Scientific Computing, Modeling & Simulation
Savitribai Phule Pune University

Master of Technology (M.Tech.)
Programme in Modeling and Simulation

Internship Project Report

Self-supervised Neural Signed Distance Function Model For Surface Reconstruction

Shubham Godase
MT2308

Academic Year 2024-25

Scientific Computing, Modeling & Simulation
Savitribai Phule Pune University

Certificate

This is certify that this report titled

Self-supervised Neural Signed Distance Function Model For Surface Reconstruction

authored by

Shubham Godase (MT2308)

describes the project work carried out by the author under our supervision during the period from January 2025 to June 2025. This work represents the project component of the Master of Technology (M.Tech.) Programme in Modeling and Simulation at the Department of Scientific Computing, Modeling & Simulation, Savitribai Phule Pune University.

Mr.Jay Kumar Faldu, Software Engineer
Dassault Systems, Pune, India

Mr.Sudhir Paithankar, Senior Manager
Dassault Systems, Pune, India

Dr.Bhalchandra Gore,Internal Guide
SCMS-SPPU, Pune, India

Dr.Arun Banpurkar, Head
SCMS-SPPU, Pune, India

Scientific Computing, Modeling & Simulation
Savitribai Phule Pune University

Author's Declaration

Statement on Authorship. This document titled

> **Self-supervised Neural Signed Distance Function Model For Surface Reconstruction**

authored by me is an authentic report of the project work carried out by me as part of the Master of Technology (M.Tech.) Programme in Modeling and Simulation at the Department of Scientific Computing, Modeling & Simulation, Savitribai Phule Pune University.

Statement on Plagiarism. In writing this report, I have taken reasonable and adequate care to ensure that material borrowed from sources such as books, research papers, the internet, etc., is acknowledged as per accepted academic norms and practices in this regard. I have read and understood the University's policy on plagiarism (`http://unipune.ac.in/administration_files/pdf/Plagiarism_Policy_University_14-5-12.pdf` or the latest version thereof).

Statement on the use of LLMs/AI. I further certify that

1. any consultations with LLMs/AI tools have been duly acknowledged;
2. any material borrowed from LLMs/AI tools (including code, text, images, etc.) has been duly acknowledged; and
3. I understand that I am the one who is responsible for the work and the results presented in this report including any measures of accuracy or quality, etc.

Shubham Godase
MT2308

Contents

Certificate iii

Author's Declaration v

Table of Contents vii

Acknowledgments ix

Abstract xi

1 Introduction 1
- 1.1 Domain Background . . . 2
- 1.2 Problem Statement . . . 2
- 1.3 Literature Review . . . 3

2 Data 5
- 2.1 Data Overview . . . 5
- 2.2 Exploratory Data Analysis . . . 5
  - 2.2.1 Geometric Analysis of the Mesh . . . 5
  - 2.2.2 Calculation of Surface Properties . . . 6
  - 2.2.3 Point Sampling Techniques . . . 6
  - 2.2.4 Visualization and Analysis . . . 7
- 2.3 Data Preprocessing . . . 8

3 Part I:SDF Modeling and Methodology 9
- 3.1 Signed Distance Function (SDF) Model . . . 9
  - 3.1.1 What is an SDF? . . . 9
  - 3.1.2 Applications of SDFs . . . 10
- 3.2 Neural Network . . . 10
  - 3.2.1 Sinusoidal Representation Network (SIREN) . . . 10
  - 3.2.2 Advantages of SIREN for SDF Representation . . . 11
  - 3.2.3 Why Sine Activation and the frequency Parameter(ω_0)? . . . 11
  - 3.2.4 SIREN Model Summary . . . 11
  - 3.2.5 Network Parameter Initialization . . . 12
- 3.3 Loss Functions . . . 13
  - 3.3.1 Eikonal Loss (L_{Eik}) . . . 13
  - 3.3.2 Dirichlet Matching Loss (L_{DM}) . . . 13
  - 3.3.3 Dirichlet Non-Matching Loss (L_{DNM}) . . . 13
  - 3.3.4 Alignment Loss (L_{AN}) . . . 14
  - 3.3.5 Neumann Loss ($L_{Neumann}$) . . . 14
  - 3.3.6 Singular Hessian Loss ($L_{SingularH}$) . . . 14

3.4 Neural Network Training Process . . . 14
3.4.1 Total Losses . . . 15
3.4.2 Annealing of the Hessian Loss (τ) . . . 15
3.4.3 Backpropagation and Gradient Computation . . . 15
3.4.4 Mini-Batch Gradient Descent . . . 16
3.4.5 Learning Rate Decay (CosineAnnealingLR) . . . 16
3.4.6 Dynamic Loss Weighting (Kendall's Method) . . . 16
3.4.7 Gradient Clipping . . . 17
3.4.8 Empirical Hyperparameter Optimization Strategy . . . 17
3.4.9 Implementation Details . . . 17

4 Part II: Surface Reconstruction using Marching Cubes 19
4.1 Overview of the Marching Cubes Algorithm . . . 19
4.2 Fundamentals of Marching Cubes Algorithm . . . 19
4.3 Implementation Details and Optimization . . . 21
4.4 Performance Evaluation of Surface Reconstruction . . . 21
4.5 Concluding Insights on Marching Cubes Integration . . . 21

5 Results 23
5.1 Reconstructed Shapes Results . . . 23
5.2 Quantitative Results . . . 28
5.2.1 Evaluation Metrics . . . 28
5.2.2 Metric Summary Table . . . 28
5.2.3 Quantitative Evaluation Results of All Models . . . 29
5.2.4 Evaluation Metrics Table For All Models . . . 33

6 Conclusion and Future Work 35
6.1 Conclusion . . . 35
6.2 Future Work . . . 36

Bibliography 37

Acknowledgements

I take great pleasure in expressing my sincere gratitude to all who supported and guided me throughout this journey.

First and foremost, I extend my heartfelt thanks to **Mr. Jay Kumar Faldu**, my industry mentor at Dassault Systèmes. His technical guidance, thoughtful insights, and steadfast encouragement were vital in shaping the direction of this project.

I am equally grateful to **Mr. Sudhir Paithankar**, Senior Manager at Dassault Systèmes, for helping me align my research with industrial goals and for offering valuable direction at every stage of the project.

I sincerely thank the entire Dassault Systèmes Simulia Abaqus CAE team for their collaborative support during technical challenges. The opportunity to work on real-world engineering problems significantly enhanced my understanding of AI/ML systems in production environments.

I am deeply indebted to my academic advisor, **Dr. Bhalchandra Gore**, my internal guide at the university, for his unwavering academic support and mentorship. His timely feedback, insightful suggestions, and consistent encouragement were instrumental in shaping both the theoretical framework and practical execution of this thesis. Working under his guidance has been an immensely enriching experience.

My heartfelt appreciation also goes to faculty members **Dr. Bhalchandra Pujari**, **Dr. Mihir Arjunwadkar**, and **Dr. Vaishali Shah** for their continuous academic guidance and insightful suggestions. Their encouragement fostered a strong foundation in scientific thinking and research methodology.

I would also like to thank all the faculty and staff of the Scientific Computing, Modeling and Simulation Program at Savitribai Phule Pune University. Their teaching and support helped build the essential skills and academic environment necessary for the successful completion of this work.

Special thanks to **Mr. Sandip Desai** for his collaborative spirit, technical insights, and consistent support throughout the research journey. His peer contributions have added great value to this project.

I gratefully acknowledge the role of open-source tools and platforms, including Python, PyTorch, scikit-learn, Open3D, scikit-image, Trimesh, Matplotlib, Plotly, and Google Colab. Access to GPU-enabled infrastructure made the technical implementation of this work possible. The use of Large Language Models (LLMs) also played an important role by aiding ideation, summarization, and research comprehension throughout the project.

On a more personal note, I would like to thank **Ms. Rani Godase** for her constant encouragement, patience, and emotional support, especially during challenging moments. Most importantly, I am forever grateful to my family. Their unconditional love, belief in me, and unwavering support have been the bedrock of this journey. I dedicate this work to them.

Abstract

Reconstructing continuous surfaces from unstructured point cloud data remains a fundamental challenge in 3D computer vision, especially in the absence of mesh connectivity, labeled distances, or normal supervision. This work introduces a self-supervised learning framework that models 3D geometry as a continuous Signed Distance Function (SDF) using a Sinusoidal Representation Network (SIREN). The proposed method leverages geometric constraints to learn implicit surfaces directly from sampled point clouds, without requiring ground-truth signed distances.

To guide the network towards accurate reconstruction, we employ a suite of geometric loss functions, including Eikonal loss for unit gradient norm enforcement, Dirichlet losses to regulate surface anchoring and spatial separation, and a Hessian-based loss to preserve sharp features and curvature continuity. The training pipeline is enhanced with adaptive loss reweighting via uncertainty modeling, cosine annealing for learning rate decay, and gradient clipping for stability.

Once trained, the continuous SDF is discretized over a voxel grid, and the reconstructed surface is extracted using the Marching Cubes algorithm. The approach demonstrates strong generalization to both simple and complex shapes, enabling the generation of high-resolution, watertight meshes from raw, noisy point data. This framework provides a scalable and label-free solution for surface reconstruction with broad applicability in fields such as medical imaging, robotics, and CAD design.

Chapter 1

Introduction

Surface reconstruction from 3D point clouds plays a vital role in computer graphics, autonomous systems, medical imaging, and virtual/augmented reality. Point clouds are unstructured sets of 3D points typically obtained from scanning tools such as LiDAR, structured light systems, or photogrammetry. While these data sources effectively capture surface geometry, they inherently lack connectivity, orientation, and completeness. This introduces major challenges in generating continuous, watertight, and topologically consistent surface representations.

Traditional methods for surface reconstruction, such as Poisson surface reconstruction, Delaunay triangulation, and volumetric integration, often assume that the input data is dense and noise-free, with precomputed surface normals. In real conditions, however, point clouds tend to be sparse, noisy, and lack normal orientation, especially when captured from mobile or cheap sensors. Consequently, traditional techniques may create artifacts, incomplete surfaces, or overly smooth geometries, particularly in the presence of sharp edges or thin forms.

Recent advances in neural implicit representations have offered promising alternatives. These models represent 3D geometry using continuous functions, such as Signed Distance Functions (SDFs), which implicitly define a surface as the zero level set of a scalar field. Neural networks, particularly Multi-Layer Perceptrons (MLPs), are trained to approximate these fields, offering flexible and high-fidelity reconstruction from incomplete or sparse data. Among them, Sinusoidal Representation Networks (SIRENs) stand out due to their ability to capture fine-scale details using periodic activation functions.

This thesis introduces a self-supervised neural pipeline for reconstructing high-quality surfaces from sparse and unoriented point clouds. Figure 1.1 presents a high-level illustration. The system combines SIRENs, geometric regularization losses, and a multi-scale data sampling strategy to learn a robust SDF representation without requiring ground-truth meshes or normals. Meshes are extracted using the Marching Cubes algorithm and evaluated using Chamfer Distance and normal consistency.

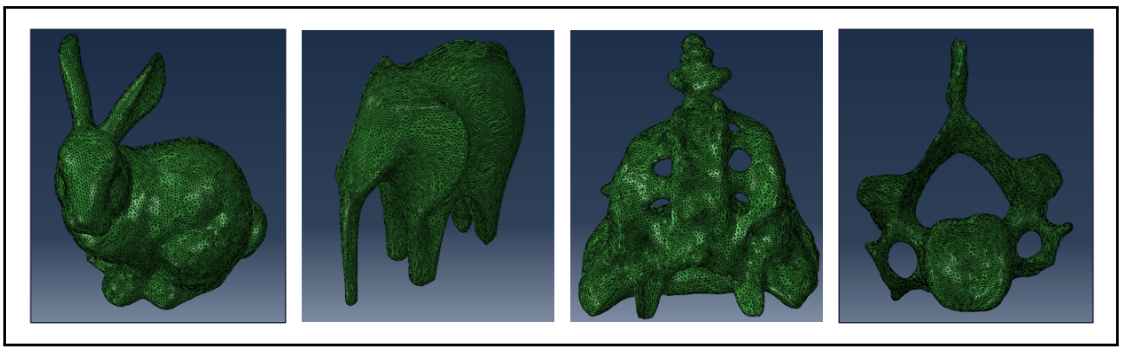


Figure 1.1: Surface reconstructions from point clouds using our self-supervised SDF model.

1.1 Domain Background

3D reconstruction is a fundamental problem in computer vision with wide applications in areas such as robotics, digital heritage preservation, and simulation. The use of point clouds as a primary 3D representation has gained popularity due to their simplicity and sensor compatibility. However, their unstructured nature poses unique challenges for downstream processing. To convert raw point clouds into usable 3D models, techniques that infer surface topology and fill missing information are required.

Implicit neural representations have reshaped the landscape of geometry processing. Unlike explicit methods that rely on mesh connectivity or voxel grids, implicit models define surfaces implicitly using neural networks. A common representation is the Signed Distance Function (SDF), where each point in space is assigned a scalar indicating its distance from the surface. This formulation enables the representation of complex geometries with smooth transitions and provides differentiable properties that are useful for optimization and learning.

1.2 Problem Statement

Despite the progress in neural implicit modeling, several core challenges remain in surface reconstruction from point clouds:

1. **unoriented point cloud data:**Reconstructing continuous and watertight 3D surfaces from sparse, noisy, and unoriented point cloud data remains a significant challenge in 3D computer vision. Traditional reconstruction methods often require dense sampling, ground-truth mesh supervision, or oriented surface normals, which are not always available in real-world scenarios.

2. **Sensitivity to Data Quality:** Many classical and learning-based methods fail to handle sparse, noisy, or unevenly distributed point clouds, resulting in artifacts, holes, or incomplete reconstructions.

3. **Capturing Fine Geometry:** Complex features such as sharp edges, thin structures, or high-curvature regions are often smoothed out or misrepresented by low-capacity networks or rigid regularization.

4. **Scalability and Efficiency:** Processing high-resolution inputs or producing dense meshes from large-scale scenes can be computationally expensive, limiting practical deployment in real-time or embedded systems.

5. **Lack of Generalization:** Existing reconstruction pipelines are often tuned for specific datasets or sensors, and require significant reconfiguration when applied to new domains or noise distributions.

6. **Lack of Ground Truth Supervision:** Most 3D datasets do not provide clean ground-truth surfaces, making it difficult to apply supervised learning directly.

To address these limitations, this project proposes a self-supervised neural pipeline that:

1. Learns an implicit SDF using a SIREN network without mesh supervision

2. Uses a combination of geometric loss functions Eikonal, Dirichlet, non-Dirichlet, and singular Hessian to regularize the SDF behavior near the surface,

3. Generates synthetic training data through surface-based sampling (centroids), Gaussian perturbations (near-surface), and uniform background sampling (far-field).

4. Produces high-quality meshes using Marching Cubes, validated via Chamfer Distance, F-score, and normal consistency.

1.3 Literature Review

Recent advancements in implicit neural representations have significantly enhanced our ability to model high-frequency 3D geometry. One such development involves the use of sinusoidal activation functions in neural networks. Researchers demonstrated that replacing traditional activation functions with periodic sine functions allows for the accurate representation of fine-grained surface details. This periodicity enables neural networks to better capture intricate geometric variations, particularly useful in image fitting and surface reconstruction tasks. Motivated by this, we adopt such a sinusoidal architecture as the backbone for our reconstruction pipeline [10].

Building on this foundation, other studies have explored the integration of second-order geometric information into learning-based representations. One notable approach involved incorporating Hessian-based loss functions, which leverage second-order derivatives to impose smoothness constraints. These higher-order constraints were shown to be particularly beneficial in preserving surface fidelity in regions of sharp curvature. Drawing from this, we employ a related alignment loss in our work to refine local surface accuracy [11].

The problem of reconstructing CAD-like surfaces from unoriented point clouds has also been addressed using self-supervised learning techniques. A recent method introduced a pipeline that combined shape priors with regularization to recover structured surfaces effectively. While this approach achieved success in domains like industrial design, it was inherently limited in handling natural or organic shapes due to its reliance on handcrafted priors. Our method diverges by focusing on general 3D surface reconstruction without making assumptions about underlying structure, allowing for broader applicability [4].

Another significant contribution in the field was the development of a framework for learning continuous Signed Distance Functions (SDFs) using neural decoders. This framework enabled shape interpolation by learning latent space embeddings from ground-truth meshes. However, it required clean mesh supervision and was less effective in handling sparse or noisy data, especially when surface normals were not available. In contrast, our approach eliminates the dependency on such ground-truth supervision, relying instead on self-supervised learning strategies [8].

Recent efforts have also explored learning directional fields on surfaces, particularly for quadrilateral mesh generation. One such approach focused on encoding cross fields using neural networks to guide mesh construction. While its primary aim was not surface reconstruction, the methodology introduced valuable techniques for representing directional consistency across geometry. Inspired by this, we incorporate an alignment-based constraint in our training procedure to enforce consistency in local surface orientations [3].

Lastly, benchmark datasets focusing on sparse and noisy point clouds have underscored the limitations of many current reconstruction methods. These benchmarks emphasized the critical need for robustness and generalization in learning frameworks. Informed by these findings, our pipeline emphasizes self-supervised training and geometric regularization to better handle challenging inputs encountered in real-world scenarios [5].

Chapter 2

Data

2.1 Data Overview

The dataset utilized in this study consists of 3D models formatted in STL (stereolithography), a popular standard for representing three-dimensional surfaces using triangular facets. Each triangle in the STL file is defined by three vertices and a corresponding unit normal vector, which collectively describe the surface geometry essential for training a neural Signed Distance Function (SDF) model. The STL format was chosen for its widespread use and compatibility with various 3D modeling software.

The main goal is to convert the discrete triangular mesh into a smooth, continuous surface through a self-supervised neural SDF model. To accomplish this, we generate structured point sets from the mesh, which serve as the training data without the need for labeled SDF values. These point sets are grouped into three distinct categories:

- **Surface Points (P):** These points are derived as the centroids of the triangular facets, located directly on the surface where the SDF value should ideally be zero.
- **Near-Surface Points (Ω):** These are created by adding Gaussian noise to the surface points, forming a narrow layer around the surface to capture local geometric details.
- **Far-Field Points (Q):** These points are sampled uniformly across the model's bounding box, representing regions far from the surface to help distinguish between the interior and exterior of the geometry.

These point sets play a crucial role in training the neural SDF model in a self-supervised manner, allowing it to infer the implicit surface geometry from point cloud data. The dataset is designed to handle challenges such as noise, sparsity, and irregular sampling, as outlined in the project's objectives.

2.2 Exploratory Data Analysis

2.2.1 Geometric Analysis of the Mesh

The STL file is imported using the Python library `trimesh`, which provides access to the mesh's vertices and triangular faces. The spatial extent of the model is determined by calculating its bounding box, defined by the minimum and maximum coordinates along each axis ($[x_{\min}, x_{\max}]$, $[y_{\min}, y_{\max}]$, $[z_{\min}, z_{\max}]$). This analysis helps in understanding the model's dimensions and preparing the data for normalization, typically to a range of $[-1, 1]^3$, which aids in the training process.

2.2.2 Calculation of Surface Properties

For each triangle in the mesh, we compute its centroid and normal vector. The centroid of a triangle with vertices $\mathbf{v}_1 = (x_1, y_1, z_1)$, $\mathbf{v}_2 = (x_2, y_2, z_2)$, and $\mathbf{v}_3 = (x_3, y_3, z_3)$ is obtained as the average of the vertex coordinates:

$$\mathbf{c} = \left(\frac{x_1 + x_2 + x_3}{3}, \frac{y_1 + y_2 + y_3}{3}, \frac{z_1 + z_2 + z_3}{3} \right)$$

The normal vector is calculated using the cross product of two edges of the triangle. Given edges $\mathbf{e}_1 = \mathbf{v}_2 - \mathbf{v}_1$ and $\mathbf{e}_2 = \mathbf{v}_3 - \mathbf{v}_1$, the normal $\mathbf{n}$ is:

$$\mathbf{n} = \mathbf{e}_1 \times \mathbf{e}_2 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix}$$

This normal is then normalized to a unit vector:

$$\mathbf{n}_{\text{unit}} = \frac{\mathbf{n}}{\|\mathbf{n}\|}$$

where $\|\mathbf{n}\| = \sqrt{n_x^2 + n_y^2 + n_z^2}$. The centroids (P) indicate surface locations, while the unit normals provide orientation information, which is useful for both visualization and training loss functions, such as the alignment loss, to ensure that the SDF gradients match the surface normals.

2.2.3 Point Sampling Techniques

Two primary sampling methods are employed to construct a comprehensive dataset for training:

- **Near-Surface Points (Ω):** These points are generated by perturbing the centroids with Gaussian noise. For each centroid $\mathbf{c}_i$, a near-surface point $\mathbf{p}_\Omega$ is created as:

  $$\mathbf{p}_\Omega = \mathbf{c}_i + \mathcal{N}(0, \sigma_i)$$

  where $\mathcal{N}(0, \sigma_i)$ represents a Gaussian distribution with a mean of 0 and a standard deviation σ_i. The value of σ_i is determined using the k-Nearest Neighbors (k-NN) algorithm with $k = 50$. For each centroid $\mathbf{c}_i$, the distance to its 50th nearest neighbor among all centroids is calculated and used as σ_i. This adaptive noise ensures that the near-surface points form a thin shell around the surface, reflecting the local mesh density.

- **Far-Field Points (Q):** These points are sampled uniformly within the bounding box, scaled to $[-1, 1]^3$. For a point $\mathbf{q} = (x, y, z)$, each coordinate is drawn from a uniform distribution:

  $$x, y, z \sim \mathcal{U}(-1, 1)$$

  To ensure that Q points represent free space, any points too close to the surface (within a predefined distance from any centroid in P) are removed. This step ensures a clear separation between the surface, near-surface, and far-field regions, which is essential for the SDF to learn the geometry's inside/outside distinction.

2.2.4 Visualization and Analysis

A 3D scatter plot is created to examine the spatial distribution of the surface points (P), near-surface points (Ω), and far-field points (Q). The plot uses different colors to distinguish the point sets: blue for the surface points (P), green for the near-surface points (Ω), and red for the far-field points (Q). The Stanford Bunny model, a standard benchmark in 3D reconstruction, is used for this visualization. The surface points (P) are displayed in blue, outlining the bunny's shape, confirming their placement on the mesh surface. The near-surface points (Ω) appear in green, forming a narrow band around the bunny, which validates the effectiveness of the Gaussian noise sampling with $k = 50$ nearest neighbors in capturing local surface variations. The far-field points (Q) are shown in red, scattered uniformly within the $[-1, 1]^3$ bounding box, surrounding the bunny and representing the free space.

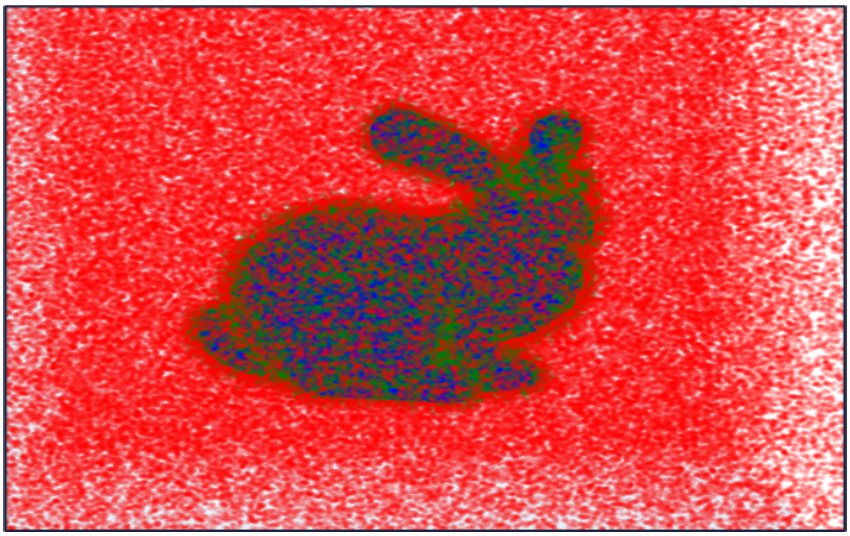


Figure 2.1: 3D scatter plot illustrating the sampled point sets for the Stanford Bunny model. Surface points (P) are shown in blue, near-surface points (Ω) in green and far points (Q) in red.

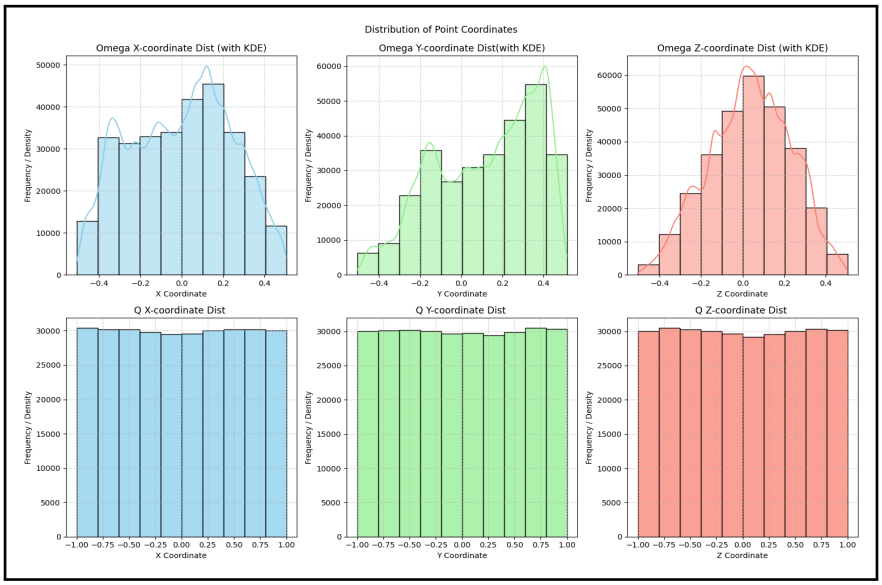


Figure 2.2: Distribution of Point Coordinates: Ω (Normal) and Q (Uniform) Points for Neural SDF Model Validation

This visualization verifies the correct spatial arrangement of the point sets, ensuring that the surface points are on the mesh, the near-surface points form a thin shell, and the far-field points are distributed appropriately in the surrounding space. Additionally, the distributions of distances between Ω points and their corresponding centroids, as well as the spread of Q points, are analyzed using histograms to confirm the expected Gaussian and uniform distributions, respectively. These analyses validate the dataset's readiness for training the neural SDF model.

2.3 Data Preprocessing

The data Preprocessing process ensures that the sampled point sets and normals are ready for neural network training, focusing on consistency, stability, and compatibility. The steps include:

1. **Normalization:** The mesh is repositioned to the origin and scaled to fit within a $[-1, 1]$ bounding box. For a vertex $\mathbf{v} = (x, y, z)$, the normalized coordinates $\mathbf{v}_{\text{norm}} = (x_{\text{norm}}, y_{\text{norm}}, z_{\text{norm}})$ are calculated as:

$$x_{\text{norm}} = \frac{x - x_{\text{center}}}{x_{\max} - x_{\min}}, \quad y_{\text{norm}} = \frac{y - y_{\text{center}}}{y_{\max} - y_{\min}}, \quad z_{\text{norm}} = \frac{z - z_{\text{center}}}{z_{\max} - z_{\min}}$$

   where $(x_{\text{center}}, y_{\text{center}}, z_{\text{center}})$ is the midpoint of the bounding box, and the scaling maps the coordinates to $[-1, 1]$. This step ensures uniform scaling, which is vital for effective neural network training.

2. **Surface Feature Computation:** The centroids and unit normal vectors for each triangle are computed as described earlier, providing the surface points (P) and their orientation data.

3. **Near-Surface Sampling (Ω):** Near-surface points are generated by adding Gaussian noise to the centroids, with the noise level determined by the k-NN algorithm ($k = 50$) to adapt to the local mesh density.

4. **Far-Field Sampling (Q):** Far-field points are sampled uniformly within the $[-1, 1]^3$ bounding box, with a filtering step to exclude points too close to the surface, ensuring they represent free space.

5. **Tensor Conversion:** The point sets (P, Ω, Q) and their normals are converted into PyTorch tensors, enabling efficient processing on GPU/CPU during training.

These preparation steps result in a well-organized dataset that is normalized, adaptive to local geometry, and ready for training the self-supervised neural SDF model. The inclusion of surface, near-surface, and far-field points, along with their normals, ensures that the model can learn a precise and robust continuous surface representation, even from challenging point cloud data.

Chapter 3

Part I:SDF Modeling and Methodology

3.1 Signed Distance Function (SDF) Model

3.1.1 What is an SDF?

A Signed Distance Function (SDF)[10] is a continuous function, $f : \mathbb{R}^3 \to \mathbb{R}$ that represents a surface implicitly as its zero level set. For a point $\mathbf{x} \in \mathbb{R}^3$, the SDF value is defined as:

$$f(\mathbf{x}) = \begin{cases} -d(\mathbf{x}, S) & \text{if } \mathbf{x} \text{ is inside the surface} \\ 0 & \text{if } \mathbf{x} \text{ is on the surface} \\ d(\mathbf{x}, S) & \text{if } \mathbf{x} \text{ is outside the surface} \end{cases}$$

where $d(\mathbf{x}, S)$ is the Euclidean distance from $\mathbf{x}$ to the nearest point on the surface S. The sign indicates whether the point is inside (negative) or outside (positive) the surface, and the gradient $\nabla f(\mathbf{x})$ ideally has a magnitude of 1, making the SDF a true distance function.

SDFs are particularly useful for surface reconstruction because they provide a smooth, continuous representation of the geometry, enabling the use of algorithms like Marching Cubes to extract the surface mesh by finding the isosurface where $f(\mathbf{x}) = 0$.

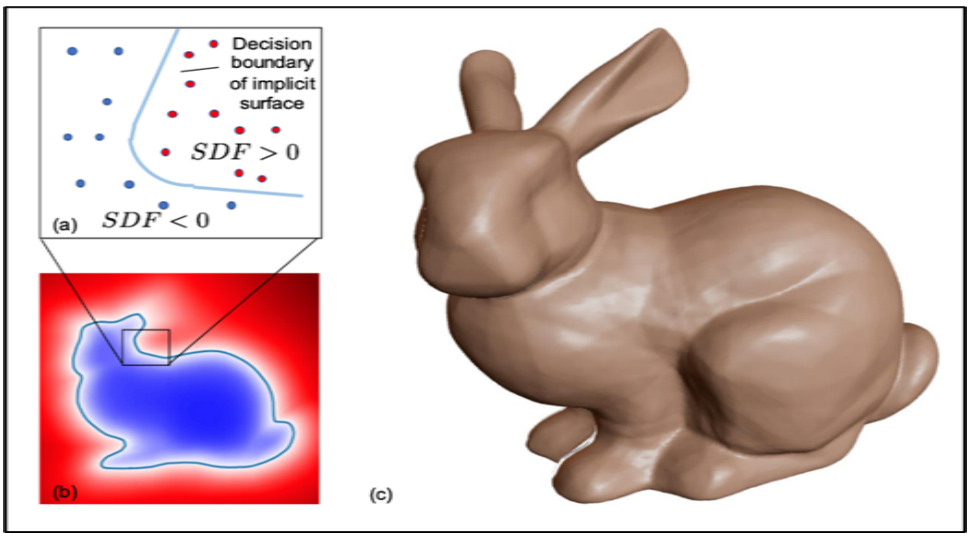


Figure 3.1: Visualization of Signed Distance Function (SDF) for Bunny Model: (a) 2D slice showing points with $SDF < 0$ (blue) and $SDF > 0$ (red) relative to the decision boundary (sky blue); (b) Heatmap of SDF values in a 2D slice (blue: inside, red: outside); (c) 3D reconstructed bunny surface using SDF.[8]

3.1.2 Applications of SDFs

SDFs have a wide range of applications in computer graphics and related fields:

- **Surface Reconstruction:** Converting noisy or sparse point clouds into watertight meshes, crucial for reconstructing accurate anatomical models from medical imaging data.
- **Rendering:** Ray marching techniques use SDFs to render implicit surfaces efficiently, enabling high-quality visualization of 3D medical models.
- **Shape Optimization:** SDFs enable gradient-based optimization of shapes, which is valuable in designing custom medical implants or prosthetics.
- **Collision Detection:** SDFs provide distance information for detecting collisions, useful in robotic surgery systems or interactive medical simulations.
- **3D Printing:** SDFs ensure watertight surfaces, critical for 3D printing patient-specific anatomical models or surgical guides.
- **Medical Image Reconstruction:** Implicit modeling enhances the reconstruction of 3D anatomical structures from 2D medical images, improving diagnostic accuracy and treatment planning.
- **Implicit Modeling for 3D Scene Understanding and Generation:** SDFs facilitate the understanding and generation of complex 3D scenes, supporting applications like virtual reality in medical training and synthetic data generation for AI models.

3.2 Neural Network

3.2.1 Sinusoidal Representation Network (SIREN)

The SDF is modeled using a Sinusoidal Representation Network (SIREN) [10, 9], a neural network architecture designed to capture high-frequency geometric details through sinusoidal activation functions. The SIREN architecture is defined as a multi-layer perceptron (MLP) with sine activations function:

$$\mathbf{h}_0 = \mathbf{x}, \quad \mathbf{h}_i = \sin(\omega_0(\mathbf{W}_i\mathbf{h}_{i-1} + \mathbf{b}_i)), \quad f(\mathbf{x}) = \mathbf{W}_{\text{out}}\mathbf{h}_L + \mathbf{b}_{\text{out}}$$

where:

- $\mathbf{x} \in \mathbb{R}^3$ is the input 3D coordinate.
- $\mathbf{W}_i$ and $\mathbf{b}_i$ are the weights and biases of layer i.
- ω_0 is a frequency parameter (set to 30 in this project).
- L is the number of hidden layers (4 in this case).
- $f(\mathbf{x})$ is the output SDF value.

Each hidden layer has 256 neurons, resulting in a network with 6 layers in total (input, 4 hidden, output).

3.2.2 Advantages of SIREN for SDF Representation

SIRENs are particularly well-suited for SDF modeling due to the following advantages:

- **Implicit Representation:** SIRENs represent surfaces implicitly as the zero level set of the SDF, avoiding the need for explicit mesh storage (e.g., vertices and faces), which is memory-intensive for complex shapes.
- **High-Frequency Detail:** Unlike ReLU-based networks, which struggle with high-frequency signals, the sine activation allows SIRENs to capture fine geometric details, such as sharp edges or intricate patterns [10].
- **Continuity and Smoothness:** The sine function is continuous and infinitely differentiable, ensuring the SDF is smooth, which is desirable for surface reconstruction and rendering.
- **Periodic Activation:** The periodic nature of sine enables the network to model repeating patterns or textures on the surface, which can be challenging for other activation functions.

3.2.3 Why Sine Activation and the frequency Parameter(ω_0)?

The choice of sine activation and the ω_0 parameter is critical for SIREN's performance:

- **High-Frequency Representation:** The sine function can represent high-frequency signals, allowing the network to capture fine details in the SDF, such as small features on the bunny model.
- **Infinitely Differentiable:** Sine is infinitely differentiable, which is essential for computing gradients (∇f) and Hessians ($\mathbf{H}_f$) used in loss functions like the Eikonal and Singular hessain losses.
- **Bounded Output:** The sine function outputs values in $[-1, 1]$, which helps stabilize training by preventing activations from growing unbounded.
- ω_0 **Parameter:** The ω_0 parameter controls the frequency of the sine function. A higher ω_0 (e.g., 30) allows the network to model higher frequencies, capturing finer details, but increases training difficulty. The value of 30 was chosen based on trial-and-error to balance detail and stability.

3.2.4 SIREN Model Summary

The SIREN architecture used in this project is summarized below:

- **Input Layer:** 3 neurons (for 3D coordinates $\mathbf{x} = (x, y, z)$).
- **Hidden Layers:** 4 layers, each with 256 neurons, using sine activation with $\omega_0 = 30$.
- **Output Layer:** 1 neuron (for the SDF value $f(\mathbf{x})$).
- **Total Parameters:** The network has approximately 264449 parameters, calculated as:

$$(3 \times 256 + 256) + (256 \times 256 + 256) \times 3 + (256 \times 1 + 1) = 1024 + (65792) \times 3 + 257 = 264449$$

```
==========================================================================
Layer (type:depth-idx)                             Param #
==========================================================================
SIREN                                              --
├─ModuleList: 1-1                                  --
│    └─SIRENLayer: 2-1                             --
│    │    └─Linear: 3-1                            1,024
│    │    └─SineActivation: 3-2                    --
│    └─SIRENLayer: 2-2                             --
│    │    └─Linear: 3-3                            65,792
│    │    └─SineActivation: 3-4                    --
│    └─SIRENLayer: 2-3                             --
│    │    └─Linear: 3-5                            65,792
│    │    └─SineActivation: 3-6                    --
│    └─SIRENLayer: 2-4                             --
│    │    └─Linear: 3-7                            65,792
│    │    └─SineActivation: 3-8                    --
│    └─SIRENLayer: 2-5                             --
│    │    └─Linear: 3-9                            65,792
│    │    └─SineActivation: 3-10                   --
├─Linear: 1-2                                      257
==========================================================================
Total params: 264,449
Trainable params: 264,449
Non-trainable params: 0
==========================================================================
```

Figure 3.2: SIREN Neural Network Architecture Summary

3.2.5 Network Parameter Initialization

Effective initialization of network parameters is essential for the successful training of deep learning models. This step is critical due to several factors:

1. **Faster Convergence:** Appropriate initialization accelerates the optimization process, enabling the model to converge more quickly during training.
2. **Preventing Gradient Issues:** Incorrect initialization may cause gradients to either vanish or explode during backpropagation, disrupting the learning process.
3. **Enhanced Model Performance:** Proper initialization improves the model's ability to reach an optimal solution in the loss landscape, leading to better overall performance.

initialization method, proposed by Sitzmann et al. [10], effectively manages the frequency response, avoids gradient-related issues, and supports faster convergence. It also enables the network to capture high-frequency details, which is crucial for precise SDF modeling.

1. **First Layer:** Weights are drawn uniformly from the range $\left[-\frac{1}{\text{in\_features}}, \frac{1}{\text{in\_features}}\right]$. This ensures that the initial activations remain within a manageable scale.
2. **Hidden Layers:** Weights are sampled from a uniform distribution $\left[-\frac{\sqrt{6/\text{in\_features}}}{\omega_0}, \frac{\sqrt{6/\text{in\_features}}}{\omega_0}\right]$. This approach regulates the network's frequency response, preventing activations from becoming excessively large.
3. **Output Layer:** Weights are initialized of output layer same as the Hidden layers. This keeps the initial SDF values within a suitable range.

3.3 Loss Functions

To train the SIREN-based implicit neural network to approximate the Signed Distance Function (SDF), a self-supervised learning framework is employed. This framework uses several carefully designed loss functions that integrate geometric and physical constraints derived from the properties of the true SDF. These loss functions not only promote accuracy in surface reconstruction but also help enforce key SDF properties such as differentiability, gradient behavior, and curvature. The total loss is a weighted combination of the individual components described below. [3, 11, 6, 2]

3.3.1 Eikonal Loss (L_{Eik})

This the norm of the gradient of the SDF should be 1 alms term enforces the Eikonal equation, which states thaost everywhere, i.e., $\|\nabla f(\mathbf{x})\| = 1$. It ensures that the function behaves like a true distance field, with uniform rate of change from the surface in all directions. The loss is applied to both surface points P and near-surface points Ω.

Mathematical Expression:

$$L_{Eik} = \frac{1}{|P| + |\Omega|} \int_{P \cup \Omega} |1 - \|\nabla f(\mathbf{x}; \Theta)\|| \, d\mathbf{x}$$

3.3.2 Dirichlet Matching Loss (L_{DM})

This loss imposes the Dirichlet condition that the SDF value should be zero exactly on the surface. For every surface point $\mathbf{p} \in P$, the predicted value $f(\mathbf{p}; \Theta)$ is expected to be as close to zero as possible. This ensures an accurate reconstruction of the surface geometry from the point cloud.

Mathematical Expression:

$$L_{DM} = \frac{1}{|P|} \int_{P} |f(\mathbf{p}; \Theta)| \, d\mathbf{p}$$

3.3.3 Dirichlet Non-Matching Loss (L_{DNM})

While the Dirichlet condition holds for surface points, this term ensures that the SDF values at off-surface points $\mathbf{q} \in Q$ do not vanish. Specifically, this encourages a clear separation between the object's geometry and its surrounding space. An exponential penalty is applied to discourage the predicted SDF from approaching zero at these points.

Mathematical Expression:

$$L_{DNM} = \frac{1}{|Q|} \int_{Q} \exp\left(-\rho \, |f(\mathbf{q}; \Theta)|\right) \, d\mathbf{q}$$

where $\rho = 100$ is a hyperparameter that controls the sharpness of the penalty.

3.3.4 Alignment Loss (L_{AN})

The alignment condition leverages the fact that, near the surface, the Hessian matrix $\mathbf{H}_x$ of the SDF has a zero eigenvalue, and the corresponding eigenvector aligns with the surface normal $\mathbf{n}_x$. This implies that the second-order behavior of the SDF is consistent with the geometry of the surface.

The loss quantifies misalignment by penalizing the deviation of the Hessian-vector product from zero:

$$\mathbf{H}_x \cdot \mathbf{n}_x = 0$$

Mathematical Expression:

$$L_{AN} = \frac{1}{|P|} \int_P |\mathbf{H}_p \cdot \mathbf{n}_p| \, d\mathbf{p}$$

3.3.5 Neumann Loss ($L_{Neumann}$)

This loss enforces consistency between the predicted gradient direction of the SDF and the known surface normals. It ensures that the unit gradient $\frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}$ is aligned with the normal $\mathbf{n}(\mathbf{x})$ at the point $\mathbf{x} \in P$. Deviations from perfect alignment (dot product less than 1) are penalized.

Mathematical Expression:

$$L_{Neumann} = \int_P \left(1 - \left\langle \frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}, \mathbf{n}(\mathbf{x}) \right\rangle\right) d\mathbf{x}$$

3.3.6 Singular Hessian Loss ($L_{SingularH}$)

The zero level set of the SDF—i.e., the surface—is often non-differentiable globally but differentiable in a narrow shell around the surface (Ω). In this region, the Hessian matrix $\mathbf{H}_f(\mathbf{x})$ is expected to be singular. This is due to the fact that the Hessian has a zero eigenvalue with the gradient of the SDF as its eigenvector, and this vector aligns with the surface normal.

Thus, the determinant of the Hessian vanishes in the differentiable zone:

$$\det\left(\mathbf{H}_f(\mathbf{x})\right) = 0$$

This term regularizes the learning process to preserve sharp features and high-curvature regions by encouraging local Hessian singularity.

Mathematical Expression:

$$L_{SingularH} = \int_{Q_{near}} \left|\det\left(\mathbf{H}_f(\mathbf{x})\right)\right| \, d\mathbf{x}$$

3.4 Neural Network Training Process

This section describes the training framework based on SIREN for learning the Signed Distance Function (SDF). The approach integrates foundational techniques such as total loss computation, Hessian loss annealing, and backpropagation, along with advanced optimization methods including mini-batch gradient descent, cosine learning rate scheduling, Kendall's dynamic loss weighting, and gradient clipping. Collectively, these strategies promote training stability and improve the accuracy of geometry-aware SDF learning.

3.4.1 Total Losses

The overall training objective is formulated as a weighted sum of several loss terms, each designed to enforce different physical or geometric constraints on the predicted SDF field. The total loss function is defined as: [3, 11]

$$L = \lambda_{\text{Eik}} L_{\text{Eik}} + \lambda_{\text{DM}} L_{\text{DM}} + \lambda_{\text{DNM}} L_{\text{DNM}} + \tau \lambda_{\text{H}} L_{\text{SingularH}}$$

Where:

1. $\lambda_{\mathbf{Eik}} = 50$: Weight for the Eikonal loss, enforcing unit-norm gradient magnitudes to preserve signed distance properties.
2. $\lambda_{\mathbf{DM}} = 7000$: Weight for the Dirichlet Matching loss, minimizing the SDF values on surface points.
3. $\lambda_{\mathbf{DNM}} = 60$: Weight for the Dirichlet Non-Matching loss, driving off-surface points away from zero.
4. $\lambda_{\mathbf{H}} = 3$: Weight for the Hessian-based regularization, encouraging local geometric smoothness.
5. τ: An annealing factor to gradually reduce the impact of the Hessian loss during training.

3.4.2 Annealing of the Hessian Loss (τ)

The influence of the Hessian loss is modulated over the course of training using an annealing factor τ, [3]defined as:

1. $\tau = 1$ during the initial 20% of training iterations,
2. Linearly decays to 3×10^{-4} between 20% and 40% of the training span,
3. Gradually reduces to 0 over the remaining 60% of the training schedule.

This schedule ensures that curvature constraints guide early learning while enabling later-stage specialization on surface-level accuracy.

3.4.3 Backpropagation and Gradient Computation

Training is conducted via backpropagation, which computes gradients of the loss function with respect to model parameters (weights and biases) using the chain rule of differentiation. The updates for each parameter $\theta \in \{W, b\}$ at iteration t follow:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\partial L}{\partial \theta}$$

Where:

- η is the learning rate,
- $\frac{\partial L}{\partial \theta}$ is computed using automatic differentiation,
- Higher-order derivatives are required for certain losses (e.g., Hessian loss), which are computed via PyTorch's higher-order gradient support.

This process is repeated iteratively across mini-batches, adjusting the model to minimize the total loss. The Adam optimizer [3] was used with a default base learning rate of 1×10^{-4}, offering adaptive moment estimation to accelerate convergence and stabilize updates.

3.4.4 Mini-Batch Gradient Descent

To balance memory efficiency with convergence speed, mini-batch gradient descent was employed. The total dataset consisted of 200,000 samples, from which a subset of 100,000 data points was randomly selected for training. This subset was divided into mini-batches of 10,000 samples each.

Training over this subset was performed in 2,000 iterations per epoch, allowing faster convergence while ensuring diverse coverage of the input space across each training cycle. This strategy reduced computational load without sacrificing generalization capability.

3.4.5 Learning Rate Decay (CosineAnnealingLR)

To prevent stagnation and overfitting, the learning rate was dynamically adjusted using a cosine annealing schedule. The learning rate η_t at epoch t is given by:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\max}}\pi\right)\right)$$

where:

- $\eta_{\min} = 1 \times 10^{-4}$,
- $\eta_{\max} = 1 \times 10^{-3}$,
- T_{cur} is the current epoch, and
- $T_{\max} = 20,000$ is the total training duration.

This schedule allows for rapid learning in the initial epochs and fine-tuned adjustments later, reducing the risk of overshooting during optimization.

3.4.6 Dynamic Loss Weighting (Kendall's Method)

To mitigate the need for extensive manual tuning of loss weights, Kendall et al.'s uncertainty-based dynamic loss weighting method was incorporated. This method treats the uncertainty σ_i of each loss term L_i as a learnable parameter and adjusts its influence dynamically:

$$L_{\text{total}} = \sum_{i=1}^{N} \left(\frac{\lambda_i}{2\sigma_i^2} L_i + \log \sigma_i\right)$$

Where:

- σ_i is the learnable standard deviation (uncertainty) associated with task i,
- The first term scales the loss by task uncertainty,
- The second term regularizes the uncertainty parameters to avoid collapse.

This allows the model to prioritize harder tasks and adapt loss contributions over time, improving both convergence and stability.

3.4.7 Gradient Clipping

Due to the use of sine-based activation functions and higher-order derivatives, the gradients may become unstable. To avoid gradient explosion, gradient clipping was applied, constraining the gradient norm to a maximum of 1.0:

$$\text{if } \|\nabla_\theta L\| > 1, \quad \nabla_\theta L \leftarrow \frac{\nabla_\theta L}{\|\nabla_\theta L\|}$$

This mechanism ensures stable training, particularly important in deeper architectures such as SIREN.

3.4.8 Empirical Hyperparameter Optimization Strategy

Several training hyperparameters were optimized through systematic experimentation:

1. **Loss Weights:** The weights $\lambda_{\text{Eik}}, \lambda_{\text{DM}}, \lambda_{\text{DNM}}, \lambda_{\text{H}}$ were initially tuned manually. Multiple configurations were evaluated to balance surface constraint enforcement, gradient regularity, and curvature minimization.

2. **Epochs:** Training duration was tuned over 7,000, 10,000, 15,000, and 20,000 epochs. A final value of 20,000 was selected, as it consistently delivered superior convergence and reconstruction quality.

3. **Learning Rate:** Initial learning rates of 10^{-2}, 10^{-3}, and 10^{-4} were tested. A starting learning rate of 10^{-3} paired with cosine annealing was found to be optimal.

4. **SIREN Frequency Factor (ω_0):** The initial frequency of the sine activation was tuned within the range $\omega_0 \in [10, 50]$. A value of $\omega_0 = 30$ was selected as it provided a good trade-off between expressivity for high-frequency geometry and training stability.

3.4.9 Implementation Details

All experiments were conducted on Google Colab using an NVIDIA Tesla T4 GPU. Training was performed using the PyTorch deep learning framework. A total of 20,000 training iterations were executed using the Adam optimizer with a default learning rate of 1×10^{-4}, consistent with the initial settings of the cosine annealing schedule.

Chapter 4

Part II: Surface Reconstruction using Marching Cubes

4.1 Overview of the Marching Cubes Algorithm

Once the neural implicit representation of the Signed Distance Function (SDF) has been successfully trained using the SIREN architecture, the subsequent goal is to extract a high-fidelity 3D surface from the continuous SDF predictions. This involves identifying the *zero level set*, which is defined as the set of all points $\mathbf{x} \in \mathbb{R}^3$ such that $f(\mathbf{x}) = 0$. Geometrically, this represents the interface between the object and the surrounding space.

To reconstruct this surface from the volumetric field, we employ the widely adopted **Marching Cubes** (MC) algorithm, first introduced by Lorensen and Cline in 1987 [7]. The MC algorithm transforms a scalar field (e.g., predicted SDF values) into a polygonal mesh by discretizing the domain and performing localized triangulations based on isosurface crossings. Its simplicity, speed, and adaptability have made it the de facto standard in implicit surface reconstruction.

This chapter details the underlying principles of the Marching Cubes algorithm, the mathematical basis for vertex interpolation, and how it integrates with SIREN outputs to generate high-resolution surface meshes.

4.2 Fundamentals of Marching Cubes Algorithm

The Marching Cubes algorithm proceeds by discretizing the continuous 3D domain into a structured grid of cubes (voxels) and analyzing the scalar field values at each vertex to approximate the surface geometry. The following steps summarize the core working procedure:

1. **3D Grid Creation:** The spatial domain of the object is enclosed within a normalized bounding box, typically $[-1, 1]^3$, and discretized into a uniform 3D grid. For this work, a resolution of $256 \times 256 \times 256$ was chosen, yielding a total of $256^3 = 16,777,216$ cubes. Each cube consists of 8 corner vertices.

2. **SDF Evaluation at Grid Vertices:** The trained SIREN network is used to compute the SDF values at each vertex of the grid. Due to the large number of evaluations required, this step is parallelized and processed in batches to optimize GPU memory usage.

3. **Cube Classification:** For each cube, the 8 vertices are examined to determine whether they lie inside ($f(\mathbf{x}) < 0$) or outside ($f(\mathbf{x}) > 0$) the surface. This binary classification yields $2^8 = 256$ possible configurations, with each configuration representing a unique way the surface may intersect the voxel.

4. **Lookup Table for Triangulation:** A precomputed lookup table is used to map each of the 256 cube configurations to a specific set of triangles. These triangulations represent how the surface intersects and passes through the voxel. The lookup table ensures consistent and efficient triangulation across the grid by exploiting geometric symmetries.

5. **Vertex Interpolation on Voxel Edges:** To accurately approximate the surface intersection, the algorithm linearly interpolates the position along edges of the cube where the SDF changes sign. The interpolated point is computed as:

$$\mathbf{v}_{\text{intersect}} = \mathbf{v}_1 + \frac{|f(\mathbf{v}_1)|}{|f(\mathbf{v}_1)| + |f(\mathbf{v}_2)|}(\mathbf{v}_2 - \mathbf{v}_1)$$

   where $\mathbf{v}_1$ and $\mathbf{v}_2$ are the endpoints of the edge, and $f(\mathbf{v}_1), f(\mathbf{v}_2)$ are the corresponding SDF values.

6. **Triangle Mesh Generation:** Using the interpolated vertices and the triangulation pattern from the lookup table, the algorithm generates one or more triangles for each cube. Collectively, these triangles approximate the zero-level surface of the SDF field and result in a polygonal mesh representation of the object.

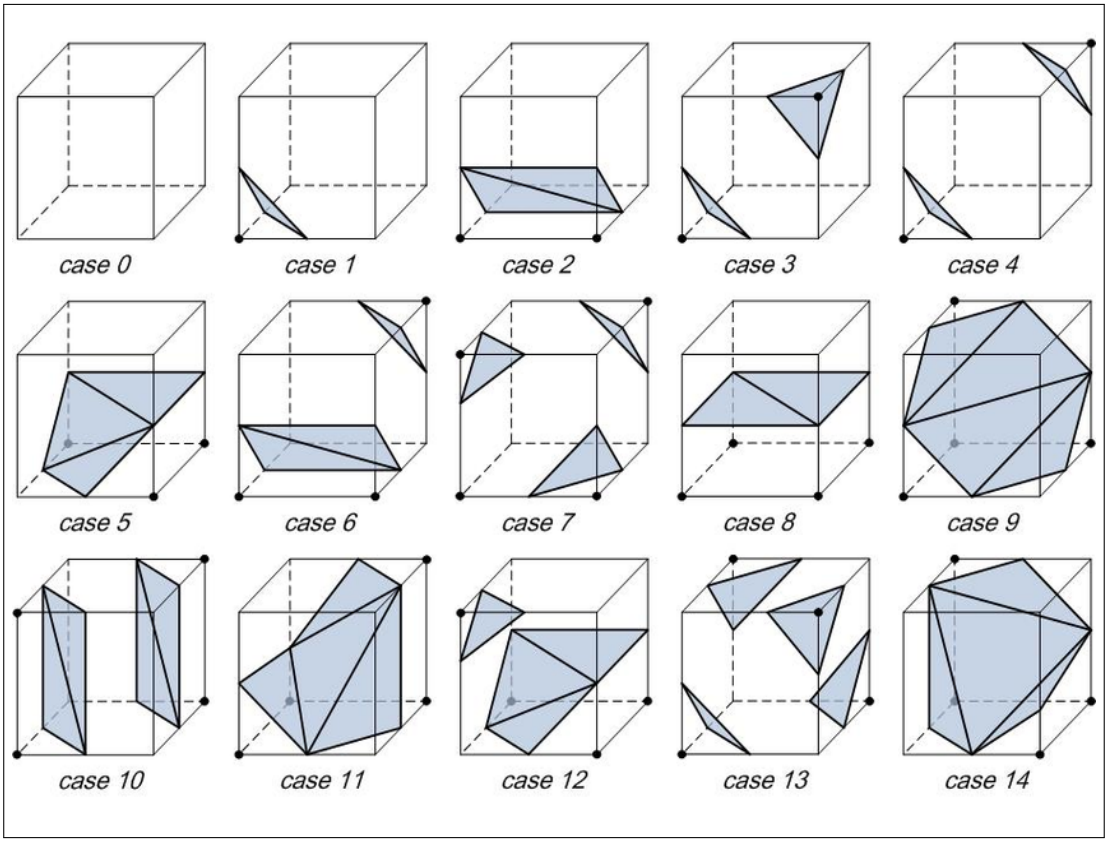


Figure 4.1: Representative 15 Cases in the Marching Cubes Algorithm for Surface Extraction

Figure Explanation: The figure shows a subset of the 256 unique configurations encountered by the Marching Cubes algorithm. Each configuration represents a different binary state of the cube vertices (black dots denote $f(\mathbf{x}) < 0$). The triangles within each cube vary based on the vertex configuration, demonstrating the algorithm's ability to adapt to local geometric variations. [1]

4.3 Implementation Details and Optimization

To ensure the Marching Cubes integration is computationally efficient and produces high-quality meshes, the following implementation strategies were adopted:

- **Grid Resolution Selection:** A resolution of 256^3 was found to provide a good trade-off between reconstruction quality and computational cost. Higher resolutions (e.g., 512^3) offer more detail but come at the expense of increased memory and runtime.
- **Batched SDF Evaluation:** To manage GPU memory, SDF evaluations were conducted in batches of approximately one million grid points. This approach enables parallelism while avoiding memory overflows.
- **Post-Processing of Meshes:** The generated triangle meshes were refined to remove degenerate elements (e.g., triangles with zero area or colinear vertices). A final manifold check was applied to ensure watertightness.
- **Mesh Export and Visualization:** The final surface mesh was exported in standard 3D formats (e.g., STL, OBJ) for further analysis, visualization, and 3D printing. Tools such as MeshLab and Blender were used for mesh inspection and validation.

4.4 Performance Evaluation of Surface Reconstruction

The reconstructed surfaces demonstrated a high degree of fidelity to the underlying 3D geometry. The combination of accurate SDF modeling using SIREN and the robust topology handling of the Marching Cubes algorithm resulted in meshes that preserved intricate features such as sharp edges, small-scale structures, and fine holes. Qualitative inspection revealed that even subtle geometric nuances (e.g., the curvature of bunny ears or sharp corners of CAD models) were faithfully captured.

Furthermore, the Marching Cubes algorithm provided topologically consistent triangulations, enabling seamless downstream applications in rendering and physical simulation. The surface quality was directly tied to the precision of the learned SDF, highlighting the interdependence between the implicit representation and the surface reconstruction pipeline.

4.5 Concluding Insights on Marching Cubes Integration

In conclusion, the Marching Cubes algorithm serves as an essential bridge between continuous implicit SDF representations and tangible 3D surface models. By sampling the trained SIREN network over a dense grid and applying a carefully optimized triangulation process, high-quality meshes can be generated with practical utility. The integration of batch-wise inference, adaptive triangulation, and mesh refinement makes the reconstruction pipeline scalable and reliable across a variety of 3D shapes and resolutions.

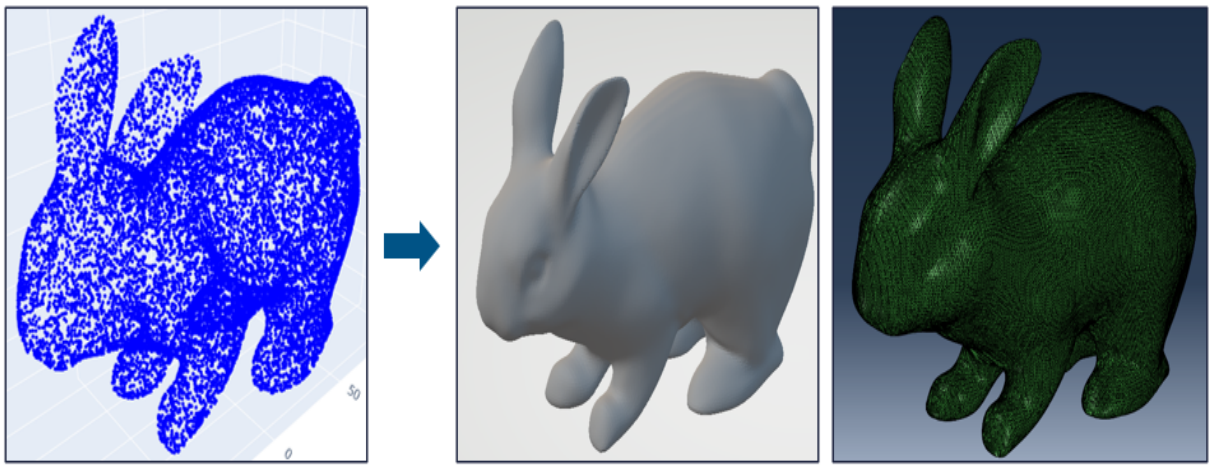
This approach not only enables applications in visualization and 3D printing but also sets the foundation for more advanced modeling techniques, such as neural rendering, deformation tracking, and shape optimization.

Chapter 5

Results

5.1 Reconstructed Shapes Results

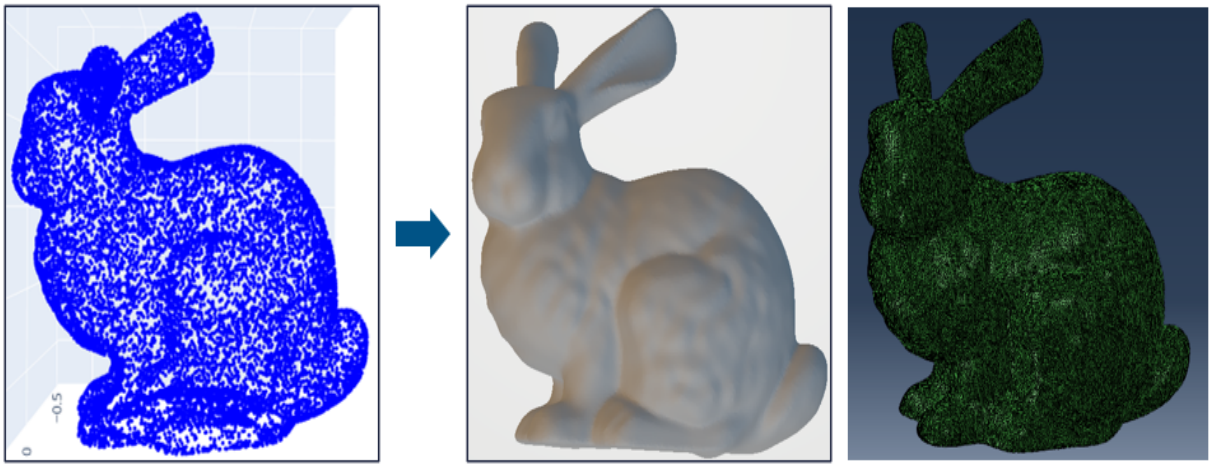
This section presents the results of surface reconstruction for various 3D shapes using the Signed Distance Function (SDF)-based neural network and the Marching Cubes algorithm. Each model is visualized as a point cloud **(a)**, its corresponding reconstructed surface **(b)**, and the final triangulated mesh **(c)**. The models include a mix of simple geometric objects and complex anatomical or mechanical structures: Bunny, Sphere, Cylinder, Explicit Lug, Elephant, Petal, Sacrum Bone, and Cervical Vertebrae.



(a) Point cloud **(b)** Reconstructed surface **(c)** Reconstructed mesh

Figure 5.1: Bunny model: (a) point cloud, (b) reconstructed surface, (c) reconstructed trimesh

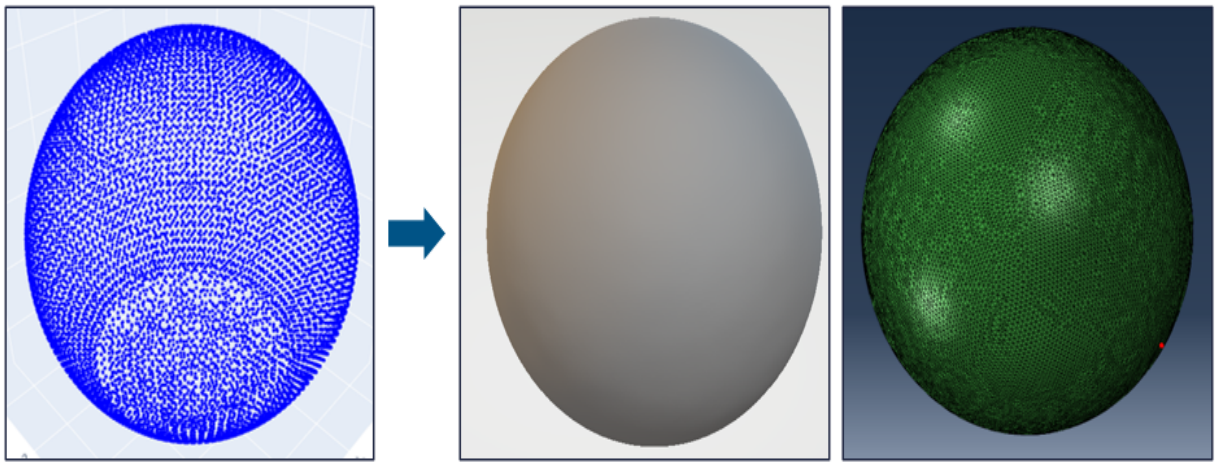
Explanation: The point cloud captures the bunny's organic shape with sparse points, highlighting features like the ears and body contours. The intermediate surface reconstruction in (b) forms a continuous, smooth shape learned from the implicit SDF representation, bridging the gap between raw points and mesh. The reconstructed mesh, generated using the SDF and Marching Cubes algorithm, produces a smooth, watertight surface that preserves the bunny's intricate details, demonstrating the model's ability to handle complex geometries.



(a) Point cloud **(b)** Reconstructed surface **(c)** Reconstructed mesh

Figure 5.2: Bunny model: (a) point cloud, (b) reconstructed surface, (c) reconstructed trimesh

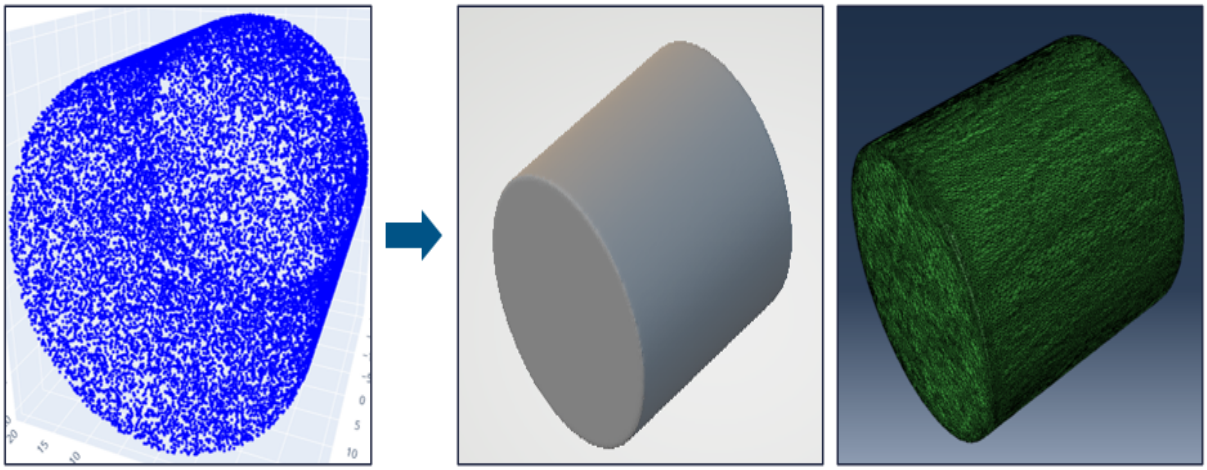
Explanation: The initial point cloud (a) illustrates the bunny's structure through a dense set of spatial points, capturing key features like the ears, limbs, and body curvature. The intermediate surface (b), reconstructed from the SDF, exhibits slight roughness that aligns with the bunny's organic form rather than oversmoothing, the model effectively retains natural surface variations. Finally, the mesh in (c), generated using the Marching Cubes algorithm, results in a watertight 3D surface while preserving the original geometry's intricate and irregular details, showcasing the method's suitability for organic object reconstruction.



(a) Point cloud **(b)** Reconstructed surface **(c)** Reconstructed mesh

Figure 5.3: Sphere model: (a) point cloud, (b) reconstructed surface, (c) reconstructed trimesh

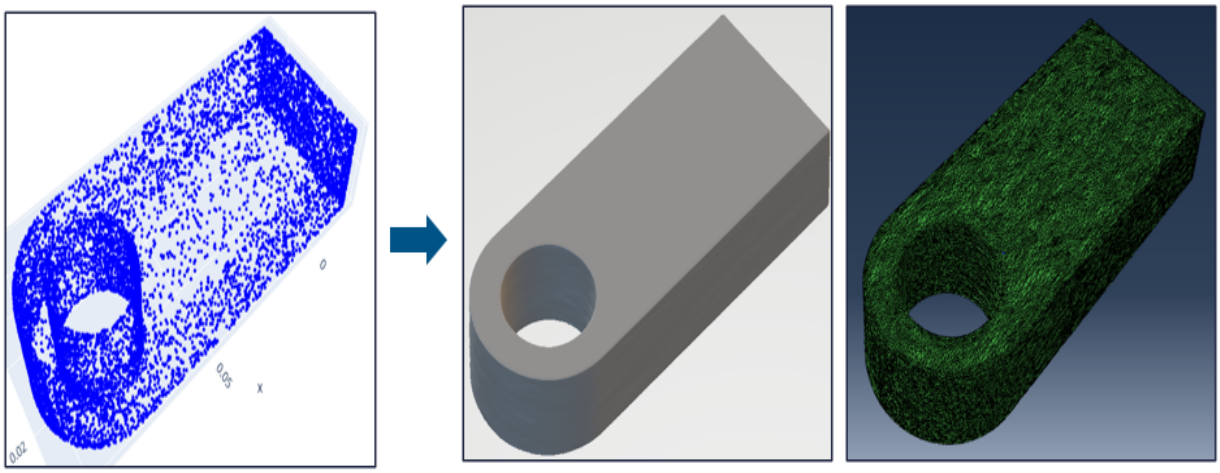
Explanation: The point cloud of the sphere consists of uniformly distributed points, representing its simple, symmetric geometry. The reconstructed mesh forms a perfectly smooth and continuous spherical surface, validating the model's precision in reconstructing basic geometric shapes with high fidelity.



(a) Point cloud **(b)** Reconstructed surface **(c)** Reconstructed mesh

Figure 5.4: Cylinder model: (a) point cloud, (b) reconstructed surface, (c) reconstructed trimesh

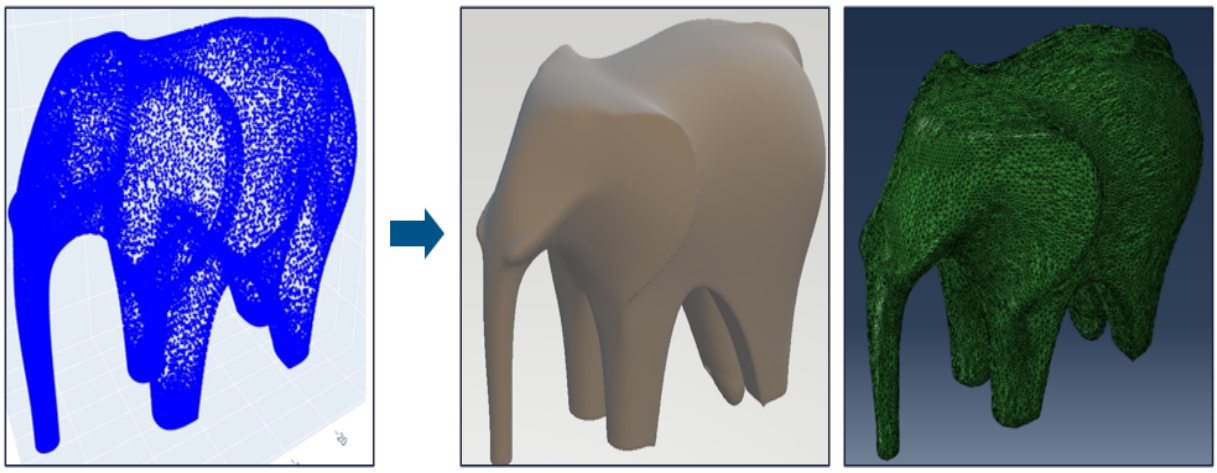
Explanation: The point cloud of the cylinder shows a sparse distribution of points along its curved surface and flat ends. The reconstructed mesh accurately captures the cylindrical shape with smooth, continuous surfaces, confirming the model's capability to handle geometric shapes with flat and curved features.



(a) Point cloud **(b)** Reconstructed surface **(c)** Reconstructed mesh

Figure 5.5: explict lug model: (a) point cloud, (b) reconstructed surface, (c) reconstructed trimesh

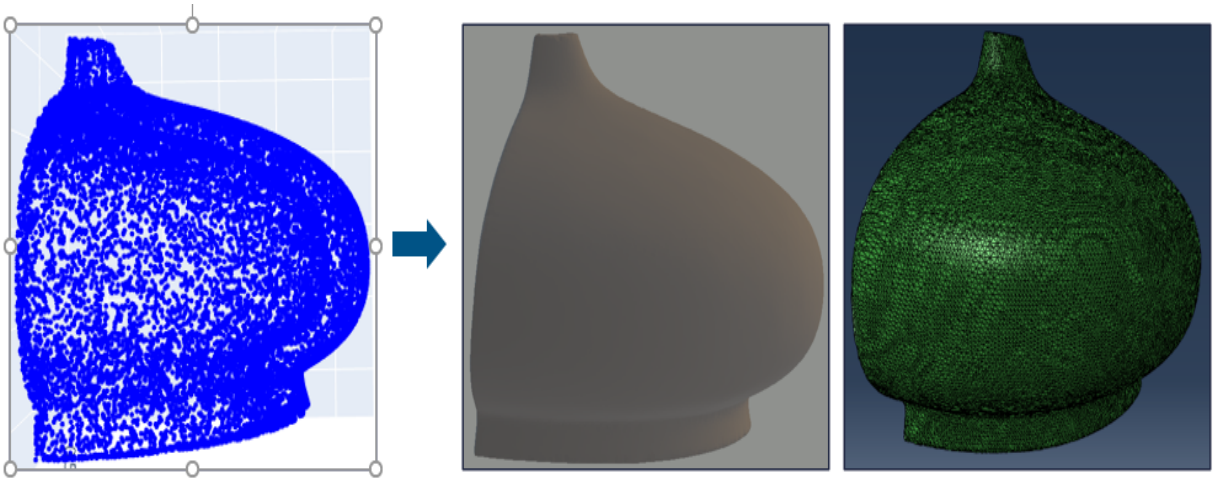
Explanation: The point cloud of the explicit lug, a mechanical component, outlines its functional shape, including the fastening hole and attachment surfaces. The reconstructed mesh produces a clean, watertight model, preserving the lug's structural details, which is crucial for engineering applications like 3D printing or simulation.



(a) Point cloud **(b)** Reconstructed surface **(c)** Reconstructed mesh

Figure 5.6: Elephant model: (a) point cloud, (b) reconstructed surface, (c) reconstructed mesh

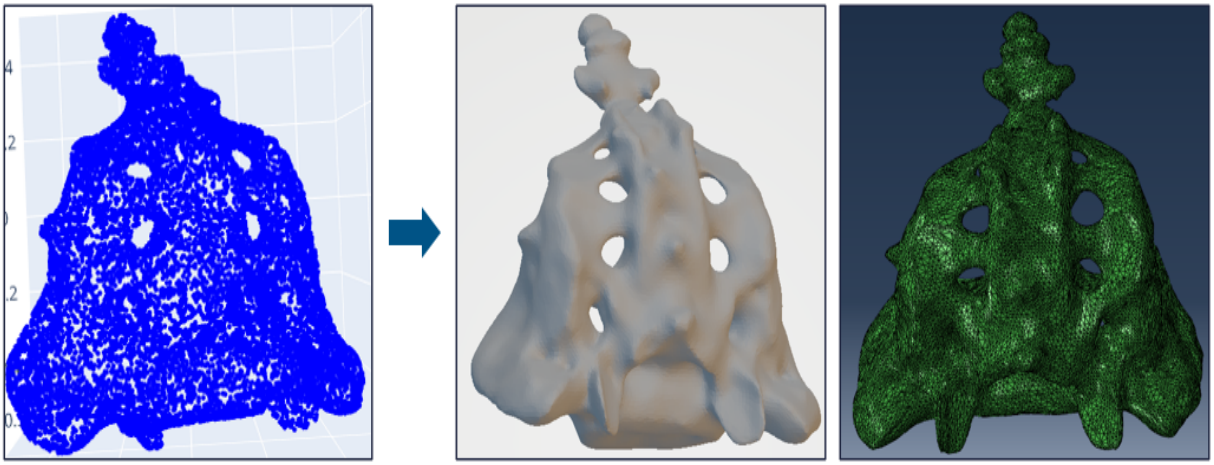
Explanation: The point cloud of the elephant highlights its complex organic structure, with points defining the trunk, ears, and legs. The reconstructed mesh forms a smooth, detailed surface that retains these features, showcasing the model's effectiveness in reconstructing intricate, non-uniform shapes.



(a) Point cloud **(b)** Reconstructed surface **(c)** Reconstructed mesh

Figure 5.7: petal model: (a) point cloud, (b) reconstructed surface, (c) reconstructed trimesh

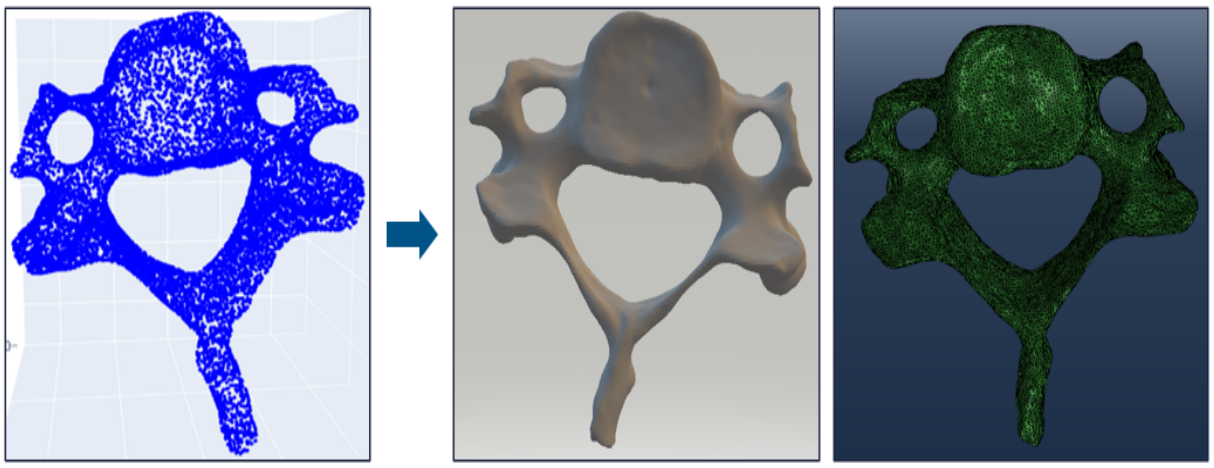
Explanation: The point cloud of the petal captures its delicate, thin structure with sparse points along its surface. The reconstructed mesh generates a smooth, continuous petal shape, demonstrating the model's ability to handle fine, lightweight structures without losing detail.



(a) Point cloud **(b)** Reconstructed surface **(c)** Reconstructed mesh

Figure 5.8: Sacrum Bone model: (a) point cloud, (b) reconstructed surface, (c) reconstructed trimesh

Explanation: The point cloud of the sacrum bone, an anatomical structure, defines its irregular, curved surfaces with sparse points. The reconstructed mesh accurately represents the bone's complex geometry, making it suitable for medical applications like surgical planning or 3D visualization.



(a) Point cloud **(b)** Reconstructed surface **(c)** Reconstructed mesh

Figure 5.9: Carvical vertebrae model: (a) point cloud, (b) reconstructed surface, (c) reconstructed trimesh

Explanation: The cervical vertebrae model begins as a sparse point cloud capturing its complex anatomical geometry, including the vertebral body and processes. The intermediate surface reconstruction exhibits natural surface roughness and morphological variations characteristic of real human bone. The final mesh yields a watertight, high-fidelity representation that retains features critical for medical imaging and diagnostic analysis.

5.2 Quantitative Results

The evaluation is conducted on various shapes: bunny, sphere, cylinder, explicit lug, elephant, petal, sacrum bone, and cervical vertebrae. These shapes vary in complexity, allowing for a comprehensive assessment of the reconstruction method across different geometric challenges. Additionally, a detailed comparison is performed for the various shapes of models, evaluating the generated (red) centroids against the original (blue) centroids to assess point-level accuracy.

5.2.1 Evaluation Metrics

To quantitatively assess the quality of 3D surface reconstruction, we utilize three widely adopted evaluation metrics [11]: Chamfer Distance (CD), F-Score, and Normal Consistency (NC). These metrics evaluate geometric alignment, completeness, and normal orientation accuracy between the predicted surface point cloud P_1 and the ground-truth point cloud P_2.

1. **Chamfer Distance (CD):** The Chamfer Distance measures the average squared distance from each point in one set to its nearest neighbor in the other set:

$$\text{CD}(P_1, P_2) = \frac{1}{2|P_1|} \sum_{p_1 \in P_1} \min_{p_2 \in P_2} d(p_1, p_2) + \frac{1}{2|P_2|} \sum_{p_2 \in P_2} \min_{p_1 \in P_1} d(p_2, p_1)$$

   lower CD value indicate a closer geometric match amid predicted and ground-truth surface.

2. **F-Score:** The F-Score balances precision and recall to evaluate both accuracy and coverage of the reconstruction. Using a threshold t, precision and recall are defined as:

$$\text{Recall}(t, P_1, P_2) = \frac{|\{p_1 \in P_1 \mid \min_{p_2 \in P_2} d(p_1, p_2) < t\}|}{|P_1|}$$

$$\text{Precision}(t, P_1, P_2) = \frac{|\{p_2 \in P_2 \mid \min_{p_1 \in P_1} d(p_2, p_1) < t\}|}{|P_2|}$$

$$\text{F-Score}(t, P_1, P_2) = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$$

   A higher F-Score (closer to 1) indicates better reconstruction performance.

3. **Normal Consistency (NC):** This metric evaluates alignment of surface normals between corresponding points:

$$\text{NC}(P_1, P_2) = \frac{1}{2|P_1|} \sum_{p_1 \in P_1} \mathbf{n}_{p_1} \cdot \mathbf{n}_{\text{closest}(p_1, P_2)} + \frac{1}{2|P_2|} \sum_{p_2 \in P_2} \mathbf{n}_{p_2} \cdot \mathbf{n}_{\text{closest}(p_2, P_1)}$$

   n_p denote the surface normal at point p.Higher NC (near 1) shows better normal alignment.

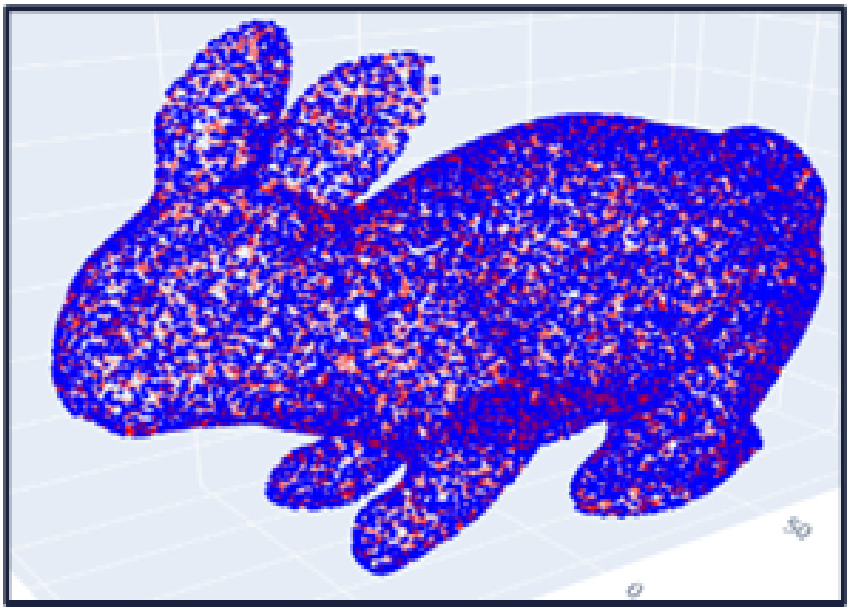
5.2.2 Metric Summary Table

Metric	Meaning	Ideal	Better When
Chamfer Dist.	Surface closeness	0	Lower
F-Score	Precision-recall	$[0, 1]$	Higher
Normal Cons.	Normal match	$[0, 1]$	Higher

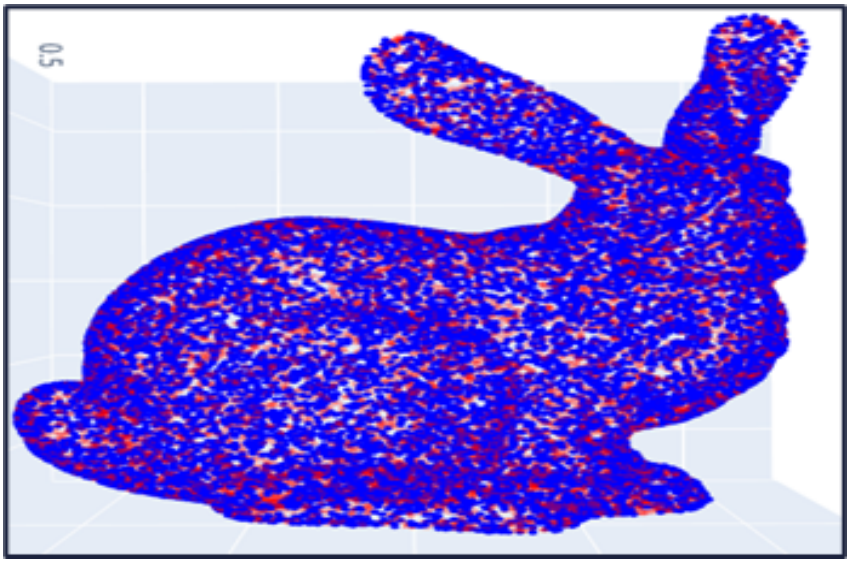
Table 5.1: Summary of metrics for 3D reconstruction evaluation.

5.2.3 Quantitative Evaluation Results of All Models

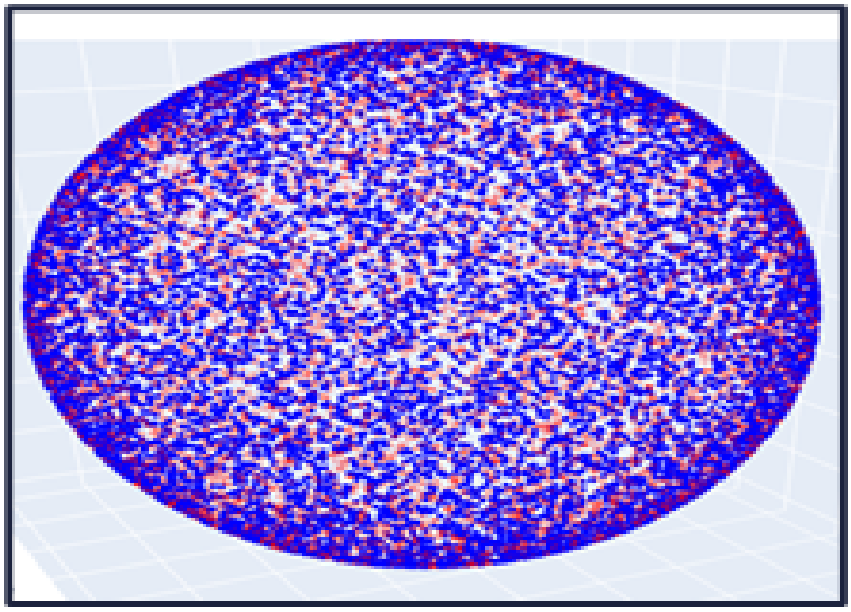
This section evaluates the reconstructed surfaces of different 3D models using Chamfer Distance (CD), F-Score, and Normal Consistency (NC). For each model, **15,000** surface points were randomly sampled from both the original (blue) and reconstructed (red) meshes. All models were scaled to fit within the $\mathbf{[0.5, 0.5]^3}$ range. The Chamfer Distance values were multiplied by **1000** to enhance readability, and the F-Score was computed with a fixed threshold of **0.05** for both precision and recall. These metrics assess the reconstruction's accuracy, completeness, and surface normal consistency.



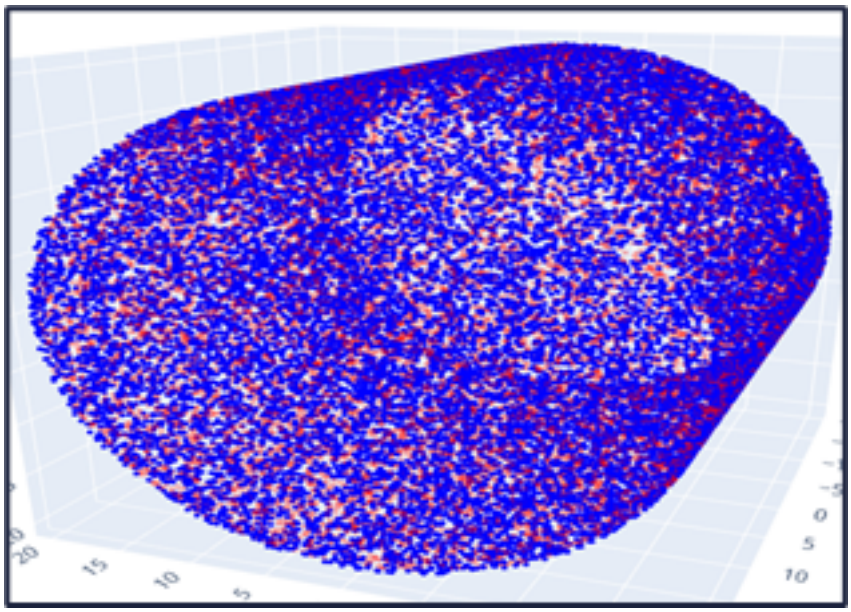
Bunny: CD = 3.2342, F-Score = 0.9893, NC = 0.9921. High F-score and NC with high CD closed to 1 indicate accurate and smooth surface reconstruction.



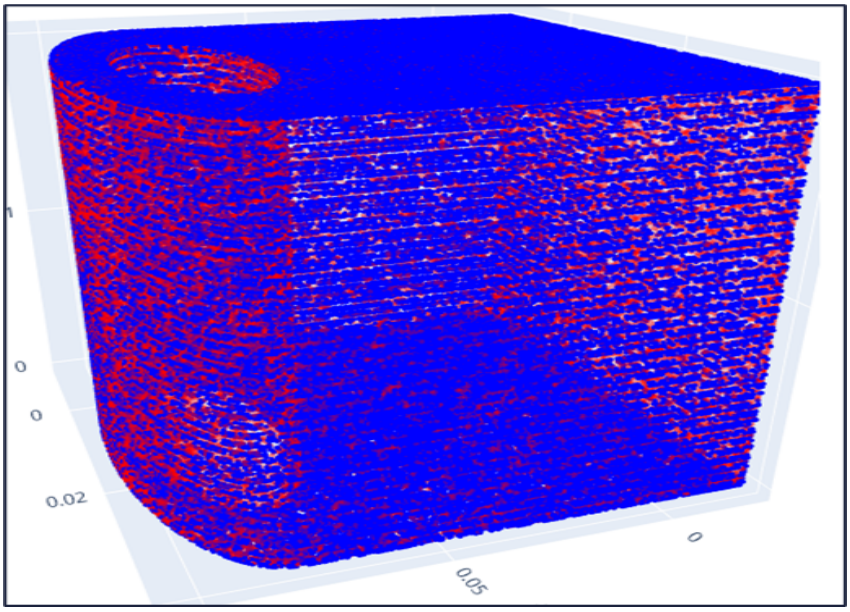
Bunny: CD = 4.8543, F-Score = 0.9321, NC = 0.9717. High F-score and NC with good CD indicate accurate and smooth surface reconstruction.



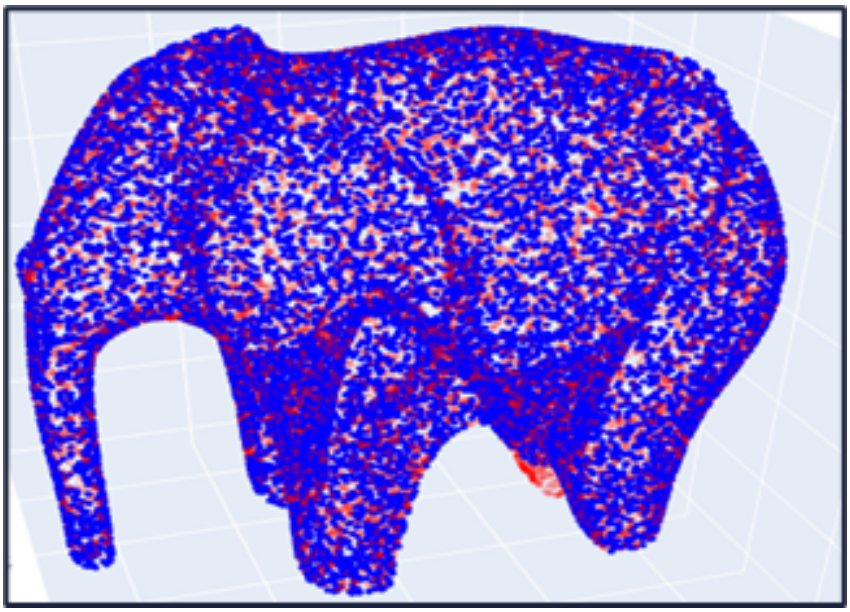
Sphere: CD = 1.2203, F-Score = 0.9912, NC = 0.9924. Slightly higher CD but excellent F-Score and NC confirm solid spherical shape recovery.



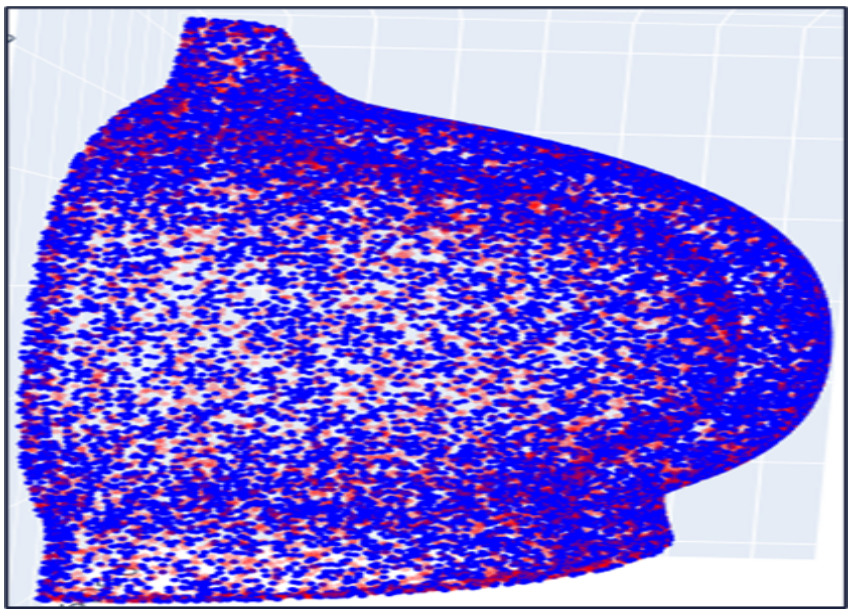
Cylinder: CD = 2.0327, F-Score = 0.9879, NC = 0.9884. Near-perfect geometry with low CD and high NC, ensuring precise surface normals and structure.



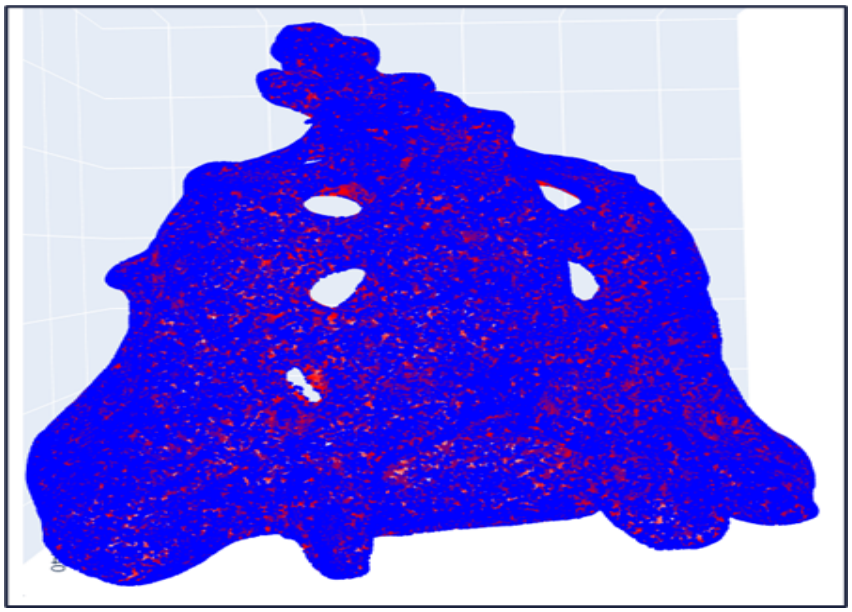
Explicit Lug: CD =4.897, F-Score =0.94, NC = 0.9927.Outstanding performance with perfect F-Score and low CD, capturing detailed mechanical features accurately.



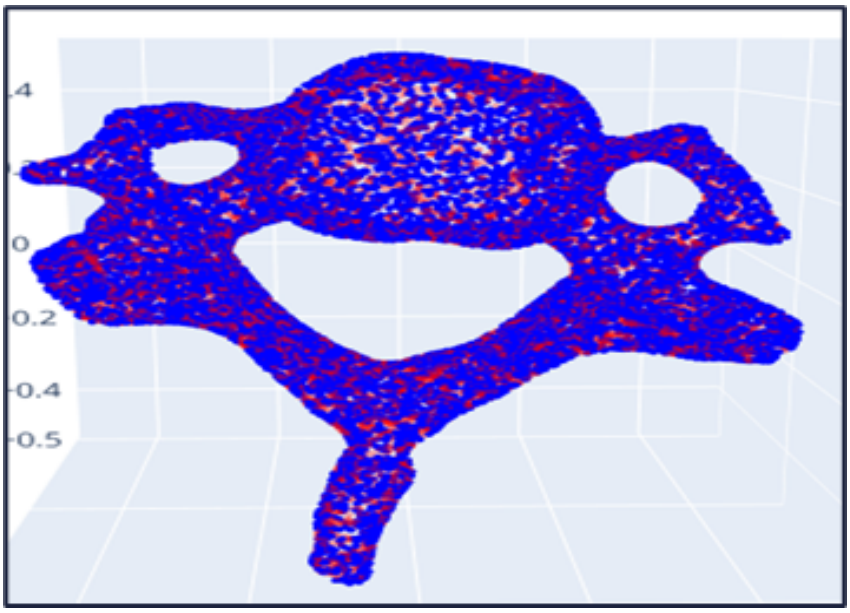
Elephant: CD = 3.777, F-Score = 0.9344, NC = 0.9717. Slightly higher CD and lower NC but strong F-Score reflects good completeness of the organic shape.



Petal: CD = 4.9925, F-Score = 0.9851, NC = 0.9911
Perfect F-Score and low CD confirm high-quality reconstruction of thin, curved structure.



Sacrum Bone: CD = 4.4596, F-Score = 0.9129, NC = 0.9795. Consistently accurate CD and F-Score; NC suggests minor orientation deviations.



Cervical Vertebrae: CD = 4.1878, F-Score = 0.9367, NC = 0.9920. Excellent reconstruction metrics for complex biological geometry.

5.2.4 Evaluation Metrics Table For All Models

Model Name	Chamfer Distance	F-Score	Normal Consistency
Bunny1	3.2342	0.9893	0.9921
Bunny2	4.8543	0.9321	0.9879
Sphere	1.2203	0.9912	0.9924
Cylinder	2.0327	0.9879	0.9889
Explicit Lug	4.897	0.94	0.9927
Elephant	3.777	0.9344	0.9717
Petal	4.9925	0.9851	0.9911
Sacrum Bone	4.4596	0.9129	0.9795
Cervical Vertebrae	4.1878	0.9367	0.9920

Table 5.2: Evaluation metric scores for each reconstructed model. Lower Chamfer Distance and higher F-Score and Normal Consistency indicate better reconstruction quality.

The quantitative results indicate that the proposed reconstruction method achieves optimal performance on relatively simple geometries such as the **sphere**, **cylinder**, and **bunny**, as evidenced by low Chamfer Distances, F-Scores approaching unity, and high Normal Consistency values. In contrast, for more topologically intricate shapes like the **scrum bone** and **cervical vertebrae**, the method maintains high F-Scores and low reconstruction errors; however, a marginal decrease in Normal Consistency is observed. This suggests slight challenges in accurately capturing fine-grained surface orientations. Overall, the method demonstrates robust generalization across both simple and complex 3D structures.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis presented a self-supervised neural framework for 3D surface reconstruction from unstructured point cloud data, leveraging the power of Signed Distance Function (SDF) learning via a Sinusoidal Representation Network (SIREN). The approach required no ground-truth supervision and was driven by a carefully crafted sampling pipeline that generated near-surface, far-field, and surface-normal point pairs, ensuring comprehensive geometric coverage during training.

The model incorporated a suite of geometric losses including Eikonal, Dirichlet (matching and non-matching), and singular hessian constraints along with practical training strategies such as learning rate decay, gradient clipping, and dynamic loss weighting. Surfaces were extracted from the trained SDF using the Marching Cubes algorithm on a 256^3 grid, capturing intricate geometric details across a variety of CAD-like shapes.

Quantitative evaluation revealed high reconstruction fidelity, evidenced by low Chamfer Distance, strong F-Score metrics, and excellent Normal Consistency—especially for simpler geometries. Additionally, qualitative analysis, such as centroid comparison on the Bunny model, confirmed the effectiveness of surface orientation alignment and highlighted opportunities for refinement in local accuracy.

Overall, the proposed framework delivers a flexible, self-supervised solution for high-fidelity mesh reconstruction, demonstrating robustness to sparse, noisy, and unordered input data. By bridging implicit representation learning with geometric priors, this work contributes a scalable foundation for learning-based surface modeling.

6.2 Future Work

Building upon the current framework, several research directions can be pursued to enhance both accuracy and scalability:

1. **Alternative Neural Architectures:** Investigate advanced architectures such as Fourier Feature Networks positional encoding-based models to better capture high-frequency geometric features and sharper boundaries.

2. **Enhanced Loss Functions:** Incorporate curvature-based regularization or anisotropic constraints to further improve the reconstruction of thin structures, edges, and fine surface details.

3. **Dynamic Point Clouds:** Extend the static reconstruction pipeline to handle time-varying or deformable point clouds, enabling applications in robotics, motion capture, and AR/VR.

4. **Optimized Surface Extraction:** Replace the standard Marching Cubes algorithm with adaptive-resolution or GPU-accelerated variants to reduce memory and computational overhead while preserving mesh quality.

5. **Noise Robustness:** Enhance the resilience of the training pipeline by introducing noise-aware sampling techniques, probabilistic loss formulations, or data augmentation strategies to handle real-world sensor imperfections.

6. **Multi-Modal Fusion:** Explore the integration of auxiliary data sources such as RGB images or depth maps to enrich the point cloud input and guide learning toward higher semantic and geometric accuracy.

7. **Quad Mesh Generation:** Investigate methods to convert the extracted triangle meshes into structured quadrilateral representations, which are better suited for downstream applications like simulation and CAD-based editing.

Bibliography

[1] L. Custodio, S. Pesco, and C. Silva. An extended triangulation to the marching cubes 33 algorithm. *Journal of XYZ*, 2023.

[2] Q. Dong, Y. Guo, J. Zhang, and Z. Han. Neurcadrecon - neural signed distance function for cad models. `https://github.com/QiujieDong/NeurCADRecon`, 2023. Accessed: 2025-06-07.

[3] Q. Dong, H. Wen, R. Xu, X. Yu, J. Zhou, S. Chen, S. Xin, C. Tu, and W. Wang. Neurcross: A self-supervised neural approach for representing cross fields in quad mesh generation. *ACM Transactions on Graphics*, 2024.

[4] Q. Dong, R. Xu, P. Wang, S. Chen, S. Xin, X. Jia, W. Wang, and C. Tu. Neurcadrecon: Neural representation for reconstructing cad surfaces by enforcing zero gaussian curvature. *ACM Transactions on Graphics*, 2024.

[5] F. Z. Iguenfer, A. Hsain, H. Amissa, and Y. Chtouki. Point cloud to mesh reconstruction. *School of Science and Engineering, Al Akhawayn University*, 2024.

[6] Y. Li, S. Liu, Y. Qian, X. Chen, H. Wu, and Y. Yu. Neural singular hessian - sharp feature preserving sdf learning. `https://github.com/bearprin/Neural-Singular-Hessian`, 2024. Accessed: 2025-06-07.

[7] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 1987.

[8] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *CVPR*, 2019.

[9] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Siren - implicit neural representations with periodic activation functions. `https://github.com/vsitzmann/siren`, 2020. Accessed: 2025-06-07.

[10] V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[11] Z. Wang, Y. Zhang, R. Xu, F. Zhang, P.-S. Wang, S. Chen, S. Xin, W. Wang, and C. Tu. Neural-singular-hessian: Implicit neural representation of unoriented point clouds by enforcing singular hessian. *ACM Transactions on Graphics*, 42(6), Dec 2023.