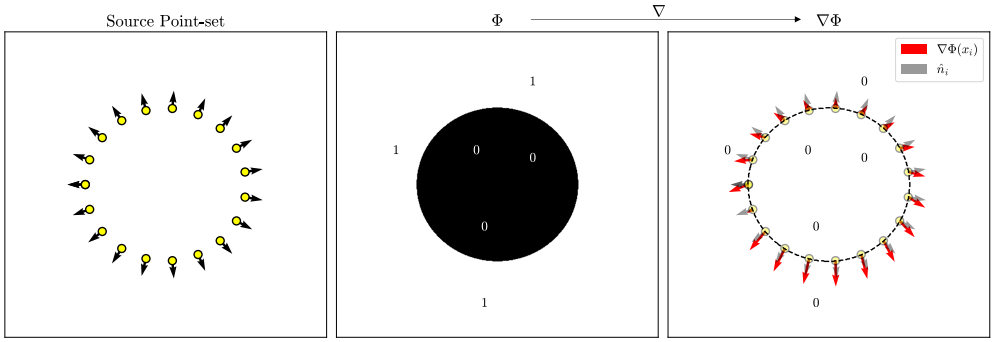


Figure 4.5: Illustration of the indicator function approximation process: Given the input point-set, the gradient of the scalar field Φ can be found and this is used to optimise the coefficients of the field by comparison with the point-set surface normals.

$$\nabla^2\Phi(\boldsymbol{x}) = f(\boldsymbol{x}), \tag{4.5}$$

where $f(\boldsymbol{x}) = \nabla \cdot \hat{\boldsymbol{n}}(\boldsymbol{x})$. Solving the Poisson problem, with the boundary conditions of $\Phi(\boldsymbol{x}) = 0$ on the boundary, then yields the implicit function $\Phi(\boldsymbol{x})$, from which the surface can be recovered by selection of an appropriate iso-value.

The original work framing surface reconstruction as a Poisson problem was presented in [43], where the Poisson problem was solved by transforming into the frequency domain, via a Fourier transform, which resulted in a simple algebraic solution for the Fourier representation of Φ. The problem with this approach is that the application of the Fast Fourier Transform algorithm, used to perform the discrete Fourier transform, requires space be meshed into a uniform grid, which in-turn limits the spatial resolution of the output. Alternative approaches to solve the Poisson problem in the spatial domain using a multilevel, course-to-fine, grid approaches have since been proposed [44] which provide a significant efficiency improvement.

Like the RBF approach, indicator function methods offer a globally smooth solution to the surface, however, as noted in [45] Poisson reconstruction can often lead to over-smoothing of surface; this behaviour is evident in Fig. 4.6.

To overcome this over-smoothing, [45] introduced additional position constraints into the optimisation problem. This later became known as a *screened Poisson reconstruction*, resulting in the optimisation function

$$\underset{\Phi}{\operatorname{argmin}} \int \|\nabla\Phi(\boldsymbol{x}) - \hat{\boldsymbol{n}}(\boldsymbol{x})\|_2^2 \, d\boldsymbol{x} + \lambda \sum_{\mathcal{P}} (\Phi(\boldsymbol{p}_i))^2, \tag{4.6}$$

where λ controls the degree of interpolation of the reconstructed surface, with larger values ensuring a tighter fit to the point-set. The improvement in adding

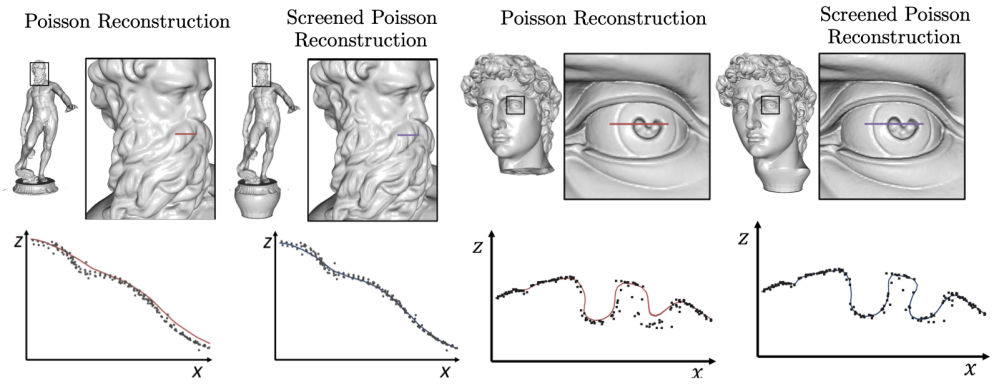


Figure 4.6: Shows the reconstruction of two point cloud models using Poisson and Screened Poisson reconstruction. Notes that Poisson reconstruction tends to over-smooth the surface. Figure adapted from [45].

the second term in Eq. (4.6) can also be seen in Fig. 4.6. However, as with all other interpolatory methods, the wrong choice of parameter λ can lead to over-fitting of the reconstruction surface to the points and so a process of fine-tuning must be implemented to obtain the optimal results.

4.3 Data-driven Approaches

Rather than relying upon hand crafted features to provide regularisation, like local smoothness in the case of MLS, data driven method attempt to learn important geometric and topological features automatically. In this section we will consider a selection of recent papers that consider using machine learning techniques in the context of point-sets surface reconstruction.

4.3.1 Direct SDF Inference

The first work on directly learning signed distance functions for shape representation, DeepSDF, was presented by Park in [46]. The paper proposed using a generative model to produce a continuous signed distance field that represented objects. The generative approach allowed for high quality shape representation, interpolation and completion from partial and noisy 3D input data. DeepSDF significantly outperformed the previous benchmarks on shape representation and completion tasks, and demonstrated the efficacy of using neural network based approaches in the context of 3D geometry. However, the authors only considered synthetic data with fairly simple geometry. Furthermore, the architecture implemented was not invariant to the coordinate systems orientation, requiring the

point clouds be provided in a *canonical* pose. This severely limited this works real world applications as this is typically not the case in real world applications.

A more recent paper [47], also considered using multi-layer perceptrons (MLPs) for directly learning SDFs. Rather than taking a generative approach, Gropp *et al* explicitly regularise their network by including the Eikonal equation, the constraint of the norm of the gradient of the implicit function to be 1, into their loss function. While their method can also include surface normal's, their analysis showed that the inclusion of the Eikonal equation is the most important term for so called *implicit geometric regularization*. The method they introduced for learning high fidelity implicit neural representations of shapes directly from raw data achieved state-of-the-art results. While normal estimations are not needed for this approach, the authors did note that the technique is more sensitivity to noisy normal's when they are used. Furthermore, while the authors did provide some analysis of their network, seeking theoretical bounds, they was only able to do this in the linear case as analysis of higher order terms becomes highly non-trivial. Nevertheless, this work demonstrated how the Eikonal equation can effectively be employed to perform regularisation of geometry in a deep learning context.

4.3.2 Neural MLS Framework

There have also been numerous papers that have considered using deep learning architectures within the IMLS approach. In [41], Liu *et al* present the DeepIMLS approach which allows for the use of IMLS with sparse or noisy point clouds. An auto-encoder architecture is used to predict where MLS points should be within an octree structure [48]. The output of the auto-encoder is then used in the traditional IMLS formulation from section 4.1.2. This method was shown to outperform RILMS (section 4.1.2) in capturing small scale and sharp details.

A different approach to using deep learning with IMLS was discussed in [49]. Here, the authors sought to remove the requirement for an input point-set to have associated surface normals. Instead, the authors propose operating two neural networks in a self-supervised manner, where the loss is coupled between the output SDFs of the network. One of the networks learns the SDF via the IMLS approach and the other uses the gradient of the predicted SDF to help regularise the loss. The authors demonstrated that taking this approach allowed the method to be more robust against noisy and sparse point clouds, while not requiring surface normal vector estimations.

4.3.3 Other Learning Based Approaches

While we concentrated on finding signed distance functions in this report, meshes are another common form of geometry representation; the work presented in Point2Mesh [50] is particularly noteworthy for this report. The aim of the work was to reconstruct a mesh given an input point-set. In this work, the authors introduced the concept of a *self-prior*, which is a network that is able to encapsulate reoccurring geometry from a single shape within the weights of a deep neural network. The method begins by encapsulating the point cloud in a mesh and then iteratively deforms to shrink-wrap the point-set, while preserving geometric features. Key to their approach, a convolutional kernel is optimised across the entire point-set, allowing it to learn global geometric features. This allows the mesh to converge to a desirable solutions without becoming trapped in undesirable local minima. Furthermore, the authors show that this approach works on both synthetic and real world scans for which traditional reconstruction methods typically degrade.

4.4 Summary

In this chapter we have considered an array of different methods for performing surface reconstruction from point-sets. To conclude, we provide a brief summary of the considered methods, highlighting each methods strength and weakness; for convenience this summary is tabulated in Table. 4.1.

Prior	Approach	$\hat{n}$	Strengths	Weakness
Local	Tangent Planes [28]	●	✓ Simple to implement ✓ Highly efficient	✗ Not robust to acquisition noise or normal estimation errors
	MLS: Levin projection [31]	◐	✓ Robust against Gaussian acquisition noise	✗ Requires local reference plane fitting ✗ Typically produces oversmooth reconstruction
	MLS: IMLS [43]	●	✓ Closed form solution to the implicit function ✓ Highly efficient to implement	✗ Typically produces oversmooth reconstruction ✗ Highly sensitive to the Gaussian weighting parameter
	MLS: RIMLS [39]	●	✓ Inherently robust to sharp features ✓ Achieves SOFA for MLS approach to surface reconstruction	✗ Iterative process leads to more computational expense ✗ Increased parameters to tune within the algorithm.
Global	RBF [42]	●	✓ Able to handle non-uniform sampling. ✓ Guaranteed globally watertight reconstruction.	✗ Requires off-surface constraints. ✗ Poor performance against acquisition noise
	Poisson [43]	●	✓ Produces globally watertight reconstruction	✓ ✗ Over-smooths reconstruction/-doesn't pass near point-set ✗ Solution of Poisson problem requires discretization to grid
	Screened Poisson [44]	●	✓ Oversmoothing reduced via a position constrain term. ✓ Produces SOFA reconstruction for indicator based methods.	✗ Has the potential to overfit (interpolate points).
Data Driven	DeepSDF [46]	○	✓ Generative neural network allowed for handling of noise. ✓ SOTA on shape representation and completion tasks.	✗ Requires 3D models be in a canonical pose.
	DeepIMLS [41]	●	✓ Auto-encoder designed to handle acquisition noise. ✓ Outperformed RIMLS on representing small and sharp detail	✗ Significant extra computational expense to compute when compared with IMLS.
	IGR [47]	○	✓ Regularises the problem using the Eikonal equation ✓ Shown to outperformed some classical surface reconstruction.	✗ Limited scope for analysis due to using MLPs ✗ Increased computation expense with training MLP.
	Point2Mesh [50]	○	✓ Demonstrates SOTA in mesh reconstruction from point-sets. ✓ Demonstrated the idea of *self-prior* for shape geometry. ✓ Demonstrated good performance on both synthetic and real-world reconstruction.	✗ Outputs a mesh, which is inherently discrete in nature. ✗ Doesn't always generalise for shapes of arbitrary topology

Key: ● = oriented surface normal's required, ◐ = un-oriented surface normal's required, ○ = no surface normal's required.

Table 4.1: Summary of the literature presented in Chapter 4, grouped by priors used to regularise the reconstruction problem.

Chapter 5

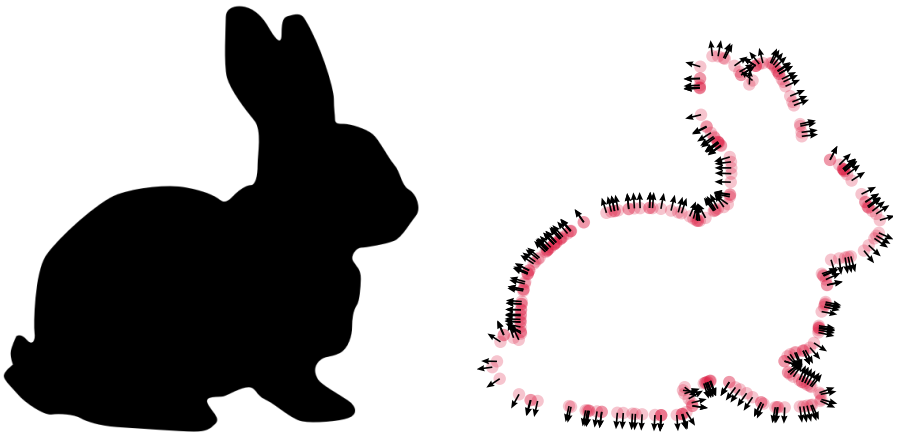
The Shapeset Dataset

High quality data-sets are fundamental to the data-driven paradigm of machine learning. While there are a number of sources of 3D scanned [6, 51, 52] and 3D CAD models [53], there are few sources of 2D point-sets that can be used for algorithm development and testing. In this Chapter we introduce the *shapeset dataset*, a repository of 2D point-sets with surface normal estimations, constructed as part of this project. We begin by presenting the pipeline used to generate the dataset itself, highlighting the computational processes used and the design decisions taken. We then present a subset of the dataset as an example.

5.1 Shapeset Dataset Pipeline

In order to generate 2D point-sets, we first need a source of geometry. As the aim is to be able to easily generate many different types of point-set, with as little human input as possible, it was decided images where the easiest sources of geometry to work with. In particular, we make use of silhouette images where the inside of an object is black and the outside of an object is white.

Images used within this project where taken from [54], where images are released under a creative commons license. An example silhouette image is shown in Fig. 5.1 (a). Then, given this image, our aim is to produce a point-set of the silhouettes boundary with surface normal estimates, like that shown in Fig. 5.1 (b). The overall pipeline for creating 2D point-sets is shown in the flow chart in Fig. 5.2 and in Sections 5.1.1 to 5.1.4, we provide a detailed overview of each stage of the pipeline.



(a) (b)

Figure 5.1: (a) Shows an example silhouette image of a bunny. (b) Shows a point-set extracted from the boundary of the silhouette image, with normal vector estimates.

5.1.1 Image Resizing

The silhouette images used in this project are all of the portable network graphic (`png`) file format. While the `png` format has lossless compression and is able to encode transparent backgrounds, the image they represent have ultimately been rasterized to a fixed pixel array size. As the surface of the geometry will be drawn from the interface between light and dark pixels (described in the following sections), the pixel array size directly impacts the final number of points that can be generated in the point-set. To increase the number of points, an interpolation procedure is applied to increase the overall image size (this idea is illustrated in Fig. 5.3).

To interpolate the pixel array we make use of the mathematical technique of bicubic interpolation, which is commonly used in many image processing tasks. Bicubic interpolation works by considering a neighborhood of 16 pixels around each interpolation pixel, and weight its value according to distance. Of course, with any approximation technique, error is inevitable. In the case of bicubic interpolation, the error manifests as aliasing where the edge inherits a jagged structure (this idea is also demonstrated in Fig. 5.3). While techniques for anti-aliasing exist, we found that they made very little difference to the overall point-set generated; hence, they have not been included in this pipeline.

5.1.2 Edge Detection

Once the image has been resized, the next step is edge detection and extraction. There exists a number of methods of extracting an object boundary, although

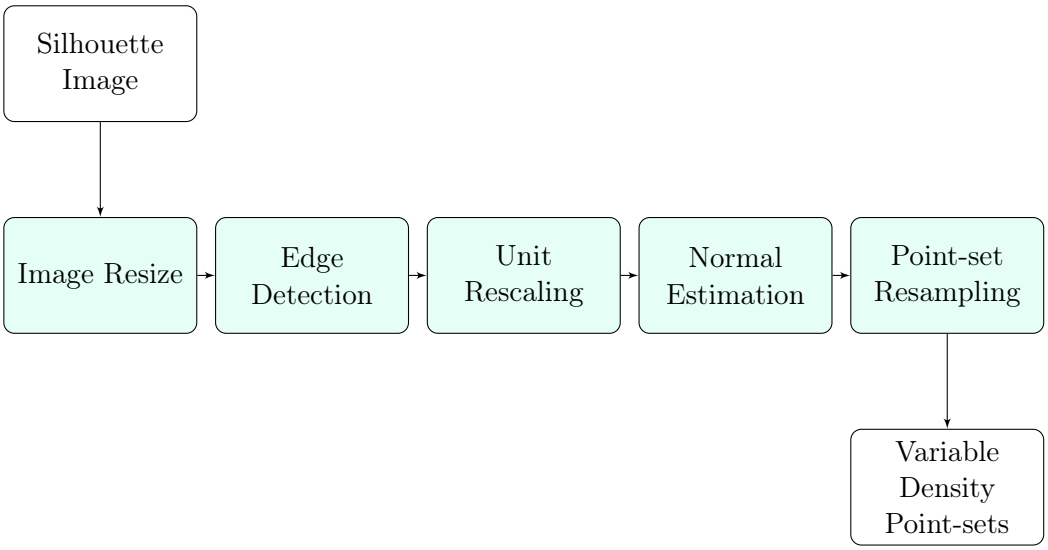


Figure 5.2: Illustration of the 2D point-set construction pipeline. The input to the pipeline is a silhouette image and the output is several point-sets, with accompanying surface normal estimations. The most dense point-set is treated as the ground-truth for any experiments.

most use the gradient of the intensity field to provide the cue for edge detection.

In this work, we use the Canny edge filter [55] because of its simplicity of implementation. In brief, the approach works by first smoothing the image and then calculating the gradient of the images intensity array. The principle is that the image intensity varies most quickly when crossing an object boundary. The algorithm then applies some further criteria, like thresholding the magnitude of the gradients, to suppress spurious responses from the gradient field.

OpenCV provides a convenient method for implementing Canny edge detection, with the output of this process being a binary image, where the black pixels represents the object boundary. The horizontal and vertical location of black pixels are then parsed into an Numpy array.

The point-set, after edge detection, has horizontal and vertical values represented by integers (i.e., the pixel array index of the detected edge). To ensure that length scales are consistent across all generated shapes, the scattered points are rescaled such that $(x, y) \in [0, 1] \times [0, 1]$ for all x and y in the point-set. In practice, this is achieved by dividing every horizontal position by the largest horizontal value, and similarly for the vertical points.

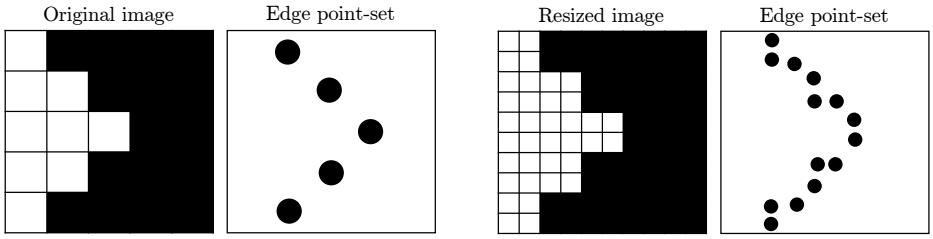


Figure 5.3: Illustrate increasing the pixel array size via interpolation. (a) Shows a section of the original image, with the possible point-set extracted from it. (b) Shows the image after interpolation has been applied, with an increased point density extracted from it.

5.1.3 Surface Normal Estimation

The final stage in creating a ground-truth point-set is to generate an estimate of the oriented surface normal for each point. This estimation process is split into two parts: first we construct tangent plane approximations for each point in the point-set, such that a unit normal can then easily be derived. The second stage is ensuring that the surface normals are oriented: that is ensuring each surface normal points in the direction away from the interior of the object. If a surface normal points into the interior of the object then we call this unoriented.

Tangent Plane Estimation

As discussed in Chapter 2, the normal vector of a point on the boundary of an object is the vector that is perpendicular to the surface tangent plane at that point. Therefore, to construct a surface normal for each point within the point-set, we must begin by constructing a local tangent plane.

For each point, we select k nearest neighbors from the point-set, sampled using closest Euclidean distance; this defines a local region. Then, we construct a linear approximation for the *local region*, such that the best-fit parameters m and c of:

$$y = mx + c, \tag{5.1}$$

are found. Given we now have an estimation of the surface tangent plane, we can construct a vector, $\boldsymbol{\tau}$, that points along the direction of the tangent by sampling two points. For simplicity, we sample the tangent line at $x = 0$ and $x = 1$, such that:

$$\boldsymbol{\tau} = [1, m]^T. \tag{5.2}$$

Then the normal vector is given by: $\boldsymbol{\tau} \cdot \boldsymbol{n} = 0$, which can be trivially satisfied by

letting:

$$\boldsymbol{n} = [-\tau_y, \tau_x]^T, \tag{5.3}$$

where τ_x and τ_y are the x and y component of the vector $\boldsymbol{\tau}$, respectively. This is illustrated in Fig. 5.4. Finally, we normalise the normal vector such that it is of unit length (i.e., $\hat{\boldsymbol{n}} = \frac{\boldsymbol{n}}{|\boldsymbol{n}|}$)

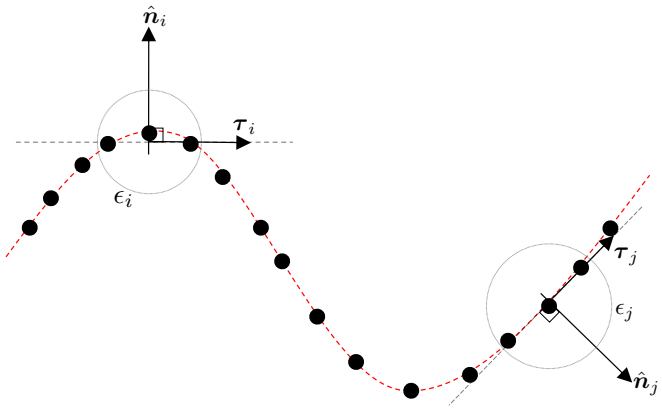


Figure 5.4: Illustration of the process of surface normal estimation for two points in the point-set. Here we use a k nearest neighbor value of 2. Note that the normal vectors are unoriented as they are not consistent in pointing inward or away from the surface.

Orientation Propagation

To ensure the normals are orientated, we select a point at random on the surface. We then march through the point-set selecting the nearest neighbours of the selected point and compute the dot product between the normal vectors. It is expected that adjacent normals should point in similar directions, such that the sign of the dot product is positive. If the sign is negative then the orientation of the normal vector is flipped by scalar multiplication with -1. This process is illustrated in Fig. 5.5.

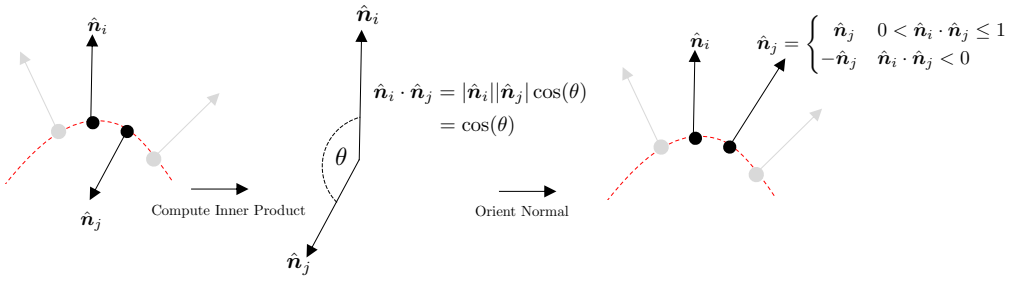


Figure 5.5: Illustration of the surface normal orientation propagation procedure.

5.1.4 Point-set Resampling

Now that we have constructed a point-set, the final stage is to construct subsets of the point-set. This is so that we may regard the most dense point-set as a ground-truth for comparison and testing is later chapters. Points of the point-set where sampled uniformly from the ground-truth point-set at 75%, 50%, 25%, and 10% of the original density. The generated 2D shapeset consists of 20 point-sets. In Fig. 5.6 and Fig. 5.7, we show a selection of 6 shapes at the 5 different sampling densities; note that 100% density is used to refer to the ground-truth geometry in this project.

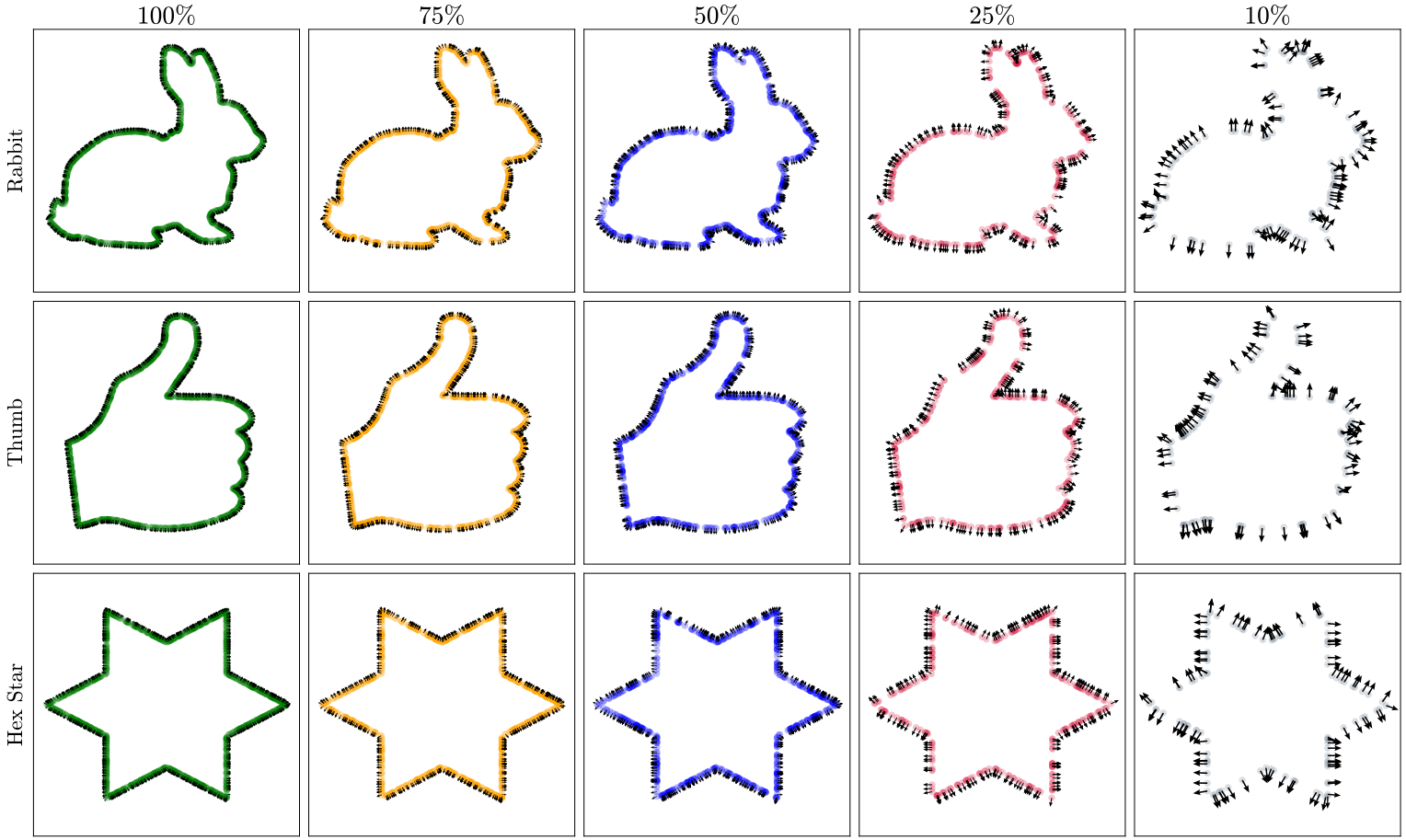


Figure 5.6: Examples of shapes from the `Pointset-Dataset`, with a singular interior region. Each shape shows the individual points and the surface normal estimates at that point. The 100% column heading refers to the ground-truth shape geometry, with the other percentages showing the percentage of points compared to the ground-truth.

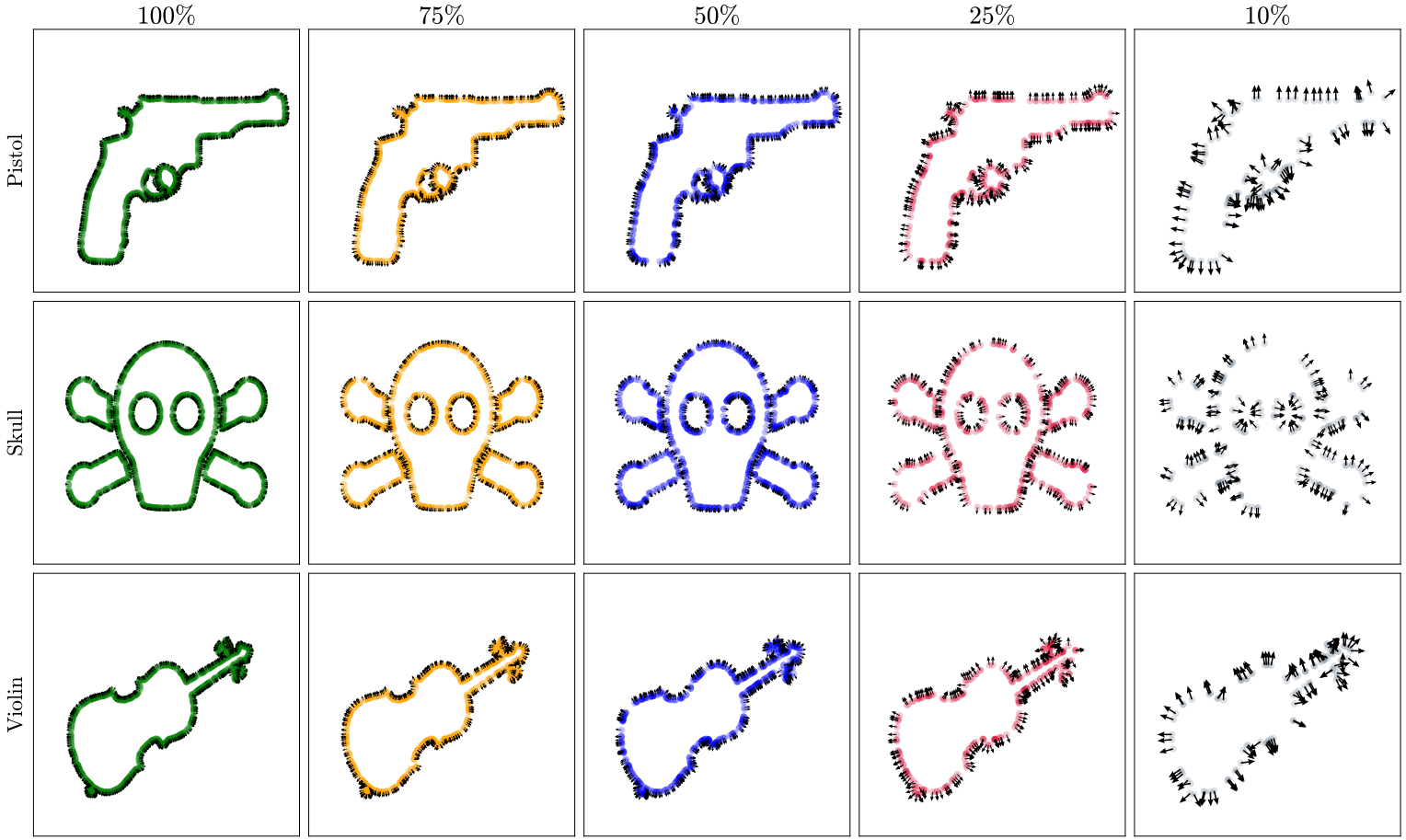


Figure 5.7: Examples of shapes from the `Pointset-Dataset`, with more than one interior region. Each shape shows the individual points and the surface normal estimates at that point. The 100% column heading refers to the ground-truth shape geometry, with the other percentages showing the percentage of points compared to the ground-truth.

Chapter 6

Benchmark Approaches

Before we consider the work conducted on developing a novel MLS surface reconstruction approach in Chapter 7, we first introduce three methods that we will consider as benchmark approaches. In this Chapter we begin with a mathematical derivation of each of the three Benchmark methods. Then we introduce the `PyPointset` Python library, written as part of this project, that provides a Python implementation for each of the benchmark reconstruction methods and utility functions for dealing with point-sets.

6.1 Reconstruction Methods

In this project we implement three methods to act as benchmarks surface reconstruction approaches; the methods selected are:

1. Implicit Moving Least Squares (IMLS),
2. Robust Implicit Moving Least Squares (RIMLS),
3. Implicit Geometry Regularisation (IGR).

In the following Section we provide a detailed derivation of each of the above methods.

6.1.1 Implicit Moving Least Squares

As introduced in Chapter 2, moving least squares (MLS) is a general framework for function approximation of scattered data. The simplest possible case of the MLS framework is where there is a constant basis function, i.e., the approximation space is restricted to:

$$\hat{f}(x) = \alpha_1(x)\phi_1(x) = \alpha_1(x). \tag{6.1}$$

This simple case of MLS is often referred to as Shepard's method in statistics literature [56], but more commonly referred to as implicit moving least squares

(IMLS) in computer graphics literature [33]; this is the convention we adopt in this work. As discussed in Section 4.1.2, while the surface produced by IMLS has a simple closed form solution the reconstructions produced are over-smoothed. Nevertheless, we will regard this method as one of our baseline approaches for comparison. Next, we provide a derivation of the closed form IMLS surface definition and a geometric interpretation of function that is approximated.

Derivation

Begin by letting a set of N scatter data point be contained within a point-set $\mathcal{P}$ such that $\mathcal{P} = \{\boldsymbol{x}_i, \boldsymbol{n}_i\}_{i=1}^{N}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is the position vector of the scattered data point and $\boldsymbol{n}_i \in \mathbb{R}^d$ is the surface normal vector associated with $\boldsymbol{x}_i$. The aim in constructing our approximation is to find a signed distance function. It is therefore important to define the relationship between a query point and the point-set $\mathcal{P}$; this is done via defining the height field.

The Height Field Definition

The aim in defining the height field is to describe a function $u_i(\boldsymbol{x}) = u(\boldsymbol{x}, \boldsymbol{x}_i)$, that approximates a signed distance function for any input $\boldsymbol{x}$. As the approximation space in IMLS is made of a singular constant basis function, the function $u_i(\boldsymbol{x})$ is typically defined as the point on the tangent plane of $\boldsymbol{x}_i$ which is closest to $\boldsymbol{x}$; this is the same technique originally proposed in the tangent plane projection method by Hoppe [28] introduce in Section 4.1.1.

Fig. 6.1 shows the geometry of a query point $\boldsymbol{x}$ being projected onto the tangent plane of $\boldsymbol{x}_i$, where for fixed $\boldsymbol{x}$ we define $d_i = u_i(\boldsymbol{x})$. To be able to use $u_i(\boldsymbol{x})$ within the MLS framework we must be able to express d_i in terms of the known quantities $\boldsymbol{x}$, $\boldsymbol{x}_i$ and $\boldsymbol{n}_i$. By construction $\boldsymbol{x}$ and $\boldsymbol{x}_i$ are both elements of $\mathbb{R}^d$, and so the geodesic distance (the shortest path between the two points) is given by the ℓ_2 vector norm:

$$\xi_i = \|\boldsymbol{x} - \boldsymbol{x}_i\|_2. \tag{6.2}$$

Using Eq. (6.2), the distance d_i can also be expressed as:

$$d_i = \xi_i \cos(\alpha). \tag{6.3}$$

Finally, by noting that surface normals $\boldsymbol{n}_i$ are of unit length (i.e., $\|\boldsymbol{n}_i\| = 1$) by construction, d_i can be rewritten in terms of an inner-product between the vector that points from $\boldsymbol{x}_i$ to $\boldsymbol{x}$ and the unit surface normal $\boldsymbol{n}_i$ by:

$$d_i = \|\boldsymbol{x} - \boldsymbol{x}_i\|_2 \cos(\alpha) = \|\boldsymbol{x} - \boldsymbol{x}_i\|_2 \|\boldsymbol{n}_i\|_2 \cos(\alpha) = \langle \boldsymbol{x} - \boldsymbol{x}_i, \boldsymbol{n}_i \rangle, \tag{6.4}$$

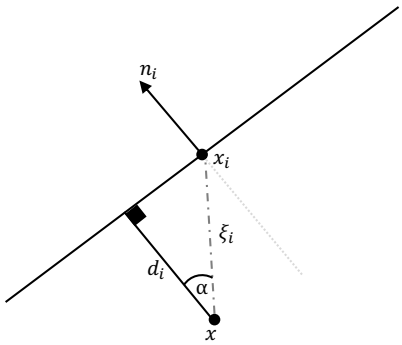


Figure 6.1: Shows the geometry within a local neighborhood of a query point, $\boldsymbol{x}$, being projected onto the closest point of the tangent plane of $\boldsymbol{x}_i$. Here the tangent plane is defined as the set of points $\boldsymbol{q}$ such that $\langle \boldsymbol{x}_i - \boldsymbol{q}, \boldsymbol{n}_i \rangle = 0$, where $\boldsymbol{n}_i$ is the unit surface normal at the point $\boldsymbol{x}_i$. The shortest distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$ is defined by ξ_i and the distance between $\boldsymbol{x}$ and the project of $\boldsymbol{x}$ onto the tangent plane of $\boldsymbol{x}_i$ is denoted as d_i. Finally, the angle between d_i and ξ is denoted by α.

and hence

$$u_i(\boldsymbol{x}) = \langle \boldsymbol{x} - \boldsymbol{x}_i, \boldsymbol{n}_i \rangle. \tag{6.5}$$

The MLS Approximation

Now we have constructed the function to be approximated, we derive the closed form solution for the IMLS surface. Recall that the MLS approximations are local (see Section 3.1.4) and so the approximation coefficients have to be found for each $\boldsymbol{x}$ we wish to evaluate the function at. Using the MLS framework introduced in Chapter 3, we aim to minimise

$$\sum_{\boldsymbol{x}_i \in \mathcal{P}} \Big(\hat{f}(\boldsymbol{x}) - u_i(\boldsymbol{x}_i) \Big)^2 W(\boldsymbol{x}, \boldsymbol{x}_i), \tag{6.6}$$

where $\hat{f}(\boldsymbol{x}) = \alpha_1(\boldsymbol{x})$, $u_i(\boldsymbol{x})$ is given by Eq. (6.5), and $W(\boldsymbol{x}, \boldsymbol{x}_i)$ is a unnormalized Gaussian weight function:

$$W(\boldsymbol{x}, \boldsymbol{x}_i) = \exp\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|_2^2}{\sigma^2} \right). \tag{6.7}$$

Minimising Eq. (6.6) requires that we find the coefficient $\alpha_1(\boldsymbol{x})$ for which Eq. (6.6) is smallest. This can be done analytically by differentiating the expression with respect to $\alpha_1(x)$ and equating the expression to zero,

$$\frac{\partial}{\partial \alpha_1}\left[\sum_{x_i \in \mathcal{P}} \Big(\alpha_1(\boldsymbol{x}) - u_i(\boldsymbol{x})\Big)^2 W(\boldsymbol{x}, \boldsymbol{x}_i)\right] = 0. \tag{6.8}$$

By the linearity of derivatives, the partial derivative in Eq. (6.8) can be exchanged with the summation over $\mathcal{P}$,

$$\sum_{x_i \in \mathcal{P}} \left[\frac{\partial}{\partial \alpha_1}\Big(\alpha_1(\boldsymbol{x}) - u_i(\boldsymbol{x})\Big)^2 W(\boldsymbol{x}, \boldsymbol{x}_i)\right] = 0, \tag{6.9}$$

which can be easily solved to yield:

$$\sum_{x_i \in \mathcal{P}} \Big[2\alpha_1(\boldsymbol{x})\Big(\alpha_1(\boldsymbol{x}) - u_i(\boldsymbol{x})\Big) W(\boldsymbol{x}, \boldsymbol{x}_i)\Big] = 0. \tag{6.10}$$

After trivial rearrangement, Eq. (6.10) can be expressed as

$$\alpha_1(\boldsymbol{x}) \sum_{\boldsymbol{x}_i \in \mathcal{P}} W(\boldsymbol{x}, \boldsymbol{x}_i) = \sum_{\boldsymbol{x}_i \in \mathcal{P}} u_i(\boldsymbol{x}) W(\boldsymbol{x}, \boldsymbol{x}_i), \tag{6.11}$$

and hence, the coefficient $\alpha_1(\boldsymbol{x})$ can be written in closed form as

$$\alpha_1(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{x}_i \in \mathcal{P}} u_i(\boldsymbol{x}) W(\boldsymbol{x}, \boldsymbol{x}_i)}{\sum_{\boldsymbol{x}_i \in \mathcal{P}} W(\boldsymbol{x}, \boldsymbol{x}_i)}. \tag{6.12}$$

Finally, substituting $u_i(\boldsymbol{x})$ of Eq. (6.5) into Eq. (6.12) and $\alpha_1(\boldsymbol{x})$ of Eq. (6.12) into Eq. (6.1), we yield the Kolluri [33] definition of the IMLS surface:

$$\hat{f}(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{x}_i \in \mathcal{P}} \langle \boldsymbol{x} - \boldsymbol{x}_i, \boldsymbol{n}_i \rangle W(\boldsymbol{x}, \boldsymbol{x}_i)}{\sum_{\boldsymbol{x}_i \in \mathcal{P}} W(\boldsymbol{x}, \boldsymbol{x}_i)}. \tag{6.13}$$

6.1.2 Robust Implicit Moving Least Squares

While the IMLS approach, derived in the previous section, offers a simple closed-form function to approximate a singed distance function, fitting tangent planes and blending them together with Gaussian weighting functions does not allow for sharp features within a scanned objects geometry to be modelled.

Robust implicit moving least squares (RIMLS) [57] improved upon the standard IMLS approach by producing an approximation that is robust against outliers in both spatial position and surface normals noise. While robustness against spatial position is useful for dealing with noise from acquisition, robustness against

surface normals is key to producing sharp feature reconstructions. When sharp features are present in geometry, the differential properties of the surface become discontinuous. Then, viewing the different discontinuous regions as *patches*, we can see that normals belonging to the surface in different patches typically point in vastly different directions (see Fig. 6.2). Therefore, the surface normals from different patches should be regarded as the outliers if sharp features are to be preserved.

Using ideas from the field of robust statistics, Öztireli [39] showed how MLS could be reframed as a local kernel regression (LKR) problem and then showed how it could be solved iteratively using re-weighted least squares (IRLS). In the following section we present the salient points needed from robust statistics and derive the implicit surface definition. We then provide an overview of the algorithmic implementation of the approach.

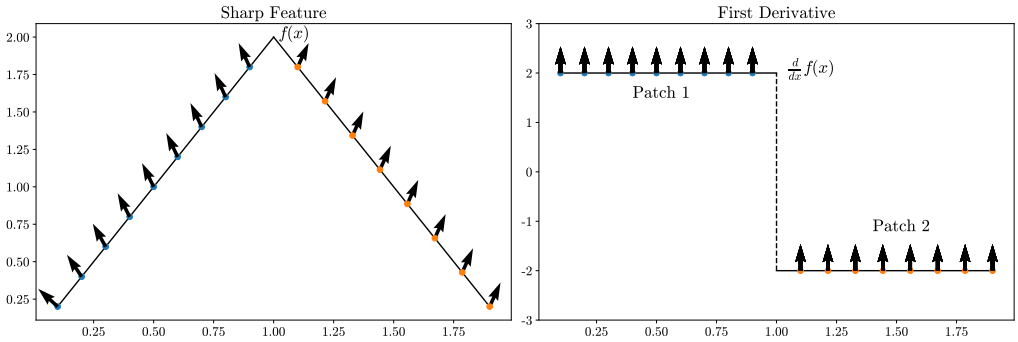


Figure 6.2: Shows the surface normal vectors near a sharp feature in a shapes geometry. The surface normal can be regarded as belonging to different patches on the shapes surface and hence points in the first plane are outliers in the second, and vice-versa.

Robust Statistics

RIMLS makes use of iteratively re-weighted least squares (IRLS) to solve its optimisation problem. While a full introduction to IRLS is beyond the scope of this report (the interested reader should consult [58]), we highlight the necessary parts to derive the RIMLS surface definition.

An *M-Estimator* is an approach that generalises the statistical techniques of maximum likelihood estimation, and is usually used as an alternative to least squares method when the data has outliers, extreme observations, or does not follow a normal distribution [59]. The fundamental idea behind RIMLS is to approach the minimisation of the MLS objective function from the perspective

of finding an *M-Estimator* for it. The aim is to minimise the objective function

$$\arg\min_{\boldsymbol{\alpha}(\boldsymbol{x})} \sum_i \rho\left(\hat{f}(\boldsymbol{x}) - y_i\right) w(\boldsymbol{x}, \boldsymbol{x}_i) \tag{6.14}$$

where the vector of coefficient $\boldsymbol{\alpha}(\boldsymbol{x}) = [\alpha_1(\boldsymbol{x}), \alpha_2(\boldsymbol{x}), ..., \alpha_n(\boldsymbol{x})]^T$ defines a linear combination of basis functions $\hat{f}(\boldsymbol{x}) = \sum_i \alpha_i(\boldsymbol{x})\phi_i(\boldsymbol{x})$ and $\rho(\cdot)$ is some arbitrary function. Obviously, if $\rho(x) = x^2$ then we recover precisely the MLS objective function, and so the choice of $\rho(\cdot)$ is important in generating a robust estimator. If we choose $\rho(\cdot)$ such that it is differentiable and we define,

$$\eta(x) = \frac{1}{x}\frac{d\rho}{dx}, \tag{6.15}$$

then the minimisation problem defined in Eq. (6.14) can be solved via iteratively re-weighted least squares (IRLS) defined by:

$$\boldsymbol{\alpha}^k = \arg\min_{\boldsymbol{\alpha}} \sum \left(\hat{f}(\boldsymbol{x}) - y_i\right)^2 w(\boldsymbol{x}, \boldsymbol{x}_i)\eta(r_i^{k-1}), \tag{6.16}$$

where $r_i^{k-1} = \hat{f}_{\boldsymbol{\alpha}^{k-1}}(\boldsymbol{x}_i) - y_i$ is the i^{th} residual at the $k-1$ iteration using the coefficients $\boldsymbol{\alpha}^{k-1}$ in the functional $\hat{f}(\boldsymbol{x})$. The key to making this optimisation problem robust to outliers is in the choice of function $\rho(\cdot)$. Ideally, $\rho(\cdot)$ should grow slowly such that the weight function $\eta(x)$ tends to zero as x tends to infinity.

In the RIMLS approach, the authors choose the $\rho(x)$ to be the Welsch function defined by,

$$\rho(x) = \frac{\sigma_r^2}{2}\left(1 - \exp\left(-\frac{x^2}{\sigma_r^2}\right)\right), \tag{6.17}$$

which leads to a weight function defined a un-normalised Gaussian,

$$\eta(x) = \frac{1}{x}\frac{\partial\rho}{\partial x} = \exp\left(-\frac{x^2}{\sigma_r^2}\right). \tag{6.18}$$

In Fig. 6.3 we provide a comparison of $\rho(x) = x^2$ (i.e., the OLS criterion) and $\rho(x)$ equal to the Welsch weight function. As can be seen, the weight function $\eta(\cdot)$ for the OLS case is constant, but in the case of the Welsch function the residuals are weighted according to a Gaussian distribution. This provides a mechanism to robustly discount the largest residuals in the minimisation problem.

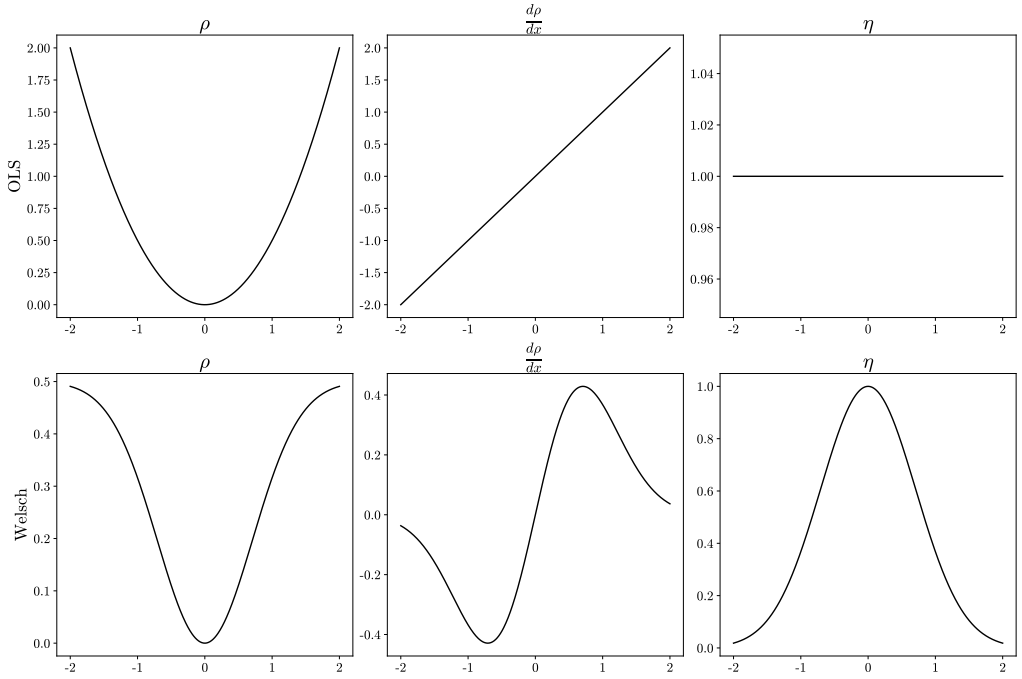


Figure 6.3: Shows the objective function, ρ, derivative of the objective function, $\frac{d\rho}{dx}$, and the weighting term, η, for both OLS and Welsch objective functions.

Implicit Function Derivations

Following almost directly the same derivation as IMLS in Section 6.1.1, we use a functional with constant basis function,

$$\hat{f}(\boldsymbol{x}) = \alpha_1(\boldsymbol{x})\phi(\boldsymbol{x}) = \alpha_1(\boldsymbol{x}), \tag{6.19}$$

and we seek to regress a signed distance function which we approximate using $y_i = \langle \boldsymbol{x} - \boldsymbol{x}_i, \boldsymbol{n}_i \rangle$ (see Section 6.1.1 for a derivation of this). Using the robust statistics introduced in the previous section, the robust IRLS minimisation then becomes

$$f^k(\boldsymbol{x}) = \arg\min_{\alpha_1} \sum \left(\alpha_1 + \langle \boldsymbol{x} - \boldsymbol{x}_i, \boldsymbol{n}_i \rangle\right)^2 w(\boldsymbol{x}, \boldsymbol{x}_i)\eta\left(r_i^{k-1}\right) \tag{6.20}$$

where $r_i^{k-1} = f^{k-1}(\boldsymbol{x}) - \langle \boldsymbol{x} - \boldsymbol{x}_i, \boldsymbol{n}_i \rangle$ is the residual between the tangent plane projection and the $k-1$ approximation to the signed distance function at $\boldsymbol{x}$. While this function definition already provides a more robust approximation in the face of spatial outliers, it does not account for outliers in the surface normal. Defining the normal vector residual as,

$$\Delta n_i^k = \|\nabla f^k(\boldsymbol{x}) - \boldsymbol{n}_i\|_2, \tag{6.21}$$

we can define a new weight function on the surface normals μ as

$$\mu(\Delta n_i^k) = \exp\left(-\left(\frac{\Delta n_i^k}{\sigma_n}\right)^2\right). \tag{6.22}$$

Integrating this final weight term into the minimisation problem in Eq. (6.20) and solving for the parameter α_1, the iterative definition for the final RIMLS surface is given by

$$f^k(\boldsymbol{x}) = \frac{\sum\langle \boldsymbol{x} - \boldsymbol{x}_i, \boldsymbol{n}_i\rangle w(\boldsymbol{x}, \boldsymbol{x}_i)\eta\left(r_i^{k-1}\right)\mu\left(\Delta n_i^k\right)}{\sum w(\boldsymbol{x}, \boldsymbol{x}_i)\eta\left(r_i^{k-1}\right)\mu\left(\Delta n_i^k\right)}. \tag{6.23}$$

Algorithmic Implementation

To implement RIMLS, there are three elements of the algorithm that must first be considered. First is the initialisation of the re-weighting terms $\eta(\cdot)$ and $\mu(\cdot)$. As discussed in [57], it is usual to initialise an IRLS method with another *robust* estimator like the median, however, as the continuity of the solution is important the authors of [57] opted to uses a least squares criterion as the initialiser. This has the advantage of being both simple to implement, the weights at the first iteration are set to 1 (i.e., $\eta\left(r_i^0\right) = \mu\left(\Delta n_i^0\right) = 1$), and also ensure that each iteration produces a continuous reconstruction.

The second element that must be addressed is the computation of the gradient of the scalar field in Eq. (6.21). As the definition of $\nabla f^k(x)$ is recursive, computation of the exact derivatives yields a complicated and computationally expensive expression. However, the gradient of the field can be approximated by assuming that the reweighing terms are constant; this leads to an approximation where the error is maximal when the refitting weights vary quickest (i.e., near sharp edges). We begin by defining:

$$\xi_i^{k-1} = \eta\left(r_i^{k-1}\right)\mu\left(\Delta n_i^{k-1}\right), \tag{6.24}$$

such that ξ_i^{k-1} is considered constant with respect to $\boldsymbol{x}$, then the gradient of the

scalar field is given by

$$\begin{aligned}\nabla f^k(\boldsymbol{x}) &\approx \nabla\left(\frac{\sum\langle \boldsymbol{x}-\boldsymbol{x}_i,\boldsymbol{n}_i\rangle w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}}{\sum w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}}\right),\\ &\approx \frac{\sum \boldsymbol{n}_i(w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1})^2+\sum\langle \boldsymbol{x}-\boldsymbol{x}_i,\boldsymbol{n}_i\rangle\nabla w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}\left(w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}\right)}{\left(\sum w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}\right)^2},\\ &\quad-\frac{\sum\langle \boldsymbol{x}-\boldsymbol{x}_i,\boldsymbol{n}_i\rangle w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}\nabla w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}}{\left(\sum w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}\right)^2},\\ &\approx \frac{\sum \boldsymbol{n}_i w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}+\sum\nabla w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}\left(\langle \boldsymbol{x}-\boldsymbol{x}_i,\boldsymbol{n}_i\rangle-f^k(\boldsymbol{x})\right)}{\sum w(\boldsymbol{x},\boldsymbol{x}_i)\xi_i^{k-1}}.\end{aligned} \tag{6.25}$$

The gradient of the Gaussian weighting term in Eq. (6.25) is then given by

$$\nabla w(\boldsymbol{x},\boldsymbol{x}_i)=-\frac{2|\boldsymbol{x}-\boldsymbol{x}_i|}{\sigma^2}w(\boldsymbol{x},\boldsymbol{x}_i). \tag{6.26}$$

The final element that must be addressed is defining a termination criterion for the IRLS minimisation. There exists two options for terminating: 1) Termination can be defined in terms of convergence

$$\max_i\left|\frac{\eta\left(r_i^k\right)\mu\left(\Delta n_i^k\right)}{\sum_j\eta\left(r_j^k\right)\mu\left(\Delta n_j^k\right)}-\frac{\eta\left(r_i^{k-1}\right)\mu\left(\Delta n_i^{k-1}\right)}{\sum_j\eta\left(r_j^{k-1}\right)\mu\left(\Delta n_j^{k-1}\right)}\right|<t \tag{6.27}$$

where t is a user defined threshold. 2) Termination can be defined in terms of a maximum number of IRLS iterations M, which is also user defined.

6.1.3 Implicit Geometry Regularisation

The final comparison method we present is the implicit geometric regularisation method [47]. In this method, a neural network is used directly to encode the signed distance function of the implicit surface, which is referred to as a *neural implicit representation*. While the IGR method cannot be derived from first principles, we will introduce the loss function and seek to explain how each term contributes to the surface reconstruction method.

More precisely, the IGR method aims to find the optimal weights and biases $\boldsymbol{\theta}$ of a fully connected MLP, which we denote $\hat{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$ where $f_{\boldsymbol{\theta}}:\mathbb{R}^2\to\mathbb{R}$, such that $\hat{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$ approximates a signed distance function. The approach accepts either just a point-set, $\boldsymbol{x}_i\in\mathcal{P}$, or a point-set with surface normals, $(\boldsymbol{x}_i,\boldsymbol{n}_i)\in\mathcal{P}$. The MLP is trained with respect to the following loss function,

$$\mathcal{L}(\boldsymbol{\theta})=\ell(\boldsymbol{\theta})+\lambda\mathbb{E}_{\boldsymbol{x}}\Big[\left(||\nabla f_{\boldsymbol{\theta}}(x)||_2-1\right)^2\Big] \tag{6.28}$$

where $\lambda > 0$ is a hyper-parameter, $||\cdot||_2$ is the Euclidean 2 norm, and

$$\ell(\boldsymbol{\theta}) = \frac{1}{|\mathcal{P}|} \sum_{i \in \mathcal{P}} \left(\| f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \|_2 + \tau \| \nabla_{\boldsymbol{x}} f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{n}_i \|_2 \right). \tag{6.29}$$

If $\mathcal{P}$ does not contain surface normals, then $\tau = 0$, else it is tuned to a value in the range $(0, 1]$.

We now seek to understand this loss function by considering each terms affect on the implicit function obtained. The first term of Eq. (6.29) constrains the implicit function at the point-set positions, which has the effect of forcing $\hat{f}(\boldsymbol{x}) \to 0$ for all $\boldsymbol{x} \in \mathcal{P}$. This is one of the requirements of a signed distance function. The second term of Eq. (6.29) constrains the gradient of the field at the point-set to point in the direction of the surface normal vector, a requirement that the implicit surface must satisfy following from the underpinning differential geometry. Finally, the second term of Eq. (6.28), is known as the Eikonal term. It is possible to show that if an implicit function $f(x)$ is a signed distance function then $f(x)$ satisfies the Eikonal equation

$$\|\nabla f(\boldsymbol{x})\| = 1 \ \forall \boldsymbol{x} \in \mathbb{R}^n. \tag{6.30}$$

While we do not provide a proof here, the interested reader should consult [60]. Hence, minimising the error in the Eikonal term over samples of $\mathbb{R}^2$ provides a regulariser that encourages the implicit function represented by the MLP to produce a signed distance function over all of space.

6.2 Point-set Reconstruction Library

Now we have introduced the three benchmark methods implemented within this project, we introduce the `PyPointset` library. The library is lightweight and has minimal dependencies, only requiring `Numpy` for numerical computations, `TensorFlow` for hardware accelerated neural network training, and `Matplotlib` for visualisation. In the following sections we present further details about the library, provide a minimal setup code example for each method, and show some example reconstructed surfaces.

6.2.1 The Point-set Class

The library is implemented such that the point-set (either just positions or positions and surface normals) are encapsulated within a point-set object. Once instantiated, the positions and surface normals can be obtained using the `plc_object.position` and `plc_object.normals` methods, respectively. Then each reconstruction method is contained within its own class and the point-set object is passed to the reconstruction methods as necessary. The following code

instantiates the built-in example star point-set and then prints both the position vectors and normal vectors.

```
import reconstruct.pointcloud_utils as plc_utils

plc = plc_utils.pointcloud() #load example star point-set

print(plc.position) # print the point-set position vectors
print(plc.normals)  # print the point-sets normal vectors
                    # both return np arr of shape (N, 2), where
                    ↪  N is the number of points.
```

6.2.2 IMLS Implementation

As the IMLS implicit surface has a closed form solution, Eq. (6.13) of Section 6.1.1, its computational implementation is trivial amounting to nothing more than a function that carries out the algebraic operations on vectors. For efficiency, the function is parallelised by vectorisation of operations like the inner product and the weighting computation, using the numerical computing package `Numpy`.

IMLS reconstruction using the `PyPointset` library can be carried out in 6 lines of Python code:

```
import reconstruct.pointcloud_utils as plc_utils
import reconstruct.IMLS as IMLS

plc = plc_utils.pointcloud() # loads example star point-set.
imls = IMLS.IMLS(plc, sigma = 0.5) # defines the weight term.
imls.construct_approximation(200) # evaluate on 200x200 grid
imls.visualise_reconstruction() # extract iso-contour
```

The reconstruction obtained via running the above code is shown in Fig. 6.4, with the signed distance function plotted in Fig. 6.4 (a) and the reconstructed surface extracted from the 0 iso-contour plotted in Fig. 6.4 (b).

RIMLS Implementation

Although the derivation of the RIMLS implicit function is some-what involved, its actual computational implementation is fairly straightforward. Our implementation largely follows the pseudo-code for the RIMLS approach presented by Öztireli in [57]. The method is built using `Numpy` and also implements vectorisation of algebraic operations for efficiency.

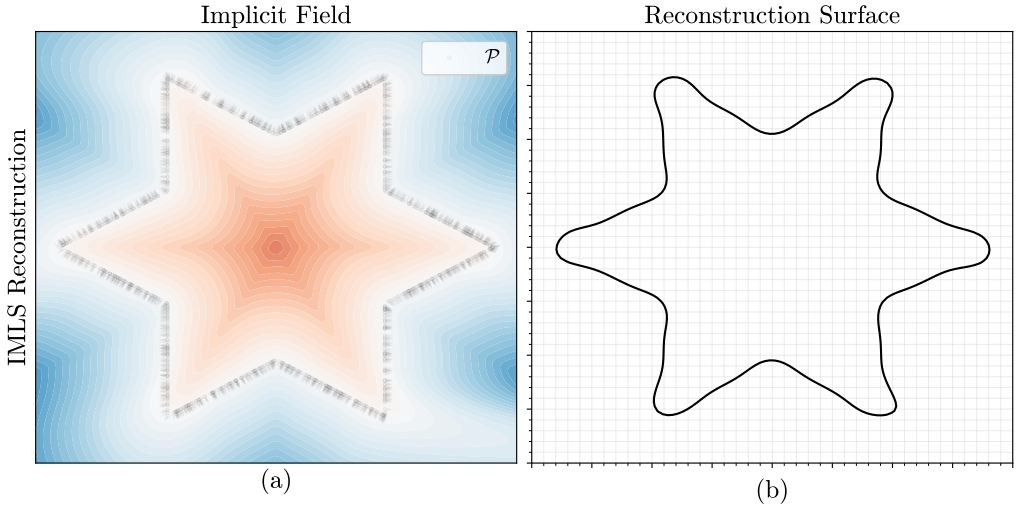


Figure 6.4: Shows the reconstruction output produced when using the IMLS module of the `PyPointset` library. (a) Shows a visualisation of the implicit field where orange represents negative values and blue represents positive values. (b) Shows the extracted surface from the 0^{th} level-set of the implicit field.

RIMLS reconstruction using the `PyPointset` library can also be carried out in 6 lines of Python code:

```
import reconstruct.pointcloud_utils as plc_utils
import reconstruct.RIMLS as RIMLS

plc = plc_utils.pointcloud() # loads example star point-set.
rimls = RIMLS.RIMLS(plc, [0.001, 0.06, 0.02]) #define alg
↪  params
rimls.construct_approximation(100) # evaluate on 200x200 grid
rimls.visualise_reconstruction() # extract iso-contour
```

The list of parameters passed as arguments when instantiating the RIMLS class refers to the Gaussian reweighting variances σ^2, σ_r^2, and σ_n^2, respectively. The above code yields the reconstruction shown in Fig. 6.5. We note that sharp features of the stars geometry are preserved much better in this reconstruction but that the algorithm doesn't not produce a water-tight reconstruction in the area where the the point-set has a less dense sampling (bottom right).

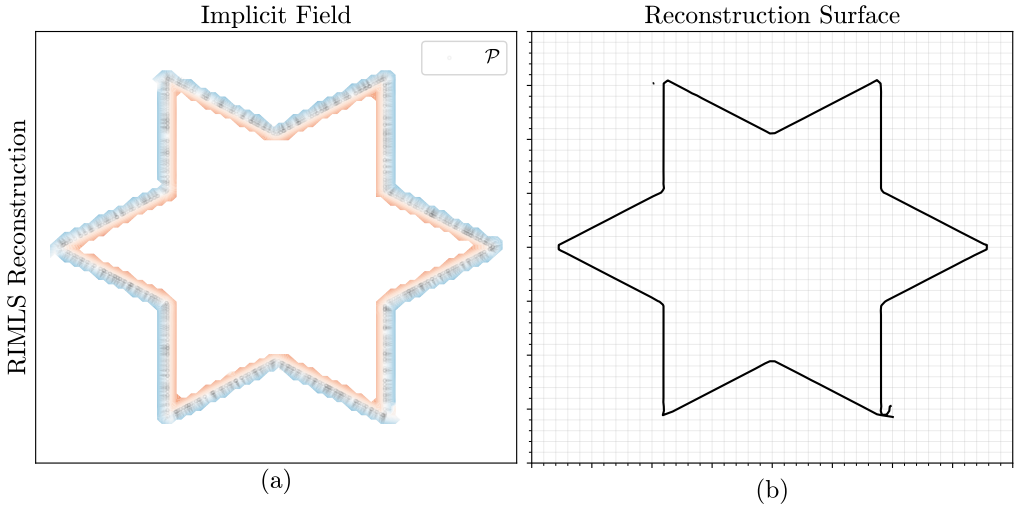


Figure 6.5: Shows the reconstruction output produced when using the RIMLS module of the `PyPointset` library. (a) Shows a visualisation of the implicit field where orange represents negative values and blue represents positive values. Note that the white area of the implicit field away from the surface defines areas where the value of the implicit field has evaluated to `nan`. (b) Shows the extracted surface from the 0^{th} level-set of the implicit field.

IGR Implementation

The final method, IGR, differs from the previous two methods in that the SDF does not have a closed or iterative closed form solution. Instead the SDF is represented in the weights and baises of an MLP after the network has been trained. The MLP was implemented in `Tensorflow` deep learning framework as this provides access to hardware acceleration, improving computation efficiency.

Like IMLS and RIMLS, IGR reconstruction using the `PyPointset` library can also be carried out in 6 lines of Python code:

```
import reconstruct.pointcloud_utils as plc_utils
import reconstruct.IGR as IGR

plc = plc_utils.pointcloud() #load example star point-set
igr = IGR.IGR(plc)
igr.train(5000) # train for 5000 epochs
igr.visualise_reconstruction() # extract iso-contour
```

The default MLP architecture for the IGR method is a feedforward MLP with 4 hidden layer, each with 80 neurons. The weights and biases are initialised via sampling from an $\mathcal{N}(0, 0.01)$ Gaussian distribution and the network uses the

stochastic gradient decent algorithm (see Section 3.2.3) with a learning rate of 10^{-3}.

The reconstruction obtained via running the above code is shown in Fig. 6.6, with the signed distance function plotted in Fig. 6.6 (a) and the reconstructed surface extracted from the 0 iso-contour plotted in Fig. 6.6 (b).

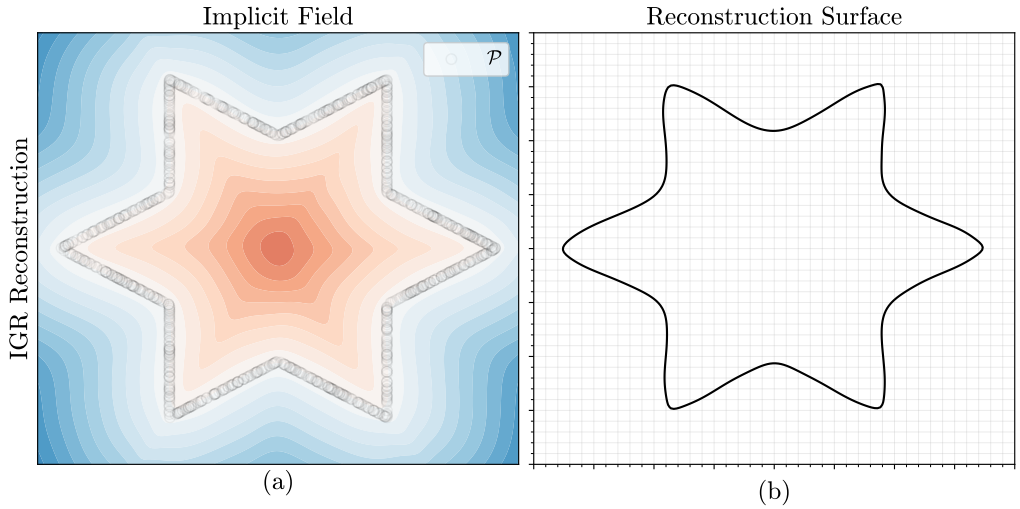


Figure 6.6: Shows the reconstruction output produced when using the IGR module of the `PyPointset` library. (a) Shows a visualisation of the implicit field where orange represents negative values and blue represents positive values. (b) Shows the extracted surface from the 0^{th} level-set of the implicit field.

Chapter 7

A Novel MLS Approach

As previously discussed, the MLS framework for function approximation and surface reconstruction acts on an inherently local scale: the approximation at each point is constructed via consideration of only a local neighborhood of scattered data-points. While the local smoothness prior has the advantage of being able to generalise to many classes of geometry, acquisition devices, and point-set spatial scales, the appropriate handling of either sharp features within the geometry or acquisition noise is typically not possible without further constraining the problem.

In this Chapter we introduce the work conducted on developing a novel approach to surface reconstruction using the moving least squares (MLS) framework. As we have discussed previously, MLPs can theoretically approximate any function. In this project, we seek to use this principle to develop an approach where *optimal* basis functions for the MLS reconstruction are learned. The aim is that learning optimal basis functions, *globally* across shape geometry, will allow us to represent different geometric features, for example sharp features, better on a *local* scale; this general principle is similar to the concept of *self-prior* introduce in the Point2Mesh work [50]. We will refer our novel approach throughout this Chapter as **MLP-MLS**.

While the approach is still currently under development, this Chapter presents the work conducted todate. In this Chapter we begin by providing an overview of the approach. Then we consider the exploratory work conducted, initially in 1 dimension and then 2 dimensions, and we present the current issues faced, and an outlook to future work that is to be conducted.

7.1 MLP-MLS Overview

As we have seen in Chapter 4, MLS approaches to surface reconstruction typically uses a set of low-order fixed algebraic polynomial as the basis functions to construct the approximation. Under our novel MLP-MLS approach we replace these polynomials basis functions with a basis defined by a ReLU MLP

neural network. An important part of our approach is the separation between MLP basis representation and the MLS framework. While the MLP is trained with respect to the output of the MLS approximation, the MLS framework works independently of this training procedure, only operating on the basis functions defined by the MLP. This has the advantage, compared with IGR for example, of meaning that established theory of the MLS framework (e.g., the provability of *good* approximations [33]) can be directly imported into our approach.

The reconstruction algorithm works in two phases:

(1) **Training**: The MLP basis functions are trained to minimise an error term between the MLS approximation at each of the point-set positions and the ground-truth value (i.e., $\sum_i \|\hat{f}(\boldsymbol{x}_i) - f(\boldsymbol{x}_i)\| \; \forall \boldsymbol{x} \in \mathcal{P}$).

(2) **Execution**: After the MLP basis functions converges, the parameters of the MLP are fixed and the usual reconstruction pipeline for MLS is enacted: The local MLS optimisation problem is solved for each position within the domain to produce a global approximation $\hat{f}(\boldsymbol{x})$. Then for surface reconstruction the surface is extracted using Marching Squares [5].

The overall reconstruction pipeline is summarised visually in Fig. 7.1.

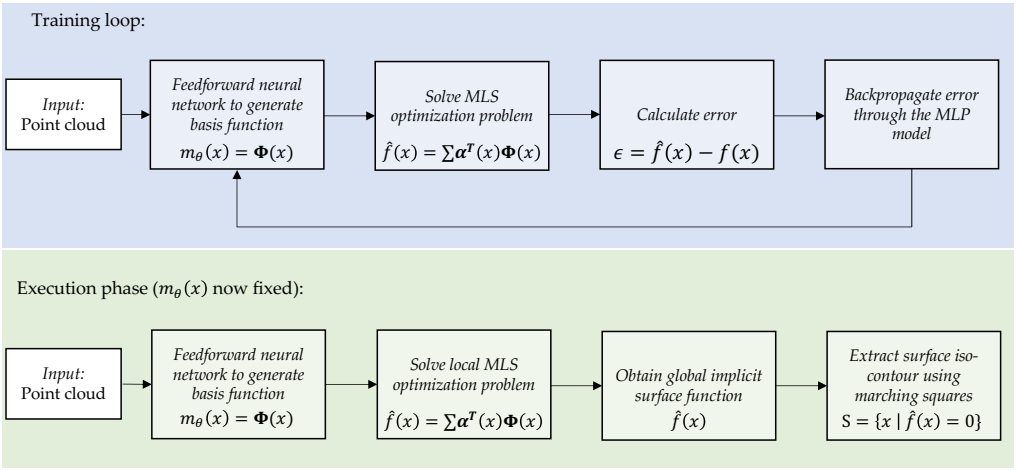


Figure 7.1: The MLP-MLS approach in general. There is a training phase where optimal basis functions are learned and an execution phase where the parameters of the MLP are fixed and the reconstruction surface is extracted.

7.2 MLP-MLS in 1D

Before considering MLP-MLS for surface reconstruction in 2 dimensions, we first explore the simpler case of 1 dimensional function approximation using the proposed MLP-MLS framework. The aim in working in 1 dimension is twofold: firstly, it enables us to establish that the proposed methods can produce satisfactory approximations. Secondly, it provides a testbed on which we can experiment with the MLP-MLP approach and the MLP architecture.

In this section we will make use of two test functions, which we introduce below. Then, we briefly introduce the reader to both the mathematical and computational implementation of the approach, and finally presenting the experiments conducted and the results obtained.

7.2.1 The Test Problem

In this section we make use of two test functions: the first function, defined by:

$$f_1(x) = \sin(4.9x) + 2\cos(10x) + 3\sin(8.9x) - 1, \tag{7.1}$$

yields a modulated periodic function, shown in black in Fig. 7.2 (left). The second function defined by:

$$f_2(x) = |x - 0.5|, \tag{7.2}$$

yields a function with a discontinuous derivative (i.e., a sharp geometric feature). This function is plotted in black in Fig. 7.2 (right). We restrict our attention to the closed interval $x \in [0, 1]$ and take 40 uniformly sampled points to produce the point-set $\mathcal{P}_1$ and $\mathcal{P}_2$ for the functions $f_1(x)$ and $f_2(x)$, respectively. The point-set for both functions is plotted in yellow in Fig. 7.2.

7.2.2 The Approximation Procedure

We first introduce some notation. Let $\boldsymbol{m}_{\boldsymbol{\theta}} : \mathbb{R}^a \to \mathbb{R}^b$ denote a functional representation of an MLP. In the 1 dimensional case $a = 1$ and b is a user defined number of basis functions. The parameters, $\boldsymbol{\theta}$, of the MLP are defined by the architecture of the network. For brevity, we use the tilde symbol above a set of parameters $\tilde{\boldsymbol{\theta}}$ as a short-hand for defining the neural architecture such that $\tilde{\boldsymbol{\theta}} = [a, b, c]$ defines an MLP with 3 layers: a neurons in the first, b neurons in the second, and c neurons in the third.

Given an MLP, the functional we will construct the approximation with will be of the form:

$$\hat{f}(x) = \boldsymbol{\alpha}(x)^T \boldsymbol{m}_{\boldsymbol{\theta}}(x), \tag{7.3}$$

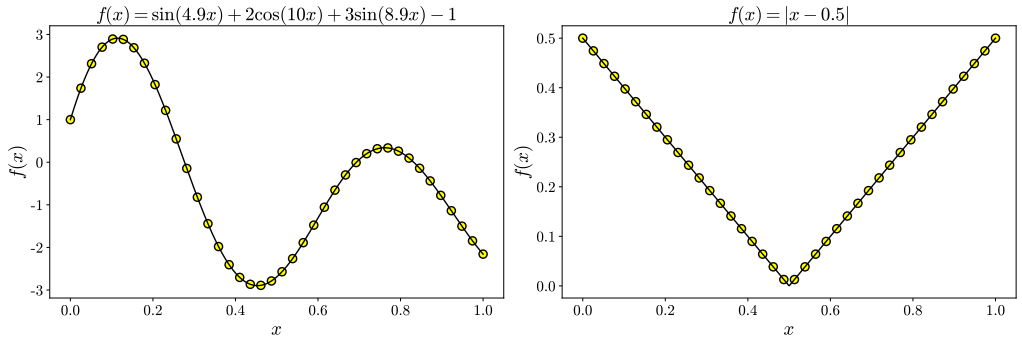


Figure 7.2: The test functions $f_1(x)$ (left) and $f_2(x)$ (right), plotted on the interval $x \in [0, 1]$. The yellow scatter points represent the point-sets $\mathcal{P}_1$ and $\mathcal{P}_2$ in the left and right plots, respectively.

where $\boldsymbol{\alpha}(x) = [\alpha_1(x), ..., \alpha_b(x)]^T$ is a vector of MLS coefficients. In this form, the coefficients $\boldsymbol{\alpha}(x)$ are then found using the usual MLS function approximation framework we introduced in Section 3.1.4. However, to obtain *optimal* basis functions, the MLP needs to be trained. In the following Section we define more precisely the training procedure.

Training the MLP

Before we begin training, the MLP first needs to be initialised. In this project we initialise with random weights and biases drawn from an $\mathcal{N}(0, 1)$ Gaussian distribution. Then the training procedure is as follows:

1. Each point of the point-set is pushed through the MLP, using the *feedforward* algorithm (see Section 3.2.2), to produce a matrix B, where:
$$B = \left[\boldsymbol{m_\theta}\left(x_1\right)^T, \boldsymbol{m_\theta}\left(x_2\right)^T, \ldots, \boldsymbol{m_\theta}\left(x_M\right)^T\right]^T. \tag{7.4}$$
2. A query point q is selected at random from $\mathcal{P}$.
3. The MLS approximation at q is then solved using Eq. (3.23) from Section 3.1.4 such that:
$$\hat{f}(q) = \boldsymbol{m_\theta}(q)^T\left[B^T W(q) B\right]^{-1} B^T W(q)\boldsymbol{y}, \tag{7.5}$$
where B is as defined in Eq. (7.4), $W(q)$ is the weight matrix defined in Eq. (3.19), and $\boldsymbol{y}$ is the vector of all $f(x_i) \in \mathcal{P}$.
4. The absolute error, ε, between the approximation $\hat{f}(x_i)$ and the true value $f(x_i) = \boldsymbol{y}_i$ is then evaluated:
$$\varepsilon = |\hat{f}(q) - f(q)|. \tag{7.6}$$

5. Steps 2 to 4 are repeated N times, with the cumulative sum of all ε from step 4 being denoted C. (i.e., $C = \sum_i \varepsilon_i$)
6. The cumulative error C is then back-propagated through the network, using the backpropagation algorithm (see Section 3.2.3); this defines the end of an epoch.
7. Steps 1 to 6 are then repeated for either a fixed number of epochs E, or until the cumulative cost converges to a minimum.

Once the MLP has been trained, the parameters $\boldsymbol{\theta}$ are fixed, and the function can be reconstructed. A dense set of points, $R \subset [0, 1]$, is then generated. Then for each $r \in R$, the function is approximated:

$$\hat{f}(r) = \boldsymbol{m}_{\boldsymbol{\theta}}(r)^T \left[B^T W(r) B\right]^{-1} B^T W(r) \boldsymbol{y}, \tag{7.7}$$

where B, $W(r)$ and $\boldsymbol{y}$ are as defined in step 3 above.

The MLPs used within this project are built using the `TensorFlow` deep learning framework. Furthermore, for computational efficiency, the MLS approximation also is carried out using the `TensorFlow` maths API; this prevents unnessassery conversion of Tensor objects to Numpy arrays, and vice-versa.

7.2.3 Experiments in 1D

In this Section we describe the experiments conducted to approximate the functions $f_1(x)$ and $f_2(x)$ defined in Eq. (7.1) and Eq. (7.2), respectively. There are three central hypotheses we will seek to consider in this Section, they are stated formally below.

Hypothesis 7.1: (Periodic and Sharp Feature Representation).
The MLP-MLS approach has the expressivity to represent both periodic and sharp features features of a function.

Hypothesis 7.2: (Network Architecture).
The network architecture (the arrangement of weights and biases) of the MLP effects the MLS approximation achieved.

Hypothesis 7.3: (Optimality of Basis Function).
For a fixed MLS approximation framework (i.e., fixed weighting variance σ^2 and number of basis functions b), a *better* approximation can be achieved via a set of learned basis functions compared with a fixed polynomial basis.

Of course, the notion of *better* in Hypothesis 7.3 must be more formally defined before it can be tested; in the 1 dimensional case we use the metric defined in Def. 7.1 to define which approximation is better.

Definition 7.1: (1D Function Approximation Metric).

Let $\mathcal{X} = \{0, \frac{1}{h_i}, \frac{2}{h_i}, ..., 1\}$ where h_i is some user defined discretisation parameter, then we define the metric between the approximation $\hat{f}(x)$ and the ground-truth $f(x)$ as

$$d(\hat{f}, f) = \sum_{x \in \mathcal{X}} \|\hat{f}(x) - f(x)\|_1. \tag{7.8}$$

Experimental Setup: Periodic Function Modelling

We begin by investigating the modulated periodic function $f_1(x)$. In this experiment, we fix the value of the MLS weighting term, σ^2 and the number of basis functions, b, to be 0.05 and 4, respectively. We then run the MLP-MLS reconstruction, as detailed in the Section 7.2.2, but with varying network architectures $\tilde{\boldsymbol{\theta}}$. The MLP-MLS training phase is run for a total of 200 epochs, with a batch size of 20, for each of the network architectures. The reconstruction metric for each of the architectures was constructed with $h_i = 1000$.

Results: Periodic Function Modelling

The quantitative reconstruction errors are tabulated in Fig. 7.3 and the qualitative results of the final reconstructions are shown in Fig. 7.5.

As is clear from Fig. 7.3, and the qualitative reconstructions achieved in Fig. 7.5, the network architectures has a dramatic effect on the resulting approximations achieved; this provides evidence for Hypothesis 7.2. Interestingly, it appears that increasing the number of layers past a certain number had a detrimental effect on the approximation achieved. Recall that in Section 3.2.4 we noted that increasing the number of layers causes the number of linear regions in the output to grow exponentially compared with the linear scaling when increasing layer widths. The fact that the reconstruction accuracy decreased likely indicates that the MLP provided too many degrees of freedom to be constrained by the problem and that the network was *overfitting* the data.

In Fig. 7.4 we plot the approximation achieved via using the standard MLS approximation with a fourth order polynomial. The approximation accuracy achieved was 0.054. This provides evidence to support Hypothesis 7.3, as the

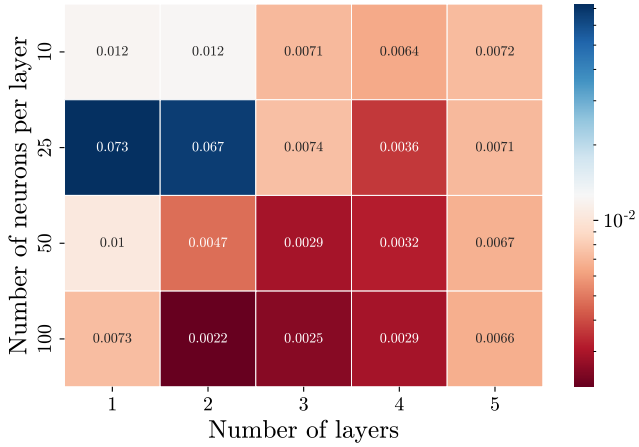


Figure 7.3: Shows the quantitative results for reconstruction of the modulated periodic function. The better the reconstruction the smaller the metric value.

best accuracy under the MLP-MLS approach was 0.0022, a 95% increase in the percentage accuracy.

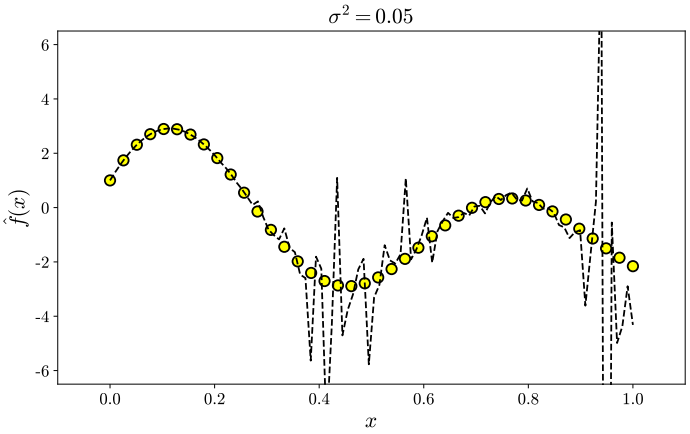


Figure 7.4: Shows the approximation of $f_1(x)$ achieved via the standard MLS approach to function fitting using a forth order polynomial basis. The approximation accuracy was 0.054.

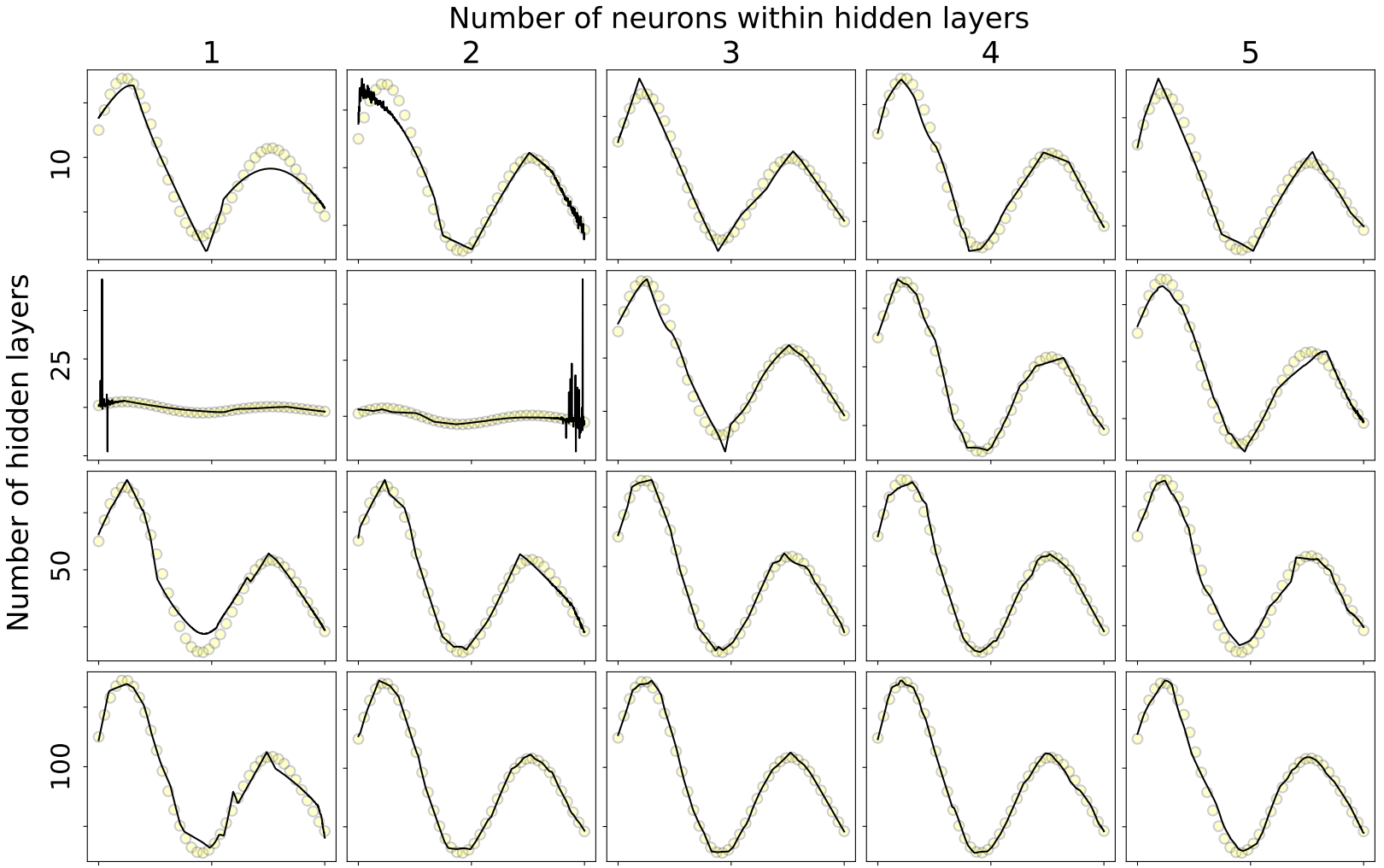


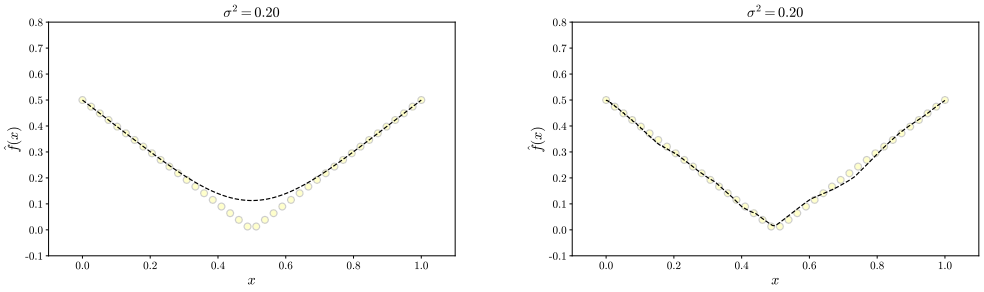
Figure 7.5: Shows the qualitative reconstruction of the point-set $\mathcal{P}_1$ using the MLP-MLS approach for different network architectures.

Experimental Setup: Sharp Feature Function Modelling

In this second set of experiment we seek to establish that the MLS-MLP approach can represent sharp features within functions. As we can see that $f_2(x)$ is comprised of two linear regions, we make use of the heuristics introduced in Section 3.2.4 to select a compact neural architecture $\tilde{\boldsymbol{\theta}} = [10]$. This time we use a wider Gaussian weighting variance of $\sigma^2 = 0.2$ but keep the number of basis function the same as previously, $b = 4$. All other training parameters remain the same as the previous experiments.

Results: Sharp Feature Function Modelling

The qualitative results from sharp feature modelling are shown in Fig. 7.6. In Fig. 7.6 (a) we plot the result from approximating $f_2(x)$ using the standard MLS framework using 4 polynomial basis functions ($\phi_i(x) = x^i | i \in \{1, 2, 3, 4\}$). In this case, the MLS approach produces an overly smooth approximation that does not capture the sharp feature of the edge. In Fig. 7.6 (b) we plot the result approximation from using the MLP-MLS approach. In this second case, the approximation captures the sharp feature well, but seems to produce poor approximations to the linear regions compared with the standard MLS approach.



(a) (b)

Figure 7.6: Show the reconstruction of the point-set $\mathcal{P}_2$, using (a) MLS with a 4th order polynomial basis (b) MLP-MLS with 4 learned basis functions.

Nevertheless, the results of this experiment do provide evidence for Hypothesis 7.1, that the MLP-MLS approach can represent sharp features of functions. Furthermore, it also provides further evidence for Hypothesis 7.3 as with fixed MLS approximation parameters the qualitatively results show that MLP-MLS achieves a better approximation, passing close to more of the points.

7.3 MLP-MLS in 2D

Now we have explored the 1 dimensional case of using MLP-MLS for function approximation, we now consider the more difficult task of using the framework for surface reconstruction in 2D. In the following section we introduce the changes that are required to use MLP-MLS for surface reconstruction, the preliminary experiments conducted, and the results obtained.

7.3.1 Alterations to the MLP-MLS Approach

Moving to two dimensions for surface reconstruction requires two significant changes to the approximation approach, compared with the approach presented in Section 7.2.2. The first is to ensure that we are able to regress a signed distance function (SDF) and the second is to ensure that the MLP operates in on a common coordinate space; we outline these changes in more detail below.

Point-set Constraints

Firstly, the function value $\boldsymbol{y}$, which we seek to approximate via MLP-MLS is inherently 0 at the points of the point-set when approximating a SDF. To prevent the trivial solution (i.e., $f(x) = 0 \; \forall x \in \mathbb{R}$) we must regularise the approximations process by imposing further constraints. In this preliminary work, we use the pseudo-normal constraints method, described in [61]. Suppose we have a point-set $\mathcal{P}$ that contains both point positions and surface normals, then an augmented point-set $\mathcal{P}'$ is constructed such that:

$$\mathcal{P}' = \{f(\boldsymbol{x}_i) = 0 \mid \forall \; \boldsymbol{x}_i \in \mathcal{P}\} \cup \{f(\boldsymbol{x}_i \pm \varepsilon) = \pm\varepsilon \mid \forall \; (\boldsymbol{x}, \boldsymbol{n}_i) \in \mathcal{P}\}, \tag{7.9}$$

where ε is a small value (e.g., $\epsilon = 10^{-4}$); this idea is illustrated for the unit circle in Fig. 7.7.

The augmented point-set $\mathcal{P}'$ prevents the trivial solution as the point-set now also contain non-zero function values. Furthermore, as the function value a small step above and below the surface is taken to be equal to the step size, the problem becomes constrained to approximating an SDF in close proximity to the surface.

MLP Coordinate System

The second challenge comes from using an MLP that operates on coordinate space. One of the main principles of using the MLP-MLS approach is that the MLP basis functions should be able to *learn* reoccurring patterns within an objects geometry, also called a *self-prior*. To facilitate this, it is import to operate the MLP within a common coordinate frame which we will refer to as the *local*

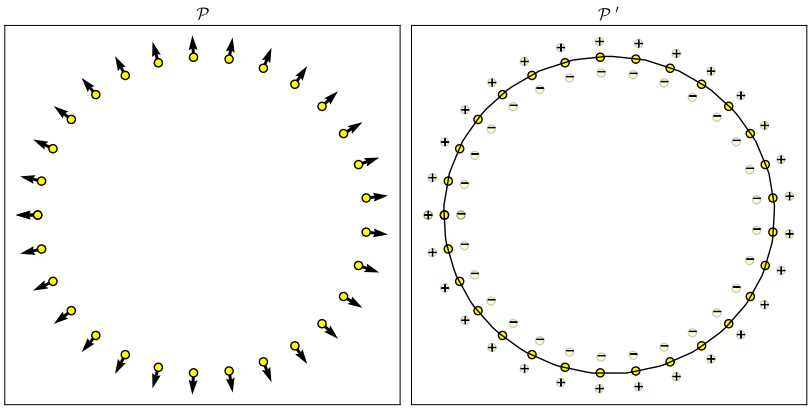


Figure 7.7: The Pseudo-normal constrain method restricts the value of the SDF to 0 at each $x \in \mathcal{P}$, a positive ε value slightly outside the surface along the normal direction, and a negative ε value slightly inside the surface along the normal direction. Left: a set of sample points with surface normal information; Right: point-set with pseudo-normal constraints.

coordinate system. For a query point q which we seek to construct the MLS approximation around, the local coordinate system is constructed in the following way:

1. The k nearest neighbours of q are selected from the point-set using a KDTree[1] sampling procedure.
2. The average point-set position, $\tilde{\boldsymbol{x}}$, and average surface normal vector, $\tilde{\boldsymbol{n}}$, are calculated from the k nearest neighbours of q.
3. The entire augmented point-set is then linear transformed via a translation by $-\tilde{\boldsymbol{x}}$ and a rotation through the angle θ, where $\theta = \arctan(\tilde{\boldsymbol{n}}_y/\tilde{\boldsymbol{n}}_x)$.

The two transformations in step 3 have the effect of moving the origin of the coordinate space to average position of the k nearest neighbours, $\tilde{\boldsymbol{x}}$ and aligning the average surface normal vector, $\tilde{\boldsymbol{n}}$, with the y axis; we refer to this as the local coordinate space for q. We also note that as the transforms are linear, the inverse mapping from the coordinate space to the original global space is trivial to find, requiring only the change of sign on the translation vector and rotation angle.

7.3.2 MLP-MLS Reconstruction: Framework

With the alterations to the MLP-MLS approach introduced in the previous Section, the updated MLP training procedure is shown in Fig. 7.8.

[1]The KDTree is a data-structure that allows for efficient distance querying of scattered data-points [62].

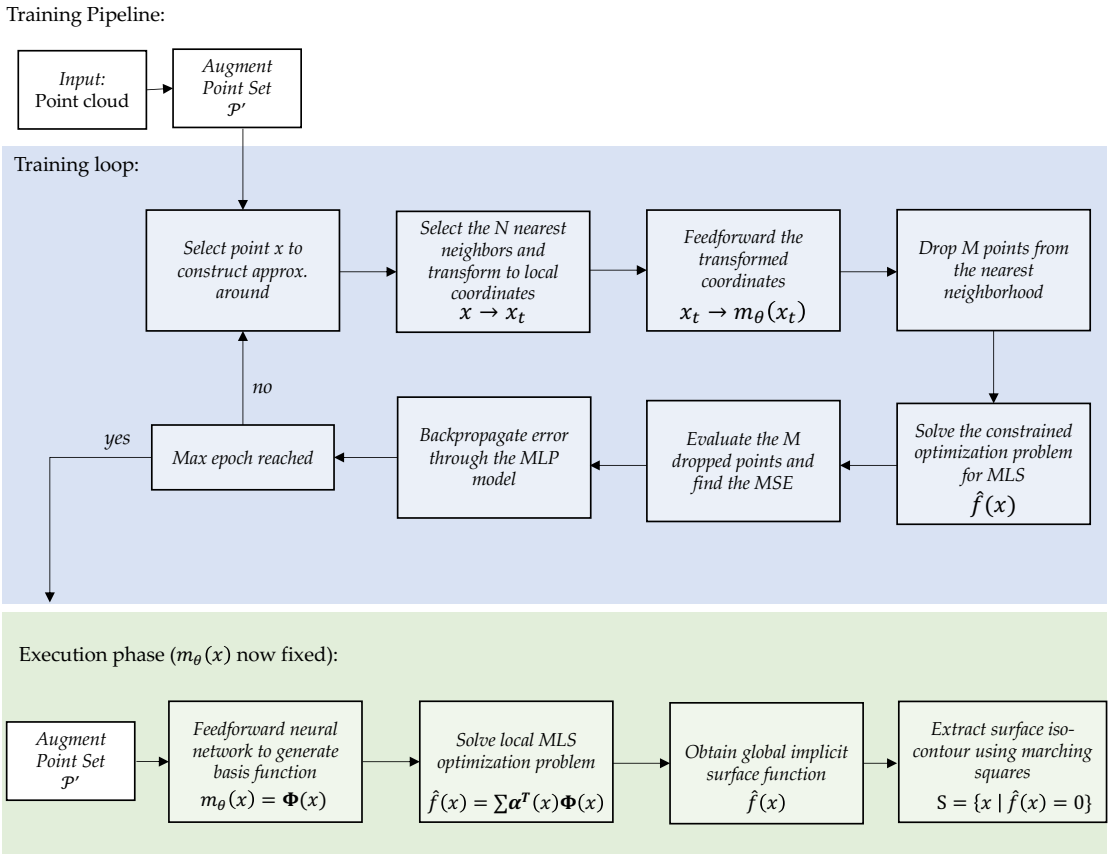


Figure 7.8: Shows the overall MLP-MLS pipeline for 2D surface reconstruction. The procedure is broken into two parts: (1) a training phase, where *optimal* basis function are learned using an MLP. (2) An execution phase, where the MLP has fixed weights and the surface is extracted using marching squares.

The majority of the 2D MLP-MLS implementation inherits existing code from the 1D function approximation case, with necessary adjustments made to accommodate the structure of point-sets (positions and surface normals), the augmentation using the pseudo-normal constraint method, and the local coordinate frame construction. Furthermore, a new pipeline for visualisation was written to enable efficient extraction of iso-contours from implicit fields; the visualisation code leverages the `Matplotlib` python package and its built in marching squares algorithm.

7.3.3 MLP-MLS Reconstruction: Experiments

While MLP-MLS is still under development, in this section we present experiments conducted on its current implementation. We make use of three point-sets from the shapeset dataset (see Chapter. 5), namely: Thumb at 50% ground-truth density, Hex Star at 25% ground-truth density, and Rabbit at 10% ground-truth density; we refer to these as `thumb-50`, `hex-star-25`, and `rabbit-10`, respectively.

The MLP-MLS training phase, for each of the point-sets, was run for 200 epochs, with a k nearest neighborough size of 40 and $M = 10$ drop points. The Gaussian weight term for the MLS approximation was selected via empirical experimentation to be $\sigma = 0.05$ for all point-sets considered. The execution phase of the MLP-MLS used marching squares on a meshed grid of size 200×200.

As comparison methods, we also reconstructed the three points-sets using IMLS, RIMLS, and IGR methods implemented in the `PyPointset` library (see Chapter 6). The parameters for IMLS and RIMLS where selected empirically for each point-set and are summarised in the table below. The IGR method was run for 5000 epochs for each of the three point-sets.

	IMLS	RIMLS
Thumb-50	$\sigma^2 = 0.001$	$\sigma^2 = 0.001$, $\sigma_r^2 = 0.01$, $\sigma_n^2 = 0.006$
Hex-Star-25	$\sigma^2 = 0.04$	$\sigma^2 = 0.08$, $\sigma_r^2 = 0.5$, $\sigma_n^2 = 0.006$
Rabbit-10	$\sigma^2 = 0.1$	$\sigma^2 = 0.15$, $\sigma_r^2 = 0.6$, $\sigma_n^2 = 0.006$

Table 7.1: Tabulates the parameters used in the IMLS and RIMLS reconstruction methods. The parameter σ^2 refers to the spatial Guassian weighting term, the parameter σ_r^2 refers to the weighting term on the spatial residual in RIMLS and σ_n^2 refers to the weighting term on the residual normal error in RIMLS.

7.3.4 MLP-MLS Reconstruction: Results

In this Section we present the results obtain from the experiments detailed in the previous Section. To evaluate the quality of the reconstructed surface obtained, we the make use of the standard surface reconstruction metric, the Chamfer distance, which has been used in [33, 46, 63]. We formally define the Chamfer distance below.

Definition 7.2: (Chamfer Distance).
Let X represent the set of points that constitute the reconstructed surface and let Y represent the ground-truth point-set, then the Chamfer distance between the reconstructed surface and the ground-truth point-set is defined by

$$d_{CD}(X,Y) = \sum_{\boldsymbol{x}\in X} \min_{\boldsymbol{y}\in Y} \|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + \sum_{\boldsymbol{y}\in Y} \min_{\boldsymbol{x}\in X} \|\boldsymbol{x}-\boldsymbol{y}\|_2^2. \tag{7.10}$$

It was observed that the current implementation of MLP-MLS produces implicit fields with significant artifacts, manifesting as 0 iso-contours far away from the surface. We will discuss possible reasons for these artifacts later within this Chapter. In presenting the results in this Section, we present the reconstruction including these artifact as MLP-MLS and with the artifact manually removed as MLP-MLS (Cleaned). This allows us to consider the merits and inferiorities of the approach both in its current implementation and in future potential implementation if the approach can be adjusted to remove artifact.

The qualitative reconstruction for `thumb-50`, `hex-star-25`, and `rabbit-10` are shown in Fig. 7.9, Fig. 7.10, and Fig. 7.11, respectively. The quantitative Chamfer distance metric, for each method, is tabulated in Table 7.2.

	Reconstruction methods d_{CD} ($\times 10^{-4}$)				
Dataset	IMLS	RIMLS	IGR	MLP-MLS	MLP-MLS (clean)
Thumb-50	18.3	**3.1**	8.2	82.3	6.4
Hex-Star-25	102.1	18.2	**7.4**	72.3	16.1
Rabbit-10	43.2	**6.1**	6.5	94.3	14.5

Table 7.2: Table shows the calculated Chamfer distance between the reconstructed surface and the ground-truth point-set (i.e., `point-set_name`-100). The MLP-MLS (clean) class refers to the surface obtained via MLP-MLS after removing the spurious iso-contours present. The best methods Chamfer distance for each dataset is indicated in bold font.

7.3.5 MLP-MLS Reconstruction: Discussion

Evidenced by Table 7.2, MLP-MLS clearly does not achieve state-of-the-art reconstruction in its current state. However, considering the MLP-MLS approach after removal of artifacts shows that the surface reconstruction consistently performs better than the IMLS approach. Furthermore, considering the qualitative reconstruction in Fig. 7.9, we see that the MLP-MLS method is able to handle sharp features (see the red inset) and also periodic features (see the blue inset) of the geometry better than both IGR and IMLS. Nevertheless, even in the cleaned

case MLP-MLS appears to struggle to produce straight line, something we also noted in our 1D experiments; this too must be rectified for MLP-MLS to be a useful technique.

To improve the MLP-MLS approach, we must first identify the cause of the current artifact that are appear within the reconstruction surface. The medial axis defines regions which have more than one nearest boundary point. Considering the reconstruction obtained for the MLP-MLS for the `hex-star-25` point-set in Fig. 7.10, we see that the artifacts lay almost exactly along the medial axis of the star. This possibly indicates that our approach to constructing a local coordinate system is not robust enough, as a small step either side of the medial axis results in dramatically different definitions of *local regions*. Future work on this framework will seek to rectify this, along with implementing more sophisticated SDF constraining methods.

Figure 7.9: Shows the reconstructed surface for the `thumb-50` point-set using the three benchmark comparison methods and MLP-MLS. The MLP-MLS column refers to the raw output of our method and the MLP-MLS (Cleaned) column refers to the reconstruction after manual removal of artifacts.

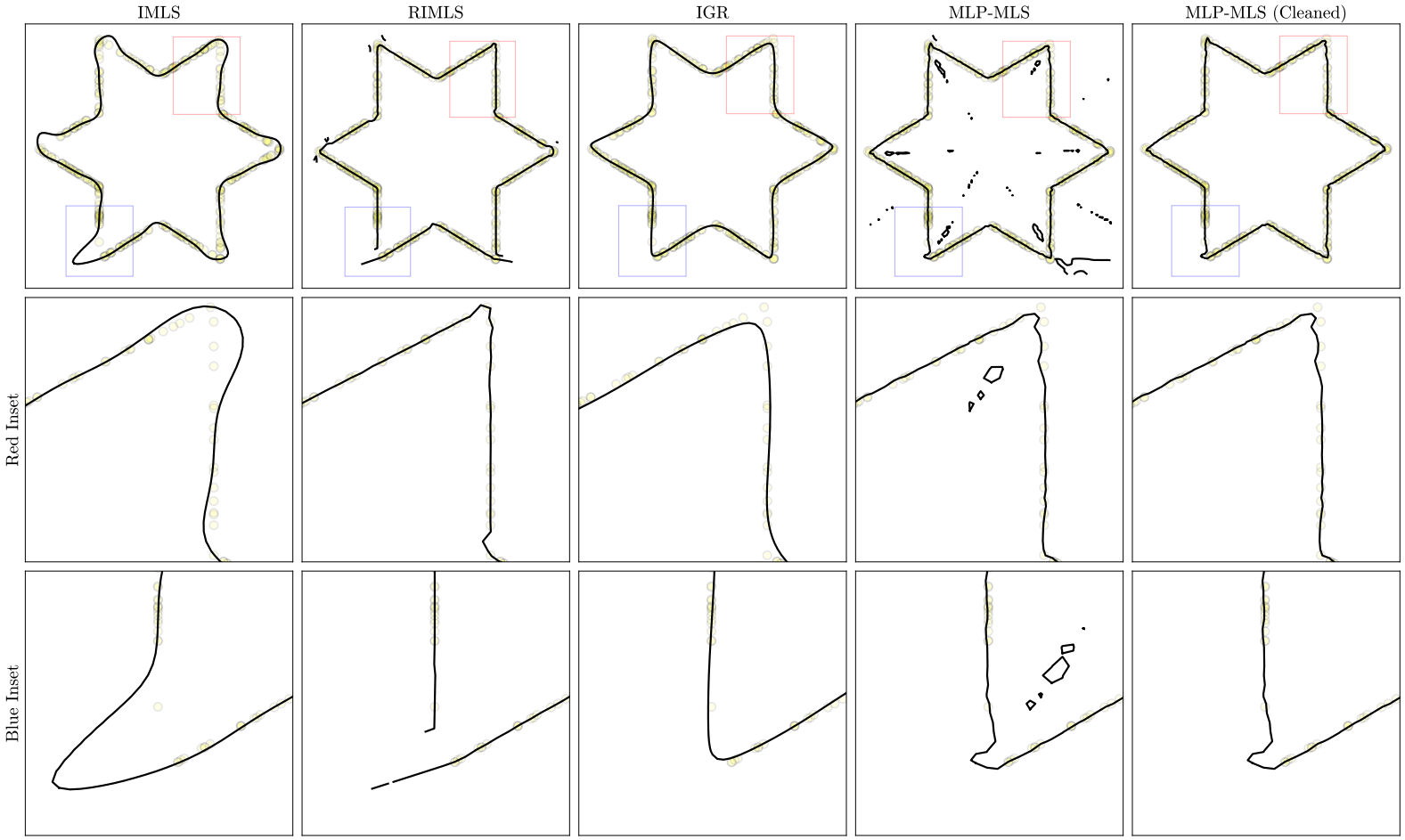


Figure 7.10: Shows the reconstructed surface for the `hex-star-25` point-set using the three benchmark comparison methods and MLP-MLS. The MLP-MLS column refers to the raw output of our method and the MLP-MLS (Cleaned) column refers to the reconstruction after manual removal of artifacts.

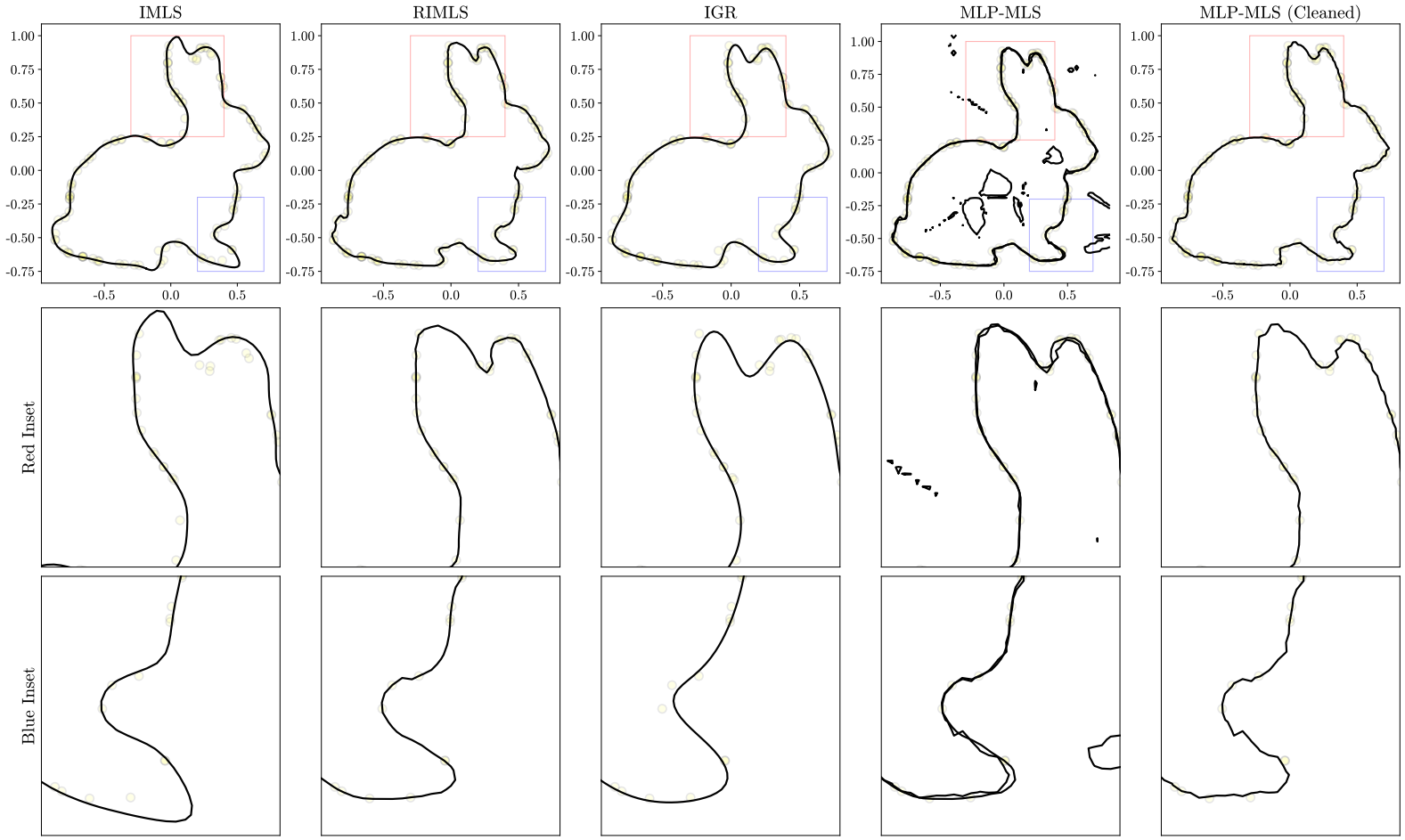


Figure 7.11: Shows the reconstructed surface for the `rabbit-10` point-set using the three benchmark comparison methods and MLP-MLS. The MLP-MLS column refers to the raw output of our method and the MLP-MLS (Cleaned) column refers to the reconstruction after manual removal of artifacts.

Chapter 8

Conclusion & Future Work

In this final Chapter we aim to bring the report to a conclusion. The fundamental aim of this project was to investigate a novel approach to surface reconstruction, we refer to as MLP-MLS. To enable the development of a novel surface reconstruction approach we first constructed our own 2D point-set dataset we refer to as the Shapeset dataset; we present the development of this in Chapter 5. Then in Chapter 6, we presented three benchmark approaches to surface reconstruction and the `PyPointset` library which provides efficient implementations of the methods in Python. Finally, in Chapter 7, we present the initial implementation, experiment and results from our novel surface reconstruction approach.

While the results obtained for the 2D surface reconstruction method show that MLP-MLS approach achieves inferior Chamfer distance scores compared with RIMLS and IGR, the method does show merit. In particular the method shows a strong potential at representing sharp and periodic features within geometry, something typically quite challenging with only a local surface prior. Nevertheless, for the MLP-MLS approach to achieve high fidelity reconstruction, it must overcome several limitations. In the remained of this Chapter, we outline several directions for future work on this project to take.

The first limitation that must be addressed in the 2D case is the artifacts that appear in the MLP-MLS reconstruction surface. As outlined in Section 7.3.5, the initial next steps are to consider the mechanism by which we define the local coordinate system and improve its robustness, especially along the medial axis. Furthermore, the simplistic approach to enforcing a signed distance function we implemented in this project (psuedo-normal constraints) should be changed for a more robust alternative. Incorporating the Eikonal equation, introduced in the IGR method in Section 6.1.3, directly into the MLS approximation framework would be an interesting avenue to explore.

The second line of work revolves around the selection of parameters within the MLS framework. It is the current authors view that if we can define what we mean by *optimal*, then data-driven approaches to parameter selection are likely to be fruit-full. As an interesting next step, we would view parameter selection from the lens of Bayesian inverse problems and make use of Markov Chain Monte Carlo

techniques to produce posterior distributions over parameter space. Further work could then take place on using MLPs to learn *optimal* parameters, in much the same way we aimed to learn optimal basis function in this project.

The final line of work centers on moving the MLP-MLS approach to reconstruction of surfaces in 3D. This is perhaps the most technically challenging line of future work. Recall that in Section 7.3.1 we introduce the local coordinate frame; the idea was that the basis functions should be optimised such that the geometry in a local region to the query points appears in a common coordinate space, therefore allowing the basis functions to extract reoccurring features within the geometry (i.e., *a self-prior*). The extra degree of freedom brought by moving to 3D means that instead of considering tangent lines, like we did in this project, we must instead consider tangent planes. Tangent planes have a rotational degree of freedom, around the defined normal vector, which makes placement of a coordinate system such that reoccurring features can be extracted difficult. Theoretically, this requires parameterising the SO(3) rotation group, which is notoriously hard. However, there has been increasing attention on solving this problem using neural networks in recent years. Future work will seek to solve the rotation problem via training a secondary MLP to learn optimal coordinate spaces for the basis functions, opening the door for MLP-MLS to 3D reconstruction.

Bibliography

[1] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva, "A survey of surface reconstruction from point clouds," *Computer Graphics Forum*, vol. 36, no. 1, p. 301–329, Jan 2017.

[2] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 12, pp. 4338–4364, 2020.

[3] F. Pomerleau, F. Colas, R. Siegwart *et al.*, "A review of point cloud registration algorithms for mobile robotics," *Foundations and Trends® in Robotics*, vol. 4, no. 1, pp. 1–104, 2015.

[4] R. T. H. Collis, "Lidar," *Appl. Opt.*, vol. 9, no. 8, pp. 1782–1788, Aug 1970. [Online]. Available: http://opg.optica.org/ao/abstract.cfm?URI=ao-9-8-1782

[5] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[6] M. Levoy and G. Turk, "The stanford 3d scanning repository." [Online]. Available: http://graphics.stanford.edu/data/3Dscanrep/

[7] S. Dineen, *Multivariate calculus and geometry / Seán Dineen.*, ser. Springer undergraduate mathematics series. London: Springer, 1998.

[8] L. L. Mirsky, *An introduction to linear algebra / by L. Mirsky.* Oxford: Clarendon Press, 1955.

[9] J. M. Lee, *Riemannian manifolds : an introduction to curvature / John M. Lee.*, ser. Graduate texts in mathematics ; 176. New York: Springer, 1997.

[10] C. Öztireli, "*Further Graphics*," Department of Computer Science and Technology, University of Cambridge, November 2021.

[11] K. Jittorntrum, "An implicit function theorem," *Journal of Optimization Theory and Applications*, vol. 25, no. 4, pp. 575–577, 1978.

[12] A. Charnes, E. L. Frome, and P. L. Yu, "The equivalence of generalized least squares and maximum likelihood estimates in the exponential family,"

Journal of the American Statistical Association, vol. 71, no. 353, pp. 169–171, 1976. [Online]. Available: http://www.jstor.org/stable/2285762

[13] H. Cox, "A demonstration of taylor's theorem," *Camb. Dublin Math. J*, vol. 6, pp. 80–81, 1851.

[14] M. Merriman, *A List of Writings Relating to the Method of Least Squares: With Historical and Critical Notes*. Academy, 1877, vol. 4.

[15] K. Fogarty, "Project 2: (neural networks and machine learning)," MATH49111: Scientific Computing, 2020, University of Manchester.

[16] D. L. Elliott, "A better activation function for artificial neural networks," Tech. Rep., 1993.

[17] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *towards data science*, vol. 6, no. 12, pp. 310–316, 2017.

[18] P. Sibi, S. A. Jones, and P. Siddarth, "Analysis of different activation functions using back propagation neural networks," *Journal of theoretical and applied information technology*, vol. 47, no. 3, pp. 1264–1268, 2013.

[19] F. Emmert-Streib, Z. Yang, H. Feng, S. Tripathi, and M. Dehmer, "An Introductory Review of Deep Learning for Prediction Models With Big Data," *Frontiers in Artificial Intelligence*, vol. 3, no. February, pp. 1–23, 2020.

[20] C. F. Higham and D. J. Higham, "Deep learning: An introduction for applied mathematicians," *SIAM Review*, vol. 61, no. 4, pp. 860–891, 2019.

[21] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[22] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[23] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[24] B. Hanin and D. Rolnick, "Deep relu networks have surprisingly few activation patterns," *Advances in neural information processing systems*, vol. 32, 2019.

[25] M. Telgarsky, "Representation benefits of deep feedforward networks," *CoRR*, vol. abs/1509.08101, 2015. [Online]. Available: http://arxiv.org/abs/1509.08101

[26] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva, "A survey of surface reconstruction from point clouds," in *Computer Graphics Forum*, vol. 36, no. 1. Wiley Online Library, 2017, pp. 301–329.

[27] K. Fogarty, "Point cloud surface reconstruction," CMP9766M : Frontier of Robotics Research, 2022, University of Lincoln.

[28] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," in *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, 1992, pp. 71–78.

[29] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[30] A. Nealen, "An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation," *URL: http://www. nealen. com/projects*, vol. 130, no. 150, p. 25, 2004.

[31] D. Levin, "The approximation power of moving least-squares," *Mathematics of computation*, vol. 67, no. 224, pp. 1517–1531, 1998.

[32] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Computing and rendering point set surfaces," *IEEE Transactions on visualization and computer graphics*, vol. 9, no. 1, pp. 3–15, 2003.

[33] R. Kolluri, "Provably good moving least squares," *ACM Transactions on Algorithms (TALG)*, vol. 4, no. 2, pp. 1–25, 2008.

[34] N. Amenta and Y. J. Kil, "Defining point-set surfaces," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 264–270, 2004.

[35] M. Alexa and A. Adamson, "On normals and projection operators for surfaces defined by point sets." in *PBG*. Citeseer, 2004, pp. 149–155.

[36] A. Adamson and M. Alexa, "Approximating and intersecting surfaces from points," in *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 2003, pp. 230–239.

[37] G. Guennebaud, M. Germann, and M. Gross, "Dynamic sampling and rendering of algebraic point set surfaces," in *Computer Graphics Forum*, vol. 27, no. 2. Wiley Online Library, 2008, pp. 653–662.

[38] C. Shen, J. F. O'Brien, and J. R. Shewchuk, "Interpolating and approximating implicit surfaces from polygon soup," in *ACM SIGGRAPH 2004 Papers*, 2004, pp. 896–904.

[39] A. C. Öztireli, G. Guennebaud, and M. Gross, "Feature preserving point set surfaces based on non-linear kernel regression," in *Computer graphics forum*, vol. 28, no. 2. Wiley Online Library, 2009, pp. 493–501.

[40] P. J. Huber, "Robust statistics," in *International encyclopedia of statistical science*. Springer, 2011, pp. 1248–1251.

[41] S.-L. Liu, H.-X. Guo, H. Pan, P.-S. Wang, X. Tong, and Y. Liu, "Deep implicit moving least-squares functions for 3d reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1788–1797.

[42] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, "Reconstruction and representation of 3d objects with radial basis functions," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, pp. 67–76.

[43] M. Kazhdan, "Reconstruction of solid models from oriented point sets," in *Proceedings of the third Eurographics symposium on Geometry processing*, 2005, pp. 73–es.

[44] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006.

[45] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 3, pp. 1–13, 2013.

[46] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 165–174.

[47] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman, "Implicit geometric regularization for learning shapes," *arXiv preprint arXiv:2002.10099*, 2020.

[48] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-cnn: Octree-based convolutional neural networks for 3d shape analysis," *ACM Transactions On Graphics (TOG)*, vol. 36, no. 4, pp. 1–11, 2017.

[49] Z. Wang, P. Wang, Q. Dong, J. Gao, S. Chen, S. Xin, and C. Tu, "Neural-imls: Learning implicit moving least-squares for surface reconstruction from unoriented point clouds," *arXiv preprint arXiv:2109.04398*, 2021.

[50] R. Hanocka, G. Metzer, R. Giryes, and D. Cohen-Or, "Point2mesh: A self-prior for deformable meshes," *arXiv:2005.11084 [cs]*, May 2020, arXiv: 2005.11084. [Online]. Available: http://arxiv.org/abs/2005.11084

[51] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5828–5839.

[52] D. Schunck, F. Magistri, R. A. Rosu, A. Cornelißen, N. Chebrolu, S. Paulus, J. Léon, S. Behnke, C. Stachniss, H. Kuhlmann *et al.*, "Pheno4d: A spatio-temporal dataset of maize and tomato plant point clouds for phenotyping and advanced plant analysis," *Plos one*, vol. 16, no. 8, p. e0256340, 2021.

[53] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[54] "Free SVG," https://freesvg.org, accessed: 2022-08-21.

[55] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[56] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM national conference*, 1968, pp. 517–524.

[57] C. Oztireli, G. Guennebaud, and M. Gross, "Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression," *Computer Graphics Forum*, vol. 28, no. 2, pp. 493–501, 2009. [Online]. Available: https://hal.inria.fr/inria-00354969

[58] W. S. Cleveland, "Robust locally weighted regression and smoothing scatterplots," *Journal of the American statistical association*, vol. 74, no. 368, pp. 829–836, 1979.

[59] Y. Susanti, H. Pratiwi, S. Sulistijowati, T. Liana *et al.*, "M estimation, s estimation, and mm estimation in robust regression," *International Journal of Pure and Applied Mathematics*, vol. 91, no. 3, pp. 349–360, 2014.

[60] T. Oberhuber, "Numerical recovery of the signed distance function," in *Czech-Japanese Seminar in Applied Mathematics, Prague, Czech Republic*, 2004, pp. 148–164.

[61] C. Shen, "Building interpolating and approximating implicit surfaces using moving least squares," Ph.D. dissertation, University of California, Berkeley, 2006.

[62] P. Ram and K. Sinha, "Revisiting kd-tree for nearest neighbor search," in *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*, 2019, pp. 1378–1388.

[63] Z. Huang, Y. Wen, Z. Wang, J. Ren, and K. Jia, "Surface reconstruction from point clouds: A survey and a benchmark," 2022.