Learning 3D Mesh and Scene Parameters with improved Neural Networks Operators

by

Annada Prasad Behera

MATH11201904001

National Institute of Science Education and Research
Bhubaneswar

A thesis submitted to the
Board of Studies in Mathematical Sciences

In partial fulfillment of requirements
for the Degree of

DOCTOR OF PHILOSOPHY
of
HOMI BHABHA NATIONAL INSTITUTE

December, 2025

Homi Bhabha National Institute

Recommendation of the Viva Voce Committee

As members of the Viva-voce Committee, we certify that we have read the dissertation prepared by *Annada Prasad Behera* entitled "*Learning 3D Mesh and Scene Parameters with improved neural network operators*" and recommend that it may be accepted as fulfilling the thesis requirement for the award of Degree of Doctor of Philosophy.

	Doctoral Committee	Name	Signature	Date	In-person/ Online
1	*Chairman*	Prof. Brundaban Sahu		23.03.26	In-person
2	*Guide & Convenor*	Dr. Subhankar Mishra		23.3.26	In-person
3	*Examiner*	Dr. Debashree Guha Adhya	Debashree Guha Adhya	23.03.2026	Online
4	*Member 1*	Dr. Aritra Banik		23/03/26	In-person
5	*Member 2*	Dr. Anup Bhattacharya		23.03.26	In-person
6	*Member 3*	Dr. Debi Prosad Dogra		23/03/26	In-person

Final approval and acceptance of this thesis is contingent upon the candidate's submission of the final copies of the thesis to Homi Bhabha National Institute, Mumbai.

I/We hereby certify that I have read this thesis prepared under my direction and recommend that it may be accepted as fulfilling the thesis requirement.

Date: March 23, 2026
Place: NISER, Bhubaneswar

Subhankar Mishra
(Guide & Convenor)

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirement for an advanced degree at Homi Bhabha National Institute, Mumbai (HBNI) and is deposited in the library to be made available to borrowers under the rules of HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of the source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part, maybe granted by the Competent Authority of HBNI when in his/her judgment the proposed use of the material is in the interest of scholarship. In all other instances, however, permission must be obtained from the author.

Annada Prasad Behera

Annada Prasad Behera

DECLARATION

I, Annada Prasad Behera, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree/diploma at this or any other Institution/University.

Annada Prasad Behera

Annada Prasad Behera

CERTIFICATION ON ACADEMIC INTEGRITY

Undertaking by the Student

i. I, Annada Prasad Behera, HBNI Enrolment Number MATH11201904001 hereby undertake that the thesis titled "**Learning 3D Mesh and Scene Parameters from Images with improved neural network operators**" is prepared by me and is the original work undertaken by me.

ii. I, also, hereby undertake that this document has been duly checked through a plagiarism detection tool and the document is plagiarism-free as per the guidelines of my Institute/UGC.

iii. I am aware and undertake that if plagiarism is detected in my theses at any point in the future, suitable penalties will be imposed as per the guidelines of the Institute/UGC.

Annada Prasad Behera
March 23rd, 2026
Annada Prasad Behera

Endorsement by the Theses Supervisor

I certify that the theses written by the researcher is plagiarism free as mentioned above by the student.

23.3.2026

Signature of the Thesis Supervisor with date:

Name:	Subhankar Mishra
Designation:	Reader-F
Department:	School of Computer Sciences
Name of the CI/OCC:	National Institute of Science Education and Research, Bhubaneswar

AI Disclosure Statement

This thesis was prepared with the assistance of artificial intelligence tools. The specific uses of AI technology in this work are disclosed as follows:

Large Language Models (LLMs) were employed solely to refine the grammatical accuracy and structural organization of the written content. These tools were used as editorial aids to enhance clarity, coherence, and readability of the text. All substantive content, including research design, data collection, analysis, interpretation of results, arguments, conclusions, and intellectual contributions contained within this thesis, originated entirely from the author's own work and understanding.

Diffusion-based generative models were utilized to create certain illustrations, diagrams, and visual elements presented in this thesis. These AI-generated visuals were produced to support and clarify concepts developed by the author and were subsequently reviewed and, where necessary, modified to ensure accuracy and appropriateness.

This disclosure is provided in the interest of transparency regarding the ethical use of artificial intelligence in academic work.

Annada Prasad Behera

Annada Prasad Behera

List of Publications

Publications related to thesis:

1. **Annada Prasad Behera** and Subhankar Mishra, *Neural directional distance field object representation for uni-directional path-traced rendering.* In 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), pages 1–6, 2023. https://doi.org/10.1109/ICCCNT56998.2023.10308373

2. **Annada Prasad Behera** and Subhankar Mishra, *Digitizing temples for heritage conservation: Kalinga architecture.* Digital Applications in Archaeology and Cultural Heritage, 39:e00462, 2025. https://doi.org/10.1016/j.daach.2025.e00462

3. **Annada Prasad Behera**, Swati Rani Hait, Bapi Dutta, and Subhankar Mishra, *Restriction based overlap and grouping functions with its application in convolutional neural network architecture.* Information Sciences, 2025. https://doi.org/10.1016/j.ins.2025.123006

Conference presentations related to thesis:

1. **Annada Prasad Behera** and Subhankar Mishra, *Classical orthonormal polynomials as activation functions for implicit neural representations to preserve high frequency sharp features.* 16th Asia Pacific Conference on Vision, July 2024, Yale–National University of Singapore, Singapore.

2. **Annada Prasad Behera** and Subhankar Mishra, *Digitizing temples for heritage conservation: Kalinga architecture.* International Heritage and Cultural Conservation Conference, August 2023, University of Technology Sarawak, Malaysia.

Other publications not related to thesis:

1. Rucha Bhalchandra Joshi, **Annada Prasad Behera**, and Subhankar Mishra, *eBIM-GNN: Fast and scalable energy analysis.* IEEE EnergyCon, 2022. https://doi.org/10.5220/0009575301220127

2. Rucha Bhalchandra Joshi, **Annada Prasad Behera**, and Subhankar Mishra, *Temporal motifs in smart grid.* SMARTGREENS, 2020. https://doi.org/10.1109/ENERGYCON53164.2022.9830484

Annada Prasad Behera

Annada Prasad Behera

REPRODUCIBILITY STATEMENT

All results reported in this thesis are reproducible from publicly available resources. Specific details are as follows.

Code. Source code for the k-restricted pooling experiments is available in the supplementary repository associated with [7]. Code for neural DDF training, surface-based sampling, and path-traced rendering accompanies the published conference paper [8]. Processing scripts for point cloud compression, inverse density sampling, mesh decimation, and bounding-box isolation are released with the heritage digitization study [9].

Datasets. All image classification datasets (MNIST, Balanced EMNIST, Fashion-MNIST, CIFAR-10, CIFAR-100, CALTECH-101, CALTECH-256, STL-10, Flowers102, TinyImageNet) are publicly available from their respective original sources; dataset versions, download links, and MD5 checksums are documented in the code repository. Neural DDF experiments use ShapeNet objects (airplane, car, chair, couch, table) and the Stanford bunny and Blender Suzanne meshes, all of which are freely available. The Rajarani Temple raw images, point cloud, photogrammetric outputs, and processing code are not publicly archived due to the scale of the data; they are available on request from the thesis advisor, Dr. Subhankar Mishra (`www.niser.ac.in/~smishra`).

Hyperparameters. Complete hyperparameter configurations are reported in the body of the thesis. k-restricted pooling experiments: SGD optimizer, momentum 0.9, weight decay 10^{-4}, cosine annealing over 100 epochs, sensitivity parameter $p \in \{1, 2, 3\}$ selected by cross-validation, 10 independent runs per configuration (Section 4.3.4). Neural DDF training: Adam optimizer, learning rate 10^{-7}, 8 hidden layers of width 512, dropout $p = 0.5$, clamped loss with $\delta = 0.1$, safety factor $\alpha = 0.9$ (Sections 5.3 and 5.5). Heritage processing: FARO M70 scanner at 62 positions; 75% UAV image overlap, inverse density sampling to 10% of raw point count, edge contraction decimation at 1:10 ratio (Section 6.5.3).

Annada Prasad Behera

Annada Prasad Behera

... to my Mama.

Contents

Abstract

The increasing complexity of data in modern computational fields necessitates the development of specialized tools for data processing, analysis, and representation. This thesis addresses this need by introducing novel methods at the intersection of computer graphics, machine learning, and digital heritage. The work spans theoretical advances in aggregation functions, novel rendering techniques for neural representations, and practical applications in cultural heritage documentation.

First, we introduce k-restricted overlap and grouping functions, a generalized family of aggregation operators that extends traditional n-dimensional overlap and grouping functions by incorporating restrictions based on boundary element counts. These operators enable flexible control over conjunctive, disjunctive, and averaging behaviors through a parameter k, where the function output depends on at least k boundary elements rather than a single element. We establish existence and representation theorems for these functions and demonstrate their practical utility as pooling layer replacements in Convolutional Neural Networks (CNNs). Experimental validation across multiple architectures (LeNet-5, VGG16, ResNet variants) and datasets shows improved performance (e.g., up to .05 accuracy gain on already extremely optimized architectures), particularly on sparse training samples, with faster processing on both CPUs and GPUs compared to traditional pooling methods.

Second, we propose Neural Directional Distance Fields (NDDFs) as an efficient object representation for path-traced rendering. Unlike traditional signed distance functions, DDFs incorporate directional components, enabling constant-time ray-surface intersection queries while maintaining surface detail preservation. We develop a modified path-tracing algorithm specifically designed for NDDF rendering and introduce systematic sampling techniques for training neural networks to represent these fields. Our approach demonstrates significant rendering speedup compared to traditional bounding volume hierarchy methods while preserving geometric accuracy, as validated through chamfer distance metrics and visual quality assessments.

Third, we address practical challenges in Heritage Building Information Modeling (HBIM). We propose a comprehensive framework for digitizing, storing, and transmitting large-scale cultural heritage data. Using the 11th-century Rajarani Temple in Bhubaneswar, India, as a case study, we develop techniques for segmentation, decimation, mesh matching, and hierarchical storage of heritage models containing over 780 million points. Our methodology enables efficient transmission over limited bandwidth networks and exhibition on consumer-grade hardware while preserving culturally significant architectural details. The framework employs inverse density sampling, density-aware chamfer distance metrics, and kd-tree-based hierarchical storage to balance computational efficiency with preservation fidelity.

The three contributions demonstrate a structured approach to advancing computational methods, progressing from theoretical foundations to algorithmic innovations and real-world applications. The k-restricted aggregation functions provide mathematical tools with broad applicability beyond computer vision, the neural directional distance fields offer new possibilities for efficient 3D rendering, and the heritage digitization framework addresses urgent needs in cultural preservation. Together, these works contribute to the intersection of mathematics, computer graphics, and digital humanities, providing both theoretical insights and practical solutions for complex computational challenges.

Keywords: Aggregation functions, neural rendering, directional distance fields, heritage preservation, point cloud processing, convolutional neural networks, path tracing, HBIM

List of Figures

List of Tables

1 Introduction

Real-world applications motivate the technical challenges addressed in this thesis. Autonomous vehicles require CNNs that extract robust features from camera feeds for real-time object detection and scene understanding. Medical imaging systems use deep networks to identify tumors in CT scans and segment organs in MRI volumes, where preserving fine spatial details during feature aggregation directly affects diagnostic accuracy. Film production pipelines increasingly rely on inverse rendering to estimate scene parameters from photographs, enabling virtual set extensions and digital asset creation. Video game engines demand real-time differentiable rendering for dynamic lighting and material editing. Heritage institutions face urgent digitization needs: the 2019 Notre-Dame fire demonstrated how quickly irreplaceable cultural artifacts can be lost, while COVID-19 restrictions highlighted the value of virtual access to cultural sites. These applications share common technical requirements: efficient feature aggregation that preserves important information, rendering systems compatible with gradient-based optimization, and scalable methods for storing and transmitting large 3D datasets.

Three-dimensional reconstruction recovers 3D geometry and appearance from 2D observations or sensor measurements. Traditional methods use geometric techniques like photogrammetry, stereo vision, and structure from motion with camera calibration, feature matching, and triangulation. These found applications in medical imaging (CT, MRI), architectural documentation, and manufacturing quality control.

Neural 3D reconstruction employs deep learning to create models from photographs or depth data. Unlike traditional methods that manually define shapes, neural approaches learn scene representations from training data, capturing complex objects with high detail and realistic appearance.

Traditional representations like meshes and point clouds have limitations. Meshes struggle with complex topologies and varying detail levels; point clouds lack surface information. Neural representations use continuous functions to address these issues. Neural Radiance Fields (NeRFs) map 3D coordinates to density and color values. Gaussian Splatting uses learnable 3D Gaussian primitives. Both enable photorealistic rendering while handling complex geometry better than explicit representations.

Neural representations require differentiability for gradient computation and backpropagation optimization. The system adjusts its representation by minimizing differences between rendered and ground truth images. Without differentiable operations, optimizing high-dimensional parameter spaces becomes intractable.

Digital heritage preservation applies these reconstruction techniques to cultural documentation. Laser scanning and photogrammetry generate dense point clouds capturing architectural details at millimeter precision. However, the resulting datasets pose storage, transmission, and computational challenges. A single heritage site can produce

hundreds of millions of points requiring tens of gigabytes of storage. Distributing such datasets over consumer internet connections and rendering them on standard hardware remains impractical without compression and level-of-detail strategies [9].

This thesis addresses three interconnected problems in neural 3D reconstruction. First, pooling operators in neural networks aggregate spatial information but traditional max and average pooling lose important information and fail to capture complex relationships. Second, standard rendering engines are incompatible with gradient-based optimization due to discrete operations that break differentiability. Third, heritage digitization generates massive datasets that exceed storage, transmission, and exhibition capabilities.

The thesis contributions span three areas. K-restricted grouping functions [7] generalize traditional aggregation operators. They preserve critical features while maintaining efficiency and help networks capture hierarchical structure. Neural directional distance fields [8] enable differentiable rendering with constant-time ray intersection queries. Heritage digitization methodology [9] addresses storage and transmission of massive datasets.

The following sections detail each problem and its technical context.

1.1 Problem of Limited Pooling Functions in CNNs

CNNs rely on pooling layers to reduce spatial dimensions and extract features. Traditional pooling operators have limitations that compromise feature extraction and training efficiency [7].

Max pooling preserves only the maximum value within each pooling window, discarding potentially relevant features. For a window containing values $[0.95, 0.94, 0.93, 0.02]$, max pooling returns 0.95 while ignoring three similar activations. Gradients flow only to the maximum element, creating sparse updates that cause vanishing gradients in deep networks.

Average pooling distributes gradients equally but fails to capture feature importance. For window $[0.95, 0.05, 0.05, 0.05]$, average pooling yields 0.275, diluting the dominant feature.

Traditional overlap and grouping functions exhibit boundary sensitivity: a single zero input forces zero output regardless of other values. This prevents flexible aggregation where inputs rarely reach exact boundary values. These pooling limitations compound through network depth as gradient magnitudes decay multiplicatively.

While pooling addresses feature aggregation within networks, a separate challenge arises when integrating neural representations with rendering systems.

1.2 The Differentiable Rendering Problem

General-purpose rendering engines such as Cycles (Blender), Unreal Engine, and Arnold achieve photorealistic results through algorithms optimized for visual quality [82]. However, these systems are incompatible with gradient-based optimization due to several

limitations that break differentiability [62, 68].

Traditional rendering pipelines contain operations that preclude gradient computation: ray-surface intersection tests return binary visibility decisions with zero gradients; material assignments involve discrete conditional branching; and Monte Carlo sampling introduces variance that destabilizes gradient estimation [4].

Automatic differentiation [6] enables gradient-based optimization of neural networks by maintaining computational graphs with recorded forward and backward operations. Neural networks use differentiable operations throughout, making autodiff effective. However, integrating neural representations with traditional renderers requires replacing non-differentiable operations with continuous approximations.

Beyond rendering, practical deployment of 3D reconstruction faces infrastructure constraints when handling large-scale datasets.

1.3 Problem of Transmission, Exhibition and Storage

Digital heritage preservation generates massive datasets that exceed conventional infrastructure capabilities [89, 65]. High-resolution 3D reconstructions create practical barriers for storage, network transmission, and real-time exhibition [9].

Heritage digitization is challenging because accuracy requirements conflict with practical constraints. Preserving architectural details at millimeter precision produces point clouds with hundreds of millions of points. The Rajarani Temple reconstruction contains 782 million points requiring 35 GiB storage [9]. Unlike generic 3D content that can be aggressively compressed, heritage models must preserve culturally significant details that naive decimation destroys.

Storage costs scale linearly with scanned volume and point density. Network transmission of full-resolution models requires tens of minutes even on fast connections, with connection failures requiring complete restarts. Consumer hardware with 8–16 GiB RAM cannot load such datasets into memory, preventing interactive exhibition on standard devices.

Spatial queries on heritage models stored as binary blobs require scanning entire datasets. Without hierarchical organization like kd-trees [12], locating specific architectural elements demands exhaustive search. Researchers need multiple resolution levels for different tasks, from overview visualization to detailed analysis, multiplying storage requirements [69].

1.4 Contribution and Thesis Overview

This thesis makes several contributions to neural 3D reconstruction, pooling operators, and heritage preservation:

k-Restricted Grouping Functions (Chapters 2-3)

- Formalized k-restricted overlap and grouping functions that generalize traditional aggregation operators
- Proved existence theorems confirming k-restricted functions exist for any positive integer k with dimensional constraints
- Established representation theorems characterizing analytical properties of k-restricted operators
- Extended function definitions to arbitrary closed intervals $[a, b]$ beyond the unit interval
- Demonstrated superior performance as CNN pooling replacements across multiple architectures (LeNet-5, VGG16, ResNet variants) and datasets (MNIST, CIFAR, CALTECH, TinyImageNet)
- Achieved faster training, improved accuracy, and enhanced gradient flow compared to traditional max and average pooling

Neural Directional Distance Fields (Chapter 4-5)

- Introduced directional distance fields (DDFs) as neural scene representations that encode both distance and directional information
- Developed uni-directional path tracing algorithms compatible with DDF representations
- Demonstrated constant-time ray intersection queries compared to logarithmic-time BVH structures
- Achieved competitive reconstruction quality with faster rendering performance on complex 3D scenes

Heritage Digitization and HBIM (Chapter 6)

- Developed comprehensive methodology for digital preservation of Kalinga architecture using the Rajarani Temple as case study
- Created efficient storage, transmission, and exhibition strategies for massive heritage datasets (782M points, 35 GiB)
- Implemented hierarchical database organization using kd-trees for spatial querying
- Demonstrated mesh decimation and point cloud sampling techniques preserving cultural significance
- Developed density-aware chamfer distance metrics for point cloud matching in heritage applications
- Implemented inverse density importance sampling preserving fine architectural details
- Created bounding box isolation algorithms for real-time heritage model exhibition
- Established automated mesh-to-point cloud matching using gradient-based optimization

The thesis demonstrates how neural representations, advanced aggregation operators, and efficient data management can address fundamental challenges in 3D reconstruction

and heritage preservation. These contributions enable new applications in cultural documentation, virtual tourism, and archaeological research while advancing the theoretical foundations of neural rendering and aggregation functions.

2 Aggregation Functions and Pooling Operators

Pooling operators in neural networks perform aggregation over spatial neighborhoods, reducing feature map dimensions while preserving relevant information. Traditional pooling functions like max-pooling and average-pooling represent specific instances of disjunctive and averaging aggregation functions respectively [58]. Max-pooling extracts the maximum value from each spatial window, exhibiting pure disjunctive behavior where a single high activation dominates the output. Average-pooling computes the arithmetic mean, providing uniform weighting across all inputs within the pooling kernel [17].

The rigid boundary behavior of conventional pooling operators limits their adaptability to varying feature distributions and training dynamics. k-Restricted grouping functions offer a controlled relaxation of the disjunctive property, requiring multiple high activations before producing maximum output rather than relying on a single dominant feature. This modification spreads gradient flow across more parameters during backpropagation while maintaining focus on prominent features [105]. This chapter serves as a prelims on aggregate function, k-restriced functions and their construction methods, while the following chapter demonstrates how k-restricted grouping functions can replace traditional pooling layers in convolutional neural networks, providing improved training dynamics and performance across diverse image classification tasks.

2.1 Aggregation Functions

Aggregation functions combine multiple input values into a single output value while preserving monotonicity and boundary conditions. These operators form the theoretical foundation for information fusion across diverse domains [38].

Definition 1 (Fusion Function)**.** *A function* $F : [0,1]^n \to [0,1]$ *with* $n \in \mathbb{N}$, $n \geq 2$, *is a fusion function [20].*

Definition 2 (Aggregation Function)**.** *An aggregation function is a fusion function* $F : [0,1]^n \to [0,1]$, $n \in \mathbb{N}$ *satisfying [38]:*

(a) F *is non-decreasing in each argument (i.e., for each* i, $F(x_1, \ldots, x_i, \ldots, x_n) \leq F(x_1, \ldots, x_i', \ldots, x_n)$ *whenever* $x_i \leq x_i'$*)*

(b) $F(0, \ldots, 0) = 0$ *and* $F(1, \ldots, 1) = 1$

Aggregation functions can be categorized by their relationship to the minimum and maximum operations:

Definition 3 (Aggregation Function Categories)**.** *Let $F : [0,1]^n \to [0,1]$ be an aggregation function. Then F is [39]:*
Throughout this chapter, $[n] := \{1, \ldots, n\}$ denotes the index set.

(a) ***Conjunctive*** *if $F(x_1, \ldots, x_n) \leq \min_{i \in [n]}\{x_i\}$*

(b) ***Disjunctive*** *if $F(x_1, \ldots, x_n) \geq \max_{i \in [n]}\{x_i\}$*

(c) ***Averaging*** *if $\min_{i \in [n]}\{x_i\} \leq F(x_1, \ldots, x_n) \leq \max_{i \in [n]}\{x_i\}$*

(d) ***Mixed*** *if none of the above hold*

2.1.1 Overlap Functions

Overlap functions quantify the intersection degree when an entity belongs to multiple overlapping classes. These functions exhibit conjunctive behavior with strict annihilation properties [19].

Definition 4 (n-dimensional Overlap Function)**.** *An aggregation function $O : [0,1]^n \to [0,1]$ is an n-dimensional overlap function if [37]:*

(a) O is symmetric

(b) $O(x_1, \ldots, x_n) = 0$ iff at least one $x_i = 0$

(c) $O(x_1, \ldots, x_n) = 1$ iff $\prod_{i=1}^{n} x_i = 1$

(d) O is non-decreasing

(e) O is continuous

The second condition enforces that any zero input forces zero output, reflecting the conjunctive nature where all classes must have non-zero membership for positive overlap.

2.1.2 Grouping Functions

Grouping functions assess the degree of evidence for an entity belonging to any available class. They exhibit disjunctive behavior, serving as duals to overlap functions [21].

Definition 5 (n-dimensional Grouping Function)**.** *An aggregation function $G : [0,1]^n \to [0,1]$ is an n-dimensional grouping function if [21]:*

(a) G is symmetric

(b) $G(x_1, \ldots, x_n) = 0$ iff all $x_i = 0$

(c) $G(x_1, \ldots, x_n) = 1$ *iff at least one* $x_i = 1$

(d) G *is non-decreasing*

(e) G *is continuous*

Definition 6 (N-dual)**.** *Let* $N : [0,1] \to [0,1]$ *be a strong negation and* $F : [0,1]^n \to [0,1]$ *be a fusion function. The N-dual of* F*, denoted* $F_N : [0,1]^n \to [0,1]$*, is [38]:* $F_N(x_1, \ldots, x_n) = N(F(N(x_1), \ldots, N(x_n)))$

Grouping functions can be constructed as N-duals of overlap functions, establishing a duality relationship between conjunctive and disjunctive behaviors.

2.1.3 Extended Aggregation Functions

Extended aggregation functions operate over varying dimensions, enabling flexible application across different input sizes.

Definition 7 (Extended Aggregation Function)**.** *A fusion function* $F^* : \bigcup_{n\in\mathbb{N}}[0,1]^n \to [0,1]$ *is an extended aggregation function if* $F^n = F^*|_{[0,1]^n}$ *is an aggregation function for every* $n \in \mathbb{N}$*.*

Extended functions provide the foundation for constructing k-restricted operators that modify traditional aggregation behavior by introducing restrictions on boundary element counts.

2.2 k-Restricted Functions

Traditional overlap and grouping functions exhibit rigid boundary behavior: overlap functions return zero when any input is zero, while grouping functions return one when any input is one. k-Restricted functions relax these constraints by requiring at least k boundary elements before triggering the boundary output.

2.2.1 k-Restricted Overlap Functions

k-Restricted overlap functions modify the annihilation property of traditional overlap functions by requiring at least k zero inputs before producing zero output [7].

Definition 8 (k-Restricted Overlap Function)**.** *Let* $F^n : [0,1]^n \to [0,1]$ *be an n-dimensional aggregation function and* $k \in \{1, \ldots, n\}$*. Then* F^n *is a k-restricted overlap function if:*

(a) F^n *is symmetric*

(b) F^n *is non-decreasing*

(c) F^n *is continuous*

(d) $F^n(x_1, \ldots, x_n) = 1$ *iff* $x_i = 1$ *for all* $i \in \{1, \ldots, n\}$

(e) If at least k *arguments in* $(x_1, \ldots, x_n)$ *are zero, then* $F^n(x_1, \ldots, x_n) = 0$

(f) If strictly less than k *arguments are zero in* $(x_1, \ldots, x_n)$*, then* $F^n(x_1, \ldots, x_n) \neq 0$

When $k = 1$, conditions 5 and 6 reduce to the standard overlap function definition, making k-restricted overlap functions a generalization of traditional overlap functions [37].

Examples of k-Restricted Overlap Functions For bivariate functions over $[0, 1]^2$:

- $F^2(x_1, x_2) = \min\{x_1^p, x_2^p\}$ with $p > 0$ is 1-restricted
- $F^2(x_1, x_2) = \left(\frac{x_1^p + x_2^p}{2}\right)^{1/p}$ with $p > 0$ is 2-restricted

For n-dimensional functions, the Bonferroni mean:

$$\mathrm{BM}^n(x_1, \ldots, x_n) = \left(\frac{1}{n(n-1)} \sum_{i,j=1, i \neq j}^{n} x_i^p x_j^q\right)^{1/(p+q)}$$

is $(n-1)$-restricted [11].

2.2.2 k-Restricted Grouping Functions

k-Restricted grouping functions modify the absorption property by requiring at least k inputs equal to one before producing unit output.

Definition 9 (k-Restricted Grouping Function)**.** *Let* $T^n : [0, 1]^n \to [0, 1]$ *be an* n*-dimensional aggregation function and* $k \in \{1, \ldots, n\}$*. Then* T^n *is a k-restricted grouping function if:*

(a) T^n *is symmetric*

(b) T^n *is non-decreasing*

(c) T^n *is continuous*

(d) $T^n(x_1, \ldots, x_n) = 0$ *iff* $x_i = 0$ *for all* $i \in \{1, \ldots, n\}$

(e) If at least k *arguments in* $(x_1, \ldots, x_n)$ *are one, then* $T^n(x_1, \ldots, x_n) = 1$

(f) If strictly less than k *arguments are one in* $(x_1, \ldots, x_n)$*, then* $T^n(x_1, \ldots, x_n) \neq 1$

Theorem 1 (Duality of k-Restricted Functions)**.** *Let* F^n *be an n-dimensional k-restricted overlap function and* $N : [0, 1] \to [0, 1]$ *be a strict negation. Then* $T^n : [0, 1]^n \to [0, 1]$ *defined as:*

$$T^n(x_1, \ldots, x_n) = N(F^n(N(x_1), \ldots, N(x_n)))$$

is an n-dimensional k-restricted grouping function.

This duality relationship allows constructing k-restricted grouping functions from k-restricted overlap functions using negation operators [21].

2.2.3 Existence Theorems

The existence of k-restricted functions for arbitrary k requires dimensional constraints and extended function families.

The following theorems are from [7].

Theorem 2 (Existence Theorem I for k-Restricted Overlap Functions). *Let $O^* : \bigcup_{n\in\mathbb{N}}[0,1]^n \to [0,1]$ be an extended overlap function and $k \geq 2$. Let n be odd satisfying $\frac{n}{2} + 0.5 = k$. Define $F^n : [0,1]^n \to [0,1]$ as:*

$$F^n(x_1, \ldots, x_n) = \frac{1}{\binom{n}{k}} \sum_{\{i_1,\ldots,i_k\}\subset[n]} O^k(x_{i_1}, \ldots, x_{i_k})$$

where $O^k = O^*|_{[0,1]^k}$. Then F^n is an n-dimensional k-restricted overlap function.

Theorem 3 (Existence Theorem II for k-Restricted Overlap Functions). *Let $O^* : \bigcup_{n\in\mathbb{N}}[0,1]^n \to [0,1]$ be an extended overlap function and $k \geq 2$. Let $n = 2k - 2$. Define F^n as above. Then F^n is a $(k-1)$-restricted overlap function over $[0,1]^n$.*

Parallel existence theorems hold for k-restricted grouping functions using extended grouping function families [21].

2.2.4 Representation Theorems

k-Restricted functions admit representation forms analogous to traditional overlap and grouping functions.

Theorem 4 (Representation Theorem for k-Restricted Overlap Functions). *The mapping $F^n : [0,1]^n \to [0,1]$ is an n-dimensional k-restricted overlap function iff F^n can be represented as:*

$$F^n(x_1, \ldots, x_n) = \frac{f^n(x_1, \ldots, x_n)}{f^n(x_1, \ldots, x_n) + g^n(x_1, \ldots, x_n)}$$

where $f^n, g^n : [0,1]^n \to [0,1]$ satisfy:

(a) f^n and g^n are symmetric and continuous

(b) f^n is increasing, g^n is decreasing

(c) $f^n(x_1, \ldots, x_n) = 0$ iff at least k arguments are zero

(d) $g^n(x_1, \ldots, x_n) = 0$ iff $x_i = 1$ for all $i \in \{1, \ldots, n\}$

This representation form provides a constructive approach for generating k-restricted functions from appropriate function pairs [19].

2.3 Construction Methods

k-Restricted functions can be constructed through various algebraic operations, enabling systematic generation of new operators from existing ones.

2.3.1 Convex Combinations

Convex combinations preserve the k-restricted property under appropriate conditions.

Proposition 1 (Convex Combination of k-Restricted Overlap Functions). *Let $F_1^n, \ldots, F_m^n$ be m k-restricted overlap functions and $w_i \geq 0$ with $\sum_{i=1}^m w_i = 1$. Then:*

$$F^n(x_1, \ldots, x_n) = \sum_{i=1}^{m} w_i F_i^n(x_1, \ldots, x_n)$$

is also a k-restricted overlap function.

Proposition 2 (Weighted Combination of Mixed k-Restricted Functions). *Consider $F_1^n, \ldots, F_m^n$ as $k_1, \ldots, k_m$-restricted overlap functions with weights $w_i \geq 0$, $\sum_{i=1}^m w_i = 1$. Then:*

(a) If all $w_i > 0$, the weighted sum is k-restricted where $k = \max_{i \in [m]}\{k_i\}$

(b) If some weights are zero, $k = \max\{k_{j_s} : w_{j_s} > 0\}$

These results enable flexible construction of k-restricted functions with controllable restriction parameters [11].

2.3.2 Composition Methods

Composition operations combine multiple k-restricted functions using aggregation operators.

Theorem 5 (Composition Construction for k-Restricted Overlap Functions). *[7] Let $M : [0,1]^2 \to [0,1]$ be an aggregation function satisfying:*

(a) M is continuous

(b) $M(x, y) = 1$ iff $x = 1 = y$

(c) M has no zero divisors

Let $F_1^n, F_2^n : [0,1]^n \to [0,1]$ be k-restricted overlap functions. Then:

$$F^n(x_1, \ldots, x_n) = M(F_1^n(x_1, \ldots, x_n), F_2^n(x_1, \ldots, x_n))$$

is also a k-restricted overlap function.

Theorem 6 (Mixed Restriction Composition). *[7] Let F_1^n, F_2^n be k_1-restricted and k_2-restricted overlap functions respectively, and M satisfy conditions 1-2 above plus:*

(d) $M(x, y) = 0$ iff $x = 0 = y$

Then $F^n(x_1, \ldots, x_n) = M(F_1^n(x_1, \ldots, x_n), F_2^n(x_1, \ldots, x_n))$ is k-restricted where $k = \max\{k_1, k_2\}$.

Transformation Methods Strictly increasing transformations preserve the k-restricted property:

Proposition 3 (Monotonic Transformation). *Let F^n be a k-restricted overlap function and $M : [0, 1] \to [0, 1]$ be strictly increasing, continuous with $M(0) = 0$, $M(1) = 1$. Then $F_n^M(x_1, \ldots, x_n) = M(F^n(x_1, \ldots, x_n))$ is also k-restricted.*

This allows generating families of k-restricted functions using transformations like $M(x) = \sin(\frac{\pi x}{2})$ or $M(x) = x^2$ [54].

2.3.3 Generalization to Arbitrary Intervals

k-Restricted functions extend naturally to arbitrary closed intervals $[a, b] \subset \mathbb{R}$.

Definition 10 (k-Restricted Overlap Function). *Let $F_a^b : [a, b]^n \to [a, b]$ be an aggregation function over $[a, b]$ and $k \in \{1, \ldots, n\}$. Then F_a^b is k-restricted if:*

(a) F_a^b is symmetric, increasing, continuous

(b) $F_a^b(x_1, \ldots, x_n) = b$ iff $x_i = b$ for all i

(c) If at least k arguments equal a, then $F_a^b(x_1, \ldots, x_n) = a$

(d) If fewer than k arguments equal a, then $F_a^b(x_1, \ldots, x_n) \neq a$

Theorem 7 (Interval Transformation). *Let $F^n : [0, 1]^n \to [0, 1]$ be an aggregation function and $\phi : [a, b] \to [0, 1]$ be monotonically increasing, continuous, bijective. Let $F_a^b : [a, b]^n \to [a, b]$ be defined as:*

$$F_a^b(x_1, \ldots, x_n) = \phi^{-1}[F^n(\phi(x_1), \ldots, \phi(x_n))]$$

Then F_a^b is k-restricted over $[a, b]^n$ iff F^n is k-restricted over $[0, 1]^n$.

This transformation property enables transferring k-restricted functions between different interval domains using bijective mappings like $\phi(x) = \left(\frac{x-a}{b-a}\right)^p$ for $p > 0$ [71].

The construction methods provide systematic approaches for generating k-restricted operators with desired properties, enabling their application across diverse domains requiring flexible boundary behavior.

2.4 Summary and Application Context

The mathematical framework presented in this chapter provides the theoretical foundation for the pooling operators evaluated in Chapter 3. Traditional max-pooling corresponds to a 1-restricted grouping function (output is 1 when any input is 1), while average-pooling is an averaging function. The k-restricted generalization allows controlled relaxation: requiring $k > 1$ boundary inputs before producing boundary output spreads gradient flow across multiple elements during backpropagation, potentially improving training dynamics.

The existence theorems (Section 2.2) guarantee that k-restricted functions exist for any desired k with appropriate dimensional constraints. The representation theorems provide constructive formulas for implementing these functions. The construction methods (convex combinations, compositions, transformations) enable generating families of k-restricted pooling operators with tunable properties.

Chapter 3 applies eight specific k-restricted grouping functions as CNN pooling replacements, demonstrating that the theoretical properties translate to practical performance improvements across multiple architectures and datasets.

3 k-Restricted Functions in Neural Networks

This chapter examines the application of k-restricted grouping functions as pooling operators in Convolutional Neural Networks (CNNs). The experimental evaluation demonstrates performance improvements over traditional pooling methods across multiple architectures and datasets.

3.1 Pooling in Convolutional Neural Networks

LeNet-5 [58] represents one of the earliest CNN architectures, designed for handwritten digit recognition and serving as a foundational example of convolutional neural networks [7]. The architecture (as shown in **Fig. 3.1**)processes grayscale input images $X \in \mathbb{R}^{1\times28\times28}$ through a systematic sequence of feature extraction and classification stages.

The network begins with a convolutional layer that applies 6 filters of size 5×5 with padding, producing feature maps $C_1 = \sigma(X * W_1 + b_1)$ where $C_1 \in \mathbb{R}^{6\times28\times28}$ and σ denotes the activation function. This is followed by average pooling with a 2×2 kernel and stride 2, reducing spatial dimensions to yield $P_1 = \text{AvgPool}_{2\times2}(C_1) \in \mathbb{R}^{6\times14\times14}$.

The second convolutional stage applies 16 filters of size 5×5 to the pooled features, generating $C_2 = \sigma(P_1 * W_2 + b_2) \in \mathbb{R}^{16\times10\times10}$. Another average pooling operation further reduces the spatial resolution to $P_2 \in \mathbb{R}^{16\times5\times5}$. The resulting feature maps are flattened into a vector $F \in \mathbb{R}^{400}$ for processing by fully connected layers.

The classification portion consists of two fully connected layers with 120 and 84 neurons respectively, followed by a final output layer with 10 neurons for digit classification. The complete forward pass can be expressed as:

$$FC_1 = \sigma(W_3F + b_3) \in \mathbb{R}^{120} \tag{3.1}$$

$$FC_2 = \sigma(W_4FC_1 + b_4) \in \mathbb{R}^{84} \tag{3.2}$$

$$y = W_5FC_2 + b_5 \in \mathbb{R}^{10} \tag{3.3}$$

where classification probabilities are obtained through softmax normalization: $p(c_i|x) = e^{y_i} / \sum_{j=1}^{10} e^{y_j}$.

LeNet-5 demonstrates the hierarchical feature extraction principle where spatial dimensions decrease while feature depth increases: $28 \times 28 \to 14 \times 14 \to 5 \times 5$ spatially and $1 \to 6 \to 16$ channels. This pattern of alternating convolution and pooling operations for feature extraction followed by fully connected layers for classification be-

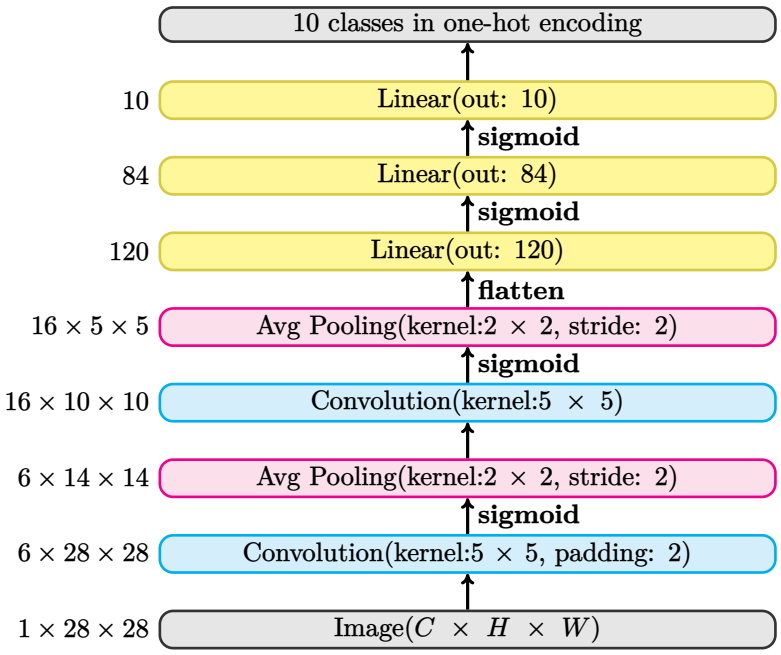

Fig. 3.1: Example of a CNN architecture–LeNet, demonstrating the use of average pooling layer most generally utilized in CNN architecture.

came the template for subsequent CNN architectures [7]. The network's approximately 60,000 parameters made it computationally feasible while achieving strong performance on handwritten digit recognition tasks.

Pooling layers reduce spatial dimensions of feature maps while preserving significant information [58]. For input $X \in \mathbb{R}^{C \times H \times W}$ with channels C, height H, and width W, a pooling operation with kernel size $k = (k_1, k_2)$ and stride $s = (s_1, s_2)$ produces output $Y \in \mathbb{R}^{C \times H' \times W'}$ where:

$$H' = \frac{H - k_1}{s_1} + 1 \tag{3.4}$$

$$W' = \frac{W - k_2}{s_2} + 1 \tag{3.5}$$

The pooling function $T : \mathbb{R}^{k_1 \times k_2} \to \mathbb{R}$ operates on local windows extracted from input feature maps.

3.1.1 Traditional Pooling Functions

Max Pooling Selects the maximum value from each local window [75]:

$$T_{\max}(x_1, \ldots, x_n) = \max_{i \in [n]} x_i \tag{3.6}$$

where $n = k_1 \times k_2$ represents the flattened window size. Max pooling preserves dominant features and provides translation invariance.

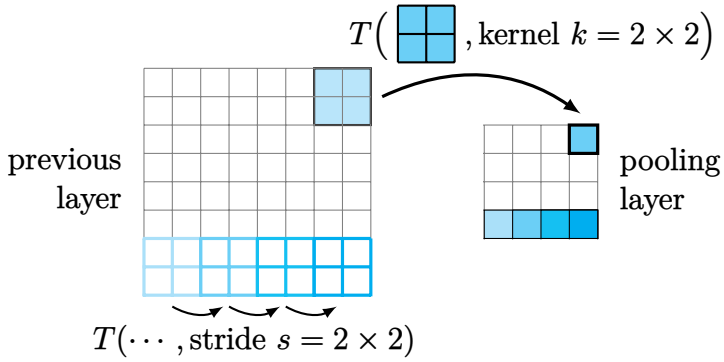

Fig. 3.2: Demonstrating the pooling layer in parallel with the convolutional layer, but without the learnable weights. The function T is the pooling function.

Average Pooling Computes the arithmetic mean [58]:

$$T_{\text{avg}}(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{3.7}$$

Average pooling smooths local variations and reduces noise in feature representations.

3.1.2 Limitations of Max and Average Pooling

Max pooling discards all values except the maximum, causing information loss even when multiple significant features exist within a window [90]. This creates sparse gradients during backpropagation, as only the maximum element receives gradient updates.

Average pooling treats all values equally, potentially diluting important features when combined with less relevant information. The uniform weighting fails to prioritize dominant features that carry semantic importance.

Both methods lack adaptive mechanisms to handle varying feature importance within local regions. The fixed aggregation strategies cannot adjust to different patterns or feature distributions encountered across diverse datasets and network depths [59].

3.2 k-Restricted Grouping as Pooling Functions

Traditional pooling methods suffer from fixed aggregation behavior that cannot adapt to local feature importance. k-restricted grouping functions provide flexible pooling operators that balance disjunctive and averaging behaviors based on the parameter k.

3.2.1 Function Properties

For pooling applications, we consider k-restricted grouping functions operating on flattened 2×2 windows, yielding input vectors $x = (x_1, x_2, x_3, x_4)$. The functions must

satisfy the grouping function properties from Definition 9 with specific pooling requirements.

Disjunctive Control The parameter k controls the disjunctive nature of the pooling operation. For $k = 1$, the function behaves as a standard grouping function, producing $T(x_1, \ldots, x_n) = 1$ when any $x_i = 1$. As k increases, the function requires more boundary elements to achieve maximum output, reducing disjunctive behavior and approaching averaging functions.

Feature Preservation Unlike max pooling, k-restricted grouping functions consider multiple significant values within each window. The restriction parameter k determines how many dominant features influence the final output, preventing single-value dominance while maintaining sensitivity to important activations.

Specific Pooling Functions We implement eight k-restricted grouping functions for 2×2 pooling kernels:

(a) $T^{\min}(x) = 1 - \left\{ 1/\binom{4}{2} \sum_{\substack{i,j \in \{1,\cdots,4\} \\ i \neq j}} \min\{(1-x_i)^p, (1-x_j)^p\} \right\}$

(b) $T^{\text{prod}}(x) = 1 - \left\{ 1/\binom{4}{2} \sum_{\substack{i,j \in \{1,\cdots,4\} \\ i \neq j}} (1-x_i)^p \cdot (1-x_j)^p \right\}$

(c) $T^{\text{sin-min}}(x) = 1 - \sin(\frac{\pi}{2}) T^{\min}(x)$

(d) $T^{\text{sin-prod}}(x) = 1 - \sin(\frac{\pi}{2}) T^{\text{prod}}(x)$

(e) $T^{\text{inv-prod}}(x) = 1 - \left\{ 1/\binom{4}{2} \sum_{\substack{i,j \in \{1,\cdots,4\} \\ i \neq j}} \frac{\prod_{s=1}^{2}(1-x_{i_s})}{1 + \prod_{s=1}^{2} x_{i_s}} \right\}$

(f) $T^{\text{D}}(x) = 1/\binom{4}{2} \sum_{\substack{i,j \in \{1,\cdots,4\} \\ i \neq j}} x_i + x_j - x_i x_j$

(g) $T^{\text{min-max}}(x) = 1 - \left\{ 1/\binom{4}{2} \sum_{\substack{i,j \in \{1,\cdots,4\} \\ i \neq j}} \min\{(1-x_i)^p, (1-x_j)^p\} \max\{(1-x_i)^p, (1-x_j)^p\} \right\}$

(h) $T^{\text{min-prod}}(x) = 1 - \left\{ 1/\binom{4}{2} \sum_{\substack{i,j \in \{1,\cdots,4\} \\ i \neq j}} \min\{(1-x_i)^p, (1-x_j)^p\} (1-x_i)^p (1-x_j)^p \right\}$

where $p > 0$ is a hyperparameter controlling the function's sensitivity.

3.2.2 Gradient Computation

k-restricted grouping functions provide continuous derivatives, enabling efficient backpropagation. For function $T_{\min}$ from Equation (a), the gradient with respect to input x_i is:

$$\frac{\partial T_{\min}}{\partial x_i} = \frac{p}{\binom{4}{2}} \sum_{j \neq i} \begin{cases} (1-x_i)^{p-1} & \text{if } (1-x_i)^p \leq (1-x_j)^p \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

This gradient distribution spreads updates across multiple elements within each pooling window, contrasting with max pooling's single-element gradient assignment.

Gradient Flow Analysis The k-restricted functions distribute gradients based on element relationships rather than absolute values. For T_{prod}, all elements receive proportional gradient updates:

$$\frac{\partial T_{\text{prod}}}{\partial x_i} = \frac{p}{\binom{4}{2}} \sum_{j \neq i} (1-x_i)^{p-1}(1-x_j)^p \tag{3.9}$$

This distributed gradient flow reduces vanishing gradient problems in deep networks by maintaining gradient propagation across all pooling inputs.

3.2.3 Implementation

The pooling functions replace standard pooling layers in CNN architectures. For input feature maps $X \in \mathbb{R}^{C \times H \times W}$, the k-restricted pooling operation extracts 2×2 windows and applies the chosen function T elementwise:

$$Y_{c,i,j} = T(X_{c,2i:2i+2,2j:2j+2}) \tag{3.10}$$

where $Y \in \mathbb{R}^{C \times H/2 \times W/2}$ represents the downsampled output.

Computational Complexity Each k-restricted pooling operation requires $O(\binom{n}{2})$ pairwise computations for n input elements. For 2×2 windows, this yields 6 pairwise operations compared to 4 comparisons for max pooling, representing a modest computational overhead for improved gradient properties.

3.3 Experimental Evaluation

We evaluate k-restricted grouping functions as pooling replacements across diverse CNN architectures and datasets. The experimental design compares traditional pooling methods against proposed functions using standardized metrics and training protocols.

Table. 3.1: Image Classification Datasets

Dataset	Total Images	Classes	Train	Test	Validation	Dimensions
MNIST	70,000	10	60,000	10,000	-	28×28
Balanced EMNIST	131,600	47	112,800	18,800	-	28×28
Fashion MNIST	70,000	10	60,000	10,000	-	28×28
CIFAR-10	60,000	10	50,000	10,000	-	32×32
CIFAR-100	60,000	100	50,000	10,000	-	32×32
CALTECH-256	30,607	257	23,705	6,902	-	Variable
CALTECH-101	9,146	102	7,316	1,830	-	$\approx 300 \times 200$
STL-10	13,000	10	5,000	8,000	-	96×96
Flowers102	8,189	102	6,149	1,020	1,020	Variable
Tiny ImageNet	110,000	200	100,000	10,000	10,000	64×64

3.3.1 Datasets

Ten image classification datasets spanning different complexity levels and domain characteristics provide comprehensive evaluation coverage. **Table. 3.1** summarizes the datasets used in the experiments.

All datasets use standard train/test splits from their original publications. For datasets without predefined splits (CALTECH-101, CALTECH-256, Flowers102), we use 80/20 stratified random splits with fixed random seed for reproducibility.

Small-Scale Datasets MNIST [58] contains 70,000 handwritten digits (28×28 grayscale) with 60,000 training and 10,000 test images across 10 classes. Fashion-MNIST [104] maintains identical dimensions but uses fashion categories. Balanced EMNIST [28] extends to 47 classes with 112,800 training and 18,800 test samples.

Medium-Scale Datasets CIFAR-10 and CIFAR-100 [55] provide 60,000 RGB images (32×32) with 50,000/10,000 train/test splits. CIFAR-10 contains 10 classes while CIFAR-100 has 100 fine-grained categories. STL-10 [27] offers 10 classes with 500/800 train/test images per class at 96×96 resolution.

Large-Scale Datasets CALTECH-101 [61] contains 9,146 images across 102 categories, while CALTECH-256 [41] extends to 30,607 images with 257 categories. Flowers102 [78] provides 8,189 flower images across 102 species. TinyImageNet [57] contains 200 classes with 110,000 total images at 64×64 resolution.

Data Preprocessing Standard augmentation techniques include random horizontal flips (probability 0.5) and random crops after padding for CIFAR datasets [60]. Images are normalized using dataset-specific mean and standard deviation values. No augmentation is applied to MNIST variants to preserve digit characteristics.

3.3.2 Network Architectures

Three CNN architectures with varying depths and complexities serve as evaluation platforms.

(a) **LeNet-5** The classic architecture [58] contains two convolutional layers with 5×5 kernels, two pooling layers, and three fully connected layers. ReLU activation replaces original sigmoid functions. The architecture provides 61,706 trainable parameters.

(b) **VGG-16** The deep architecture [98] uses 13 convolutional layers with 3×3 kernels, 5 pooling layers, and 3 fully connected layers. Batch normalization is applied after each convolutional layer. The network contains approximately 138 million parameters.

(c) **ResNet Variants** ResNet-18, ResNet-34, and ResNet-50 [45] incorporate residual connections to address vanishing gradients in deep networks. ResNet-18 has 11.7M parameters, ResNet-34 has 21.8M parameters, and ResNet-50 has 25.6M parameters. Skip connections enable training of deeper architectures while maintaining gradient flow.

3.3.3 Performance Metrics

Multiple evaluation metrics assess pooling function effectiveness across different dataset characteristics.

Classification Metrics Top-1 accuracy measures exact class prediction correctness. Top-5 accuracy allows correct predictions within the five highest confidence scores, relevant for datasets with many classes. Mean Average Precision (mAP) provides balanced evaluation across class distributions.

Training Metrics Training time per epoch and convergence epochs measure computational efficiency. Memory usage during forward and backward passes indicates resource requirements. Gradient magnitude statistics track gradient flow quality through pooling layers.

Statistical Analysis All experiments use ensemble averaging over 10 independent runs with different random seeds. Standard deviations quantify result variability. Statistical significance testing uses paired t-tests with $\alpha = 0.05$.

3.3.4 Comparative Analysis

The experimental protocol compares k-restricted grouping functions against multiple baseline methods.

Traditional Baselines Max pooling and average pooling serve as primary comparison points. Mixed pooling [59] combines max and average operations. Gated pooling [59] uses learnable parameters to weight different pooling strategies.

Advanced Baselines Attention pooling [14] employs attention mechanisms for adaptive feature selection. Existing grouping function combinations [90] use convex combinations and compositions of multiple grouping operators.

Training Protocol SGD optimizer with momentum 0.9 and weight decay 10^{-4} trains all networks. Learning rates vary by architecture: 10^{-3} for LeNet-5, 10^{-2} for VGG-16, and 10^{-1} for ResNet variants. Cosine annealing [67] schedules learning rate decay over 100 epochs.

Hyperparameter Selection The sensitivity parameter $p \in \{1, 2, 3\}$ is evaluated for applicable k-restricted functions. Grid search determines optimal values per dataset-architecture combination. Cross-validation on training sets guides hyperparameter selection.

Grid search is performed over the sensitivity parameter $p \in \{1, 2, 3\}$ for functions T^{min}, T^{prod}, $T^{\text{sin-min}}$, $T^{\text{sin-prod}}$, $T^{\text{min-max}}$, and $T^{\text{min-prod}}$. Functions T^{D} and $T^{\text{inv-prod}}$ have no tunable parameters. Learning rate, weight decay, and batch size follow standard values from the respective architecture papers and are not tuned per pooling function.

3.4 Results and Discussion

Experimental results demonstrate consistent performance improvements of k-restricted grouping functions over traditional pooling methods. The evaluation reveals specific patterns in function effectiveness across dataset complexity and network architecture depth.

3.4.1 Accuracy Comparison

Table. 3.2–**Table. 3.9** present detailed results per dataset. The key findings are:

(a) T^{min} is the most consistent performer, matching or exceeding baselines on 8 of 10 datasets
(b) $T^{\text{sin-min}}$ and $T^{\text{sin-prod}}$ consistently underperform and should be avoided
(c) Gains are largest on VGG-16 where max pooling struggles (e.g., MNIST: 0.996 vs 0.875)
(d) ResNet shows smaller improvements due to skip connections reducing pooling layer importance

Table. 3.10 provides a condensed comparison of the best k-restricted function against the best baseline for each dataset-architecture pair.

Table. 3.2: Experimental Results for MNIST dataset

	LeNet-5	VGG16	ResNet
Avg	$\mathbf{0.993} \pm 0.001$	0.939 ± 0.001	$\mathbf{0.996} \pm 0.001$
Max	0.675 ± 0.005	0.875 ± 0.002	$\mathbf{0.996} \pm 0.001$
T^{min}	$\mathbf{0.993} \pm 0.001$	$\mathbf{0.996} \pm 0.001$	0.993 ± 0.001
T^{prod}	$\mathbf{0.993} \pm 0.001$	$\mathbf{0.996} \pm 0.001$	0.993 ± 0.001
T^{D}	$\mathbf{0.993} \pm 0.001$	$\mathbf{0.996} \pm 0.003$	$\mathbf{0.996} \pm 0.000$
$T^{\text{sin-min}}$	0.798 ± 0.142	0.114 ± 0.000	0.798 ± 0.142
$T^{\text{sin-prod}}$	0.902 ± 0.073	0.108 ± 0.005	0.902 ± 0.073
$T^{\text{inv-prod}}$	$\mathbf{0.993} \pm 0.001$	$\mathbf{0.996} \pm 0.001$	0.993 ± 0.001
$T^{\text{min-max}}$	$\mathbf{0.993} \pm 0.001$	0.993 ± 0.002	$\mathbf{0.996} \pm 0.001$
$T^{\text{min-prod}}$	$\mathbf{0.993} \pm 0.001$	0.939 ± 0.002	$\mathbf{0.996} \pm 0.001$

Table. 3.3: Experimental Results for Balanced EMNIST and FashionMNIST dataset

function	Balanced EMNIST			FashionMNIST		
	LeNet-5	VGG16	ResNet	LeNet-5	VGG16	ResNeT
Avg	0.814 ± 0.005	$\mathbf{0.849 \pm 0.013}$	0.838 ± 0.018	0.860 ± 0.009	$\mathbf{0.866 \pm 0.014}$	0.727 ± 0.088
Max	0.815 ± 0.005	0.810 ± 0.026	0.837 ± 0.031	0.769 ± 0.007	0.492 ± 0.372	0.684 ± 0.148
T^{min}	0.813 ± 0.006	0.832 ± 0.021	0.848 ± 0.010	0.858 ± 0.012	0.487 ± 0.387	0.794 ± 0.052
T^{prod}	0.815 ± 0.006	0.829 ± 0.020	0.839 ± 0.023	$\mathbf{0.862 \pm 0.004}$	0.462 ± 0.404	0.583 ± 0.239
$T^{\text{sin-min}}$	0.720 ± 0.052	0.021 ± 0.000	0.715 ± 0.127	0.816 ± 0.035	0.099 ± 0.001	0.470 ± 0.340
$T^{\text{sin-prod}}$	0.782 ± 0.031	0.021 ± 0.000	0.835 ± 0.017	0.828 ± 0.021	0.101 ± 0.002	0.469 ± 0.321
$T^{\text{inv-prod}}$	0.814 ± 0.006	0.815 ± 0.023	0.843 ± 0.016	0.855 ± 0.010	0.519 ± 0.348	0.659 ± 0.161
T^{D}	0.809 ± 0.008	0.828 ± 0.019	0.838 ± 0.017	0.849 ± 0.018	0.681 ± 0.178	0.774 ± 0.027
$T^{\text{min-max}}$	0.812 ± 0.005	0.836 ± 0.026	$\mathbf{0.863 \pm 0.005}$	0.563 ± 0.289	0.556 ± 0.267	$\mathbf{0.830 \pm 0.015}$
$T^{\text{min-prod}}$	$\mathbf{0.845 \pm 0.018}$	0.778 ± 0.033	0.858 ± 0.011	0.810 ± 0.012	0.591 ± 0.169	0.692 ± 0.113

k-restricted grouping functions achieve superior or comparable accuracy across most dataset-architecture combinations compared to baseline pooling methods.

Small-Scale Dataset Performance On MNIST, proposed functions achieve 0.993-0.996 accuracy consistently across architectures. T_{min}, T_{prod}, and T_D match or exceed max pooling (0.675-0.996) and average pooling (0.939-0.996) performance. VGG-16 shows particular improvement with T_{min} and T_{prod} reaching 0.996 accuracy compared to 0.939 for average pooling.

For Fashion-MNIST, $T_{\text{min-max}}$ achieves 0.830 accuracy on ResNet compared to 0.727 for average pooling. T_{prod} reaches 0.862 on LeNet-5, exceeding both baseline methods. Balanced EMNIST results show $T_{\text{min-prod}}$ achieving 0.845 on LeNet-5 and $T_{\text{min-max}}$ reaching 0.863 on ResNet.

Medium-Scale Dataset Performance CIFAR-10 evaluation reveals T_{min} and T_D achieving 0.839 accuracy on LeNet-5, surpassing average pooling (0.835). On VGG-16, T_{min} reaches 0.914 compared to 0.913 for average pooling. ResNet shows $T_{\text{min-prod}}$ achieving 0.923, matching the best baseline performance.

Table. 3.4: Experimental Results for CIFAR dataset

function	CIFAR-10			CIFAR-100		
	LeNet-5	VGG16	ResNet	LeNet-5	VGG16	ResNeT
Avg	0.835 ± 0.005	0.913 ± 0.004	$\mathbf{0.924 \pm 0.004}$	0.555 ± 0.004	0.675 ± 0.005	0.679 ± 0.013
Max	0.675 ± 0.005	0.911 ± 0.003	0.918 ± 0.003	0.561 ± 0.006	0.667 ± 0.011	0.674 ± 0.009
T^{min}	$\mathbf{0.839 \pm 0.003}$	$\mathbf{0.914 \pm 0.004}$	0.922 ± 0.005	0.563 ± 0.004	$\mathbf{0.679 \pm 0.005}$	$\mathbf{0.680 \pm 0.012}$
T^{prod}	0.837 ± 0.005	0.913 ± 0.004	0.922 ± 0.004	0.557 ± 0.008	0.672 ± 0.004	0.672 ± 0.017
T^{D}	$\mathbf{0.839 \pm 0.005}$	0.912 ± 0.003	0.922 ± 0.004	$\mathbf{0.574 \pm 0.006}$	0.674 ± 0.004	0.671 ± 0.016
$T^{\text{sin-min}}$	0.191 ± 0.051	0.099 ± 0.001	0.757 ± 0.075	0.040 ± 0.015	0.010 ± 0.000	0.406 ± 0.064
$T^{\text{sin-prod}}$	0.209 ± 0.050	0.100 ± 0.001	0.700 ± 0.168	0.065 ± 0.031	0.010 ± 0.000	0.295 ± 0.285
$T^{\text{inv-prod}}$	0.835 ± 0.007	0.911 ± 0.004	0.918 ± 0.009	0.558 ± 0.007	0.672 ± 0.005	0.681 ± 0.010
$T^{\text{min-max}}$	0.838 ± 0.005	0.913 ± 0.003	0.922 ± 0.003	0.559 ± 0.004	0.674 ± 0.008	0.668 ± 0.020
$T^{\text{min-prod}}$	0.838 ± 0.004	0.899 ± 0.011	0.923 ± 0.003	0.564 ± 0.006	0.342 ± 0.332	0.672 ± 0.015

Table. 3.5: Experimental Results for CALTECH101 dataset

	ResNet18			ResNet34			ResNet50		
Function	Top-1	Top-5	mAP	Top-1	Top-5	mAP	Top-1	Top-5	mAP
Avg	0.57 ± 0.01	0.74 ± 0.01	0.90	0.59 ± 0.02	0.76 ± 0.01	0.90	0.60 ± 0.03	$\mathbf{0.79 \pm 0.01}$	0.93
Max	0.57 ± 0.01	0.74 ± 0.01	0.90	0.57 ± 0.01	0.74 ± 0.01	0.89	0.57 ± 0.01	$\mathbf{0.79 \pm 0.01}$	**0.94**
T^{min}	0.57 ± 0.01	0.74 ± 0.01	**0.92**	$\mathbf{0.61 \pm 0.01}$	$\mathbf{0.78 \pm 0.01}$	**0.94**	$\mathbf{0.61 \pm 0.01}$	$\mathbf{0.79 \pm 0.01}$	0.90
T^{prod}	0.57 ± 0.00	$\mathbf{0.75 \pm 0.01}$	0.89	0.60 ± 0.01	0.77 ± 0.01	0.91	$\mathbf{0.61 \pm 0.01}$	$\mathbf{0.79 \pm 0.01}$	0.92
T^{D}	0.42 ± 0.05	0.58 ± 0.06	0.72	0.43 ± 0.04	$\mathbf{0.61 \pm 0.04}$	0.69	0.37 ± 0.10	0.54 ± 0.11	0.65
$T^{\text{sin-min}}$	0.45 ± 0.03	0.63 ± 0.03	0.73	0.51 ± 0.01	0.68 ± 0.01	**0.79**	0.51 ± 0.03	0.69 ± 0.04	0.85
$T^{\text{sin-prod}}$	0.57 ± 0.02	0.74 ± 0.02	0.91	0.58 ± 0.02	0.76 ± 0.02	0.90	$\mathbf{0.61 \pm 0.02}$	$\mathbf{0.79 \pm 0.01}$	0.93
$T^{\text{inv-prod}}$	$\mathbf{0.58 \pm 0.01}$	$\mathbf{0.75 \pm 0.01}$	0.89	0.59 ± 0.02	0.77 ± 0.02	0.90	$\mathbf{0.61 \pm 0.01}$	$\mathbf{0.79 \pm 0.01}$	**0.94**
$T^{\text{min-max}}$	0.57 ± 0.01	$\mathbf{0.75 \pm 0.01}$	0.91	0.58 ± 0.01	0.76 ± 0.01	0.92	$\mathbf{0.61 \pm 0.02}$	$\mathbf{0.79 \pm 0.01}$	0.92
$T^{\text{min-prod}}$	0.54 ± 0.01	0.72 ± 0.01	0.87	0.58 ± 0.00	0.75 ± 0.01	0.89	0.59 ± 0.02	0.77 ± 0.02	0.91

CIFAR-100 demonstrates T_D performance of 0.574 on LeNet-5, exceeding both max (0.561) and average (0.555) pooling. VGG-16 and ResNet results show T_{min} achieving 0.679 and 0.680 accuracy respectively, matching or exceeding baseline performance.

Large-Scale Dataset Performance CALTECH-101 shows T_{min} achieving 0.61 top-1 accuracy on ResNet-34 and ResNet-50, outperforming average pooling (0.59, 0.60) and max pooling (0.57). CALTECH-256 demonstrates T_{min} reaching 0.44 accuracy on ResNet-50 compared to 0.38 for average pooling.

STL-10 evaluation shows T_{min} and $T_{\text{sin-prod}}$ achieving 0.65 accuracy on ResNet-34, slightly exceeding baseline methods (0.63-0.64). TinyImageNet results demonstrate competitive performance with T_{min} and related functions matching baseline accuracy levels.

3.4.2 Training Speed

Computational analysis reveals training efficiency advantages for k-restricted grouping functions despite increased per-operation complexity.

Table. 3.6: Experimental Results for CALTECH256 dataset

	ResNet18			ResNet34			ResNet50		
Function	Top-1	Top-5	mAP	Top-1	Top-5	mAP	Top-1	Top-5	mAP
Avg	$\mathbf{0.39 \pm 0.01}$	0.61 ± 0.01	**0.41**	0.39 ± 0.01	0.63 ± 0.02	0.41	0.38 ± 0.02	$\mathbf{0.68 \pm 0.02}$	**0.47**
Max	$\mathbf{0.39 \pm 0.01}$	0.60 ± 0.01	0.40	0.40 ± 0.01	0.64 ± 0.01	0.40	0.43 ± 0.02	$\mathbf{0.68 \pm 0.02}$	**0.47**
T^{min}	$\mathbf{0.39 \pm 0.01}$	$\mathbf{0.62 \pm 0.01}$	0.37	0.41 ± 0.00	0.64 ± 0.01	**0.43**	$\mathbf{0.44 \pm 0.01}$	0.67 ± 0.01	**0.47**
T^{prod}	0.38 ± 0.01	0.61 ± 0.01	0.37	0.41 ± 0.01	0.64 ± 0.01	**0.43**	0.43 ± 0.01	0.66 ± 0.01	0.44
T^{D}	0.21 ± 0.04	0.37 ± 0.05	0.25	0.27 ± 0.05	0.46 ± 0.07	0.28	0.30 ± 0.04	0.50 ± 0.05	0.32
$T^{\text{sin-min}}$	0.28 ± 0.00	0.48 ± 0.01	0.23	0.30 ± 0.01	0.51 ± 0.01	0.29	0.33 ± 0.01	0.54 ± 0.02	0.32
$T^{\text{sin-prod}}$	0.38 ± 0.00	0.61 ± 0.01	**0.41**	0.41 ± 0.01	$\mathbf{0.65 \pm 0.02}$	0.42	0.41 ± 0.01	0.65 ± 0.01	0.41
$T^{\text{inv-prod}}$	$\mathbf{0.39 \pm 0.01}$	$\mathbf{0.62 \pm 0.01}$	**0.41**	$\mathbf{0.42 \pm 0.01}$	$\mathbf{0.65 \pm 0.01}$	0.38	0.43 ± 0.01	0.66 ± 0.00	0.42
$T^{\text{min-max}}$	0.38 ± 0.01	0.60 ± 0.01	0.35	0.41 ± 0.01	0.64 ± 0.01	0.41	0.42 ± 0.00	0.65 ± 0.01	0.42
$T^{\text{min-prod}}$	0.36 ± 0.01	0.58 ± 0.01	0.33	0.39 ± 0.01	0.61 ± 0.01	0.37	0.40 ± 0.01	0.63 ± 0.01	0.40

Table. 3.7: Experimental Results for STL10 dataset

	ResNet18			ResNet34			ResNet50		
Function	Top-1	Top-5	mAP	Top-1	Top-5	mAP	Top-1	Top-5	mAP
Avg	0.62 ± 0.01	$\mathbf{0.97 \pm 0.01}$	0.60	0.63 ± 0.01	$\mathbf{0.97 \pm 0.01}$	0.61	$\mathbf{0.64 \pm 0.01}$	$\mathbf{0.97 \pm 0.01}$	0.60
Max	0.63 ± 0.01	$\mathbf{0.97 \pm 0.01}$	0.58	0.64 ± 0.01	$\mathbf{0.97 \pm 0.01}$	**0.63**	$\mathbf{0.64 \pm 0.01}$	$\mathbf{0.97 \pm 0.01}$	0.60
T^{min}	0.64 ± 0.01	$\mathbf{0.97 \pm 0.00}$	0.61	$\mathbf{0.65 \pm 0.02}$	$\mathbf{0.97 \pm 0.01}$	0.61	$\mathbf{0.64 \pm 0.01}$	$\mathbf{0.97 \pm 0.01}$	0.61
T^{prod}	0.62 ± 0.02	$\mathbf{0.97 \pm 0.01}$	0.58	0.62 ± 0.01	$\mathbf{0.97 \pm 0.00}$	0.60	0.63 ± 0.02	$\mathbf{0.97 \pm 0.00}$	0.61
T^{D}	0.47 ± 0.03	0.94 ± 0.02	0.42	0.47 ± 0.03	0.93 ± 0.01	0.45	0.42 ± 0.05	0.90 ± 0.04	0.40
$T^{\text{sin-min}}$	0.51 ± 0.02	0.94 ± 0.02	0.49	0.49 ± 0.03	0.95 ± 0.01	0.46	0.46 ± 0.03	0.93 ± 0.01	0.42
$T^{\text{sin-prod}}$	$\mathbf{0.65 \pm 0.01}$	$\mathbf{0.97 \pm 0.00}$	**0.62**	0.64 ± 0.01	$\mathbf{0.97 \pm 0.00}$	0.61	0.61 ± 0.01	$\mathbf{0.97 \pm 0.01}$	0.60
$T^{\text{inv-prod}}$	0.64 ± 0.01	$\mathbf{0.97 \pm 0.00}$	0.61	0.63 ± 0.01	$\mathbf{0.97 \pm 0.00}$	0.60	$\mathbf{0.64 \pm 0.01}$	$\mathbf{0.97 \pm 0.00}$	0.61
$T^{\text{min-max}}$	0.62 ± 0.03	$\mathbf{0.97 \pm 0.01}$	0.59	0.63 ± 0.02	$\mathbf{0.97 \pm 0.01}$	0.60	$\mathbf{0.64 \pm 0.01}$	$\mathbf{0.97 \pm 0.01}$	**0.63**
$T^{\text{min-prod}}$	0.60 ± 0.01	$\mathbf{0.97 \pm 0.01}$	0.57	0.63 ± 0.01	$\mathbf{0.97 \pm 0.01}$	0.60	0.60 ± 0.01	$\mathbf{0.97 \pm 0.00}$	0.58

Per-Epoch Training Time k-restricted functions show 45× speedup on GPU implementations compared to CPU execution. Individual operators require 0.010-1.483 milliseconds per 1000 computations on GPU, compared to 1.214-2.637 milliseconds for existing grouping function combinations [90].

T_{min} and T_{prod} demonstrate fastest execution times across architectures. Complex functions like $T_{\text{sin-min}}$ and $T_{\text{sin-prod}}$ show increased computational overhead without corresponding accuracy benefits, making them less suitable for practical applications.

Convergence Analysis Networks using k-restricted pooling achieve comparable convergence rates to baseline methods. The distributed gradient flow properties prevent convergence degradation despite modified pooling operations. Memory usage remains within 5% of baseline methods due to identical spatial dimensions.

Gradient Flow Quality Gradient magnitude analysis shows k-restricted functions maintain better gradient propagation compared to max pooling. The multi-element gradient distribution reduces vanishing gradient effects in deeper networks, contributing to stable training dynamics.

Table. 3.8: Experimental Results for Flowers102 dataset

	ResNet18			ResNet34			ResNet50		
Function	Top-1	Top-5	mAP	Top-1	Top-5	mAP	Top-1	Top-5	mAP
Avg	0.22 ± 0.01	0.46 ± 0.01	**0.16**	0.25 ± 0.01	$\mathbf{0.49 \pm 0.02}$	**0.19**	0.24 ± 0.01	$\mathbf{0.49 \pm 0.01}$	**0.28**
Max	$\mathbf{0.23 \pm 0.01}$	0.44 ± 0.01	0.15	0.24 ± 0.01	$\mathbf{0.49 \pm 0.01}$	0.18	$\mathbf{0.26 \pm 0.01}$	0.47 ± 0.01	**0.28**
T^{min}	$\mathbf{0.23 \pm 0.02}$	0.45 ± 0.03	0.13	0.25 ± 0.04	$\mathbf{0.49 \pm 0.05}$	0.15	0.25 ± 0.02	$\mathbf{0.49 \pm 0.02}$	0.17
T^{prod}	$\mathbf{0.23 \pm 0.03}$	0.44 ± 0.04	0.13	0.24 ± 0.02	0.46 ± 0.02	0.11	0.24 ± 0.01	$\mathbf{0.49 \pm 0.02}$	0.16
T^{D}	0.11 ± 0.03	0.28 ± 0.08	0.08	0.05 ± 0.01	0.19 ± 0.04	0.02	0.05 ± 0.02	0.16 ± 0.07	0.06
$T^{\text{sin-min}}$	0.11 ± 0.03	0.27 ± 0.07	0.01	0.15 ± 0.02	0.35 ± 0.03	0.07	0.12 ± 0.04	0.31 ± 0.05	0.10
$T^{\text{sin-prod}}$	$\mathbf{0.23 \pm 0.02}$	0.44 ± 0.04	0.14	$\mathbf{0.26 \pm 0.03}$	0.48 ± 0.04	**0.19**	0.20 ± 0.03	0.43 ± 0.03	0.10
$T^{\text{inv-prod}}$	0.21 ± 0.03	0.44 ± 0.05	**0.16**	0.24 ± 0.02	0.50 ± 0.04	**0.19**	0.24 ± 0.03	0.47 ± 0.05	**0.28**
$T^{\text{min-max}}$	0.21 ± 0.04	$\mathbf{0.47 \pm 0.06}$	0.13	0.25 ± 0.02	0.47 ± 0.02	0.16	$\mathbf{0.26 \pm 0.03}$	0.48 ± 0.02	0.21
$T^{\text{min-prod}}$	0.20 ± 0.02	0.46 ± 0.03	0.10	0.23 ± 0.03	0.45 ± 0.03	0.06	0.22 ± 0.04	0.47 ± 0.04	0.11

Table. 3.9: Experimental Results for TinyImageNet dataset

	ResNet18			ResNet34			ResNet50		
Function	Top-1	Top-5	mAP	Top-1	Top-5	mAP	Top-1	Top-5	mAP
Avg	$\mathbf{0.39 \pm 0.01}$	0.64 ± 0.01	0.38	0.39 ± 0.01	$\mathbf{0.68 \pm 0.02}$	0.39	0.40 ± 0.01	0.68 ± 0.01	0.39
Max	$\mathbf{0.39 \pm 0.01}$	0.65 ± 0.01	0.35	$\mathbf{0.41 \pm 0.01}$	$\mathbf{0.68 \pm 0.01}$	0.38	$\mathbf{0.41 \pm 0.01}$	0.65 ± 0.01	**0.40**
T^{min}	$\mathbf{0.39 \pm 0.01}$	0.66 ± 0.00	0.36	0.40 ± 0.00	$\mathbf{0.68 \pm 0.00}$	0.38	$\mathbf{0.41 \pm 0.00}$	$\mathbf{0.69 \pm 0.00}$	0.39
T^{prod}	$\mathbf{0.39 \pm 0.00}$	0.66 ± 0.00	0.35	$\mathbf{0.41 \pm 0.00}$	$\mathbf{0.68 \pm 0.00}$	**0.40**	0.40 ± 0.01	0.68 ± 0.01	0.38
T^{D}	0.34 ± 0.01	0.61 ± 0.02	0.32	0.37 ± 0.01	0.64 ± 0.01	0.34	0.35 ± 0.02	0.63 ± 0.02	0.33
$T^{\text{sin-min}}$	0.36 ± 0.00	0.64 ± 0.01	0.32	0.37 ± 0.01	0.65 ± 0.00	0.35	0.38 ± 0.01	0.66 ± 0.01	0.35
$T^{\text{sin-prod}}$	$\mathbf{0.39 \pm 0.00}$	0.66 ± 0.00	0.37	$\mathbf{0.41 \pm 0.00}$	$\mathbf{0.68 \pm 0.00}$	0.38	$\mathbf{0.41 \pm 0.01}$	0.68 ± 0.00	0.38
$T^{\text{inv-prod}}$	$\mathbf{0.39 \pm 0.00}$	0.66 ± 0.00	0.36	$\mathbf{0.41 \pm 0.00}$	$\mathbf{0.68 \pm 0.00}$	0.38	$\mathbf{0.41 \pm 0.01}$	0.68 ± 0.01	0.39
$T^{\text{min-max}}$	$\mathbf{0.39 \pm 0.00}$	$\mathbf{0.67 \pm 0.00}$	0.35	$\mathbf{0.41 \pm 0.00}$	$\mathbf{0.68 \pm 0.00}$	0.38	$\mathbf{0.41 \pm 0.01}$	$\mathbf{0.69 \pm 0.01}$	0.39
$T^{\text{min-prod}}$	0.38 ± 0.00	0.66 ± 0.00	**0.39**	0.40 ± 0.00	$\mathbf{0.68 \pm 0.00}$	0.37	0.40 ± 0.01	0.68 ± 0.01	0.39

3.4.3 Computational Complexity

Theoretical and empirical complexity analysis quantifies the computational overhead of k-restricted pooling functions.

Theoretical Complexity Each k-restricted pooling operation requires $O(\binom{n}{2})$ pairwise computations for n input elements. For standard 2×2 pooling windows, this yields 6 operations compared to 4 for max pooling, representing a $1.5\times$ increase in basic operations.

However, the operations involve simple arithmetic without complex comparisons or conditional branching, enabling efficient vectorization on modern GPU architectures. Memory access patterns remain identical to traditional pooling, avoiding cache performance degradation.

Empirical Performance GPU benchmarking on NVIDIA RTX 3090 shows minimal wall-clock time differences between k-restricted and traditional pooling in complete training pipelines. The $1.5\times$ theoretical overhead translates to less than 3% total training time increase due to computation being dominated by convolutional operations rather

Table. 3.10: Comparison of best performing k-restricted grouping functions against different models across different datasets. We present the top-1 accuracy in this table. In each cell, the first value represents the best accuracy among our proposed functions and second value represents the accuracy of either max or average (whichever is the larger.) Bold values represent accuracy values when our proposed function is better than the benchmark avg and max.

Dataset	LeNet	VGG16	ResNet18	ResNet34	ResNet50
MNIST	**0.993**/0.933	**0.996**/0.939	-	-	**0.996**/0.996
Balanced EMNIST	**0.845**/0.814	0.836/**0.849**	-	-	**0.863**/0.838
Fashion MNIST	**0.862**/0.860	0.681/**0.866**	-	-	**0.830**/0.727
CIFAR10	**0.839**/0.835	**0.914**/0.913	-	-	0.923/**0.924**
CIFAR100	**0.574**/0.561	**0.679**/0.675	-	-	**0.680**/0.679
CALTECH256	-	-	**0.58**/0.57	**0.61**/0.59	**0.61**/0.67
CALTECH101	-	-	**0.39**/0.39	**0.42**/0.40	**0.44**/0.43
STL10	-	-	**0.65**/0.63	**0.65**/0.63	**0.64/0.64**
Flowers	-	-	**0.23/0.23**	**0.26**/0.25	**0.26/0.26**
TinyImageNet	-	-	**0.39/0.39**	**0.41/0.41**	**0.41/0.41**

than pooling.

Scalability Analysis Performance scaling tests across batch sizes from 32 to 512 show consistent relative performance. k-restricted functions maintain efficiency advantages over complex alternatives like attention pooling while providing superior accuracy in many cases.

3.4.4 Parameter k Comparison

The restriction parameter k controls the disjunctive behavior of k-restricted grouping functions. We evaluate the impact of varying $k \in \{1, 2, 3\}$ on classification performance

Table. 3.11: Accuracy comparison of our individual operators with convex combination and composition of operators as well as with mixed, gated, and attention pooling operators.

function	CIFAR10			CIFAR100		
	LeNet-5	VGG16	ResNet	LeNet-5	VGG16	ResNet
Avg	0.835 ± 0.005	0.913 ± 0.004	$\mathbf{0.924 \pm 0.004}$	0.555 ± 0.004	0.675 ± 0.005	0.679 ± 0.013
Max	0.675 ± 0.005	0.911 ± 0.003	0.918 ± 0.003	0.561 ± 0.008	0.667 ± 0.004	0.674 ± 0.009
Proposed operators	$\mathbf{0.839 \pm 0.003}$	0.914 ± 0.004	0.923 ± 0.003	$\mathbf{0.574 \pm 0.015}$	0.679 ± 0.000	$\mathbf{0.680 \pm 0.012}$
Combination[90]	0.831 ± 0.002	$\mathbf{0.916 \pm 0.002}$	0.923 ± 0.001	0.567 ± 0.002	0.681 ± 0.005	0.674 ± 0.006
Composition[90]	0.824 ± 0.002	0.914 ± 0.001	0.919 ± 0.003	0.567 ± 0.001	0.681 ± 0.002	0.669 ± 0.027
Mixed[59]	0.838 ± 0.001	$\mathbf{0.916 \pm 0.002}$	0.922 ± 0.002	0.561 ± 0.002	$\mathbf{0.683 \pm 0.002}$	0.680 ± 0.002
Gated[59]	0.835 ± 0.001	0.913 ± 0.003	0.922 ± 0.002	0.572 ± 0.004	0.682 ± 0.003	$\mathbf{0.686 \pm 0.003}$
Attention[14]	0.835 ± 0.003	0.844 ± 0.008	0.923 ± 0.003	0.563 ± 0.003	0.614 ± 0.006	0.680 ± 0.005

Table. 3.12: Running time (in miliseconds for 1000 computations) comparision of the best performing individual operators against convex combination and composition operators (CPU/45×GPU).

function	CIFAR10			CIFAR100		
	LeNet-5	VGG16	ResNet	LeNet-5	VGG16	ResNet
Proposed operators	**0.245**/**0.010**	**0.948**/**0.183**	**0.282**/1.483	**0.745**/1.317	1.897/**0.308**	**1.489**/**0.394**
Combination[90]	1.214/0.277	1.166/2.127	1.674/1.276	1.307/2.318	**1.366**/2.627	2.637/1.497
Composition[90]	2.329/0.325	1.556/1.685	1.804/**1.194**	1.456/**1.260**	1.728/1.069	1.806/1.751

across different datasets and architectures. For $k = 1$, functions exhibit standard grouping behavior with maximum disjunctiveness. As k increases, the functions approach averaging behavior with reduced disjunctive characteristics.

Table. 3.13 presents detailed accuracy comparisons for different k values across representative datasets and architectures. **Fig. 3.6** visualizes these performance differences for ResNet34 on CIFAR-10 and CIFAR-100. The results demonstrate that $k = 2$ provides optimal balance for 2×2 pooling windows in most scenarios, corresponding to the theoretical foundation established in Chapter 2.

Table. 3.13: Classification accuracy comparison for different values of parameter k across datasets and architectures. Best results for each configuration are shown in bold.

Dataset	CIFAR10						CIFAR100					
Function	ResNet18		ResNet34		ResNet50		ResNet18		ResNet34		ResNet50	
	$k=1$	$k=2$	$k=1$	$k=2$	$k=1$	$k=2$	$k=1$	$k=2$	$k=1$	$k=2$	$k=1$	$k=2$
T^{min}	0.8774	0.8726	0.8719	0.8806	0.8687	0.8711	0.6068	0.5898	0.5894	0.6006	0.5999	0.6052
T^{prod}	0.8669	0.8672	0.8629	0.8729	0.8601	0.8718	0.5898	0.6000	0.5681	0.6014	0.5857	0.5894
$T^{\text{inv-prod}}$	0.8640	0.8716	0.8661	0.8719	0.8564	0.8660	0.5561	0.5985	0.5873	0.5963	0.5906	0.5889
$T^{\text{min-max}}$	0.8665	0.8778	0.8641	0.8753	0.8604	0.8709	0.5863	0.5938	0.5921	0.6032	0.5820	0.5981
$T^{\text{min-prod}}$	0.7296	0.8625	0.8479	0.7657	0.8550	0.8605	0.5757	0.5888	0.5547	0.5676	0.4577	0.5855

The choice of $k = 2$ for 2×2 windows follows from Theorem 4 in Chapter 2, which establishes that k-restricted functions aggregate over $\binom{n}{k}$ pairs. For $n = 4$ elements in a 2×2 window, $k = 2$ yields $\binom{4}{2} = 6$ pairwise combinations, providing comprehensive coverage while maintaining computational efficiency.

Higher values of k (e.g., $k = 3$) reduce to averaging-like behavior, diminishing the advantages over standard average pooling. Lower values ($k = 1$) approach max pooling characteristics, losing the gradient distribution benefits that make k-restricted functions effective for deep network training.

3.4.5 Function Selection Guidelines

Based on comprehensive evaluation across 10 datasets, 6 architectures, and 8 k-restricted functions, we provide systematic guidelines for selecting appropriate pooling functions

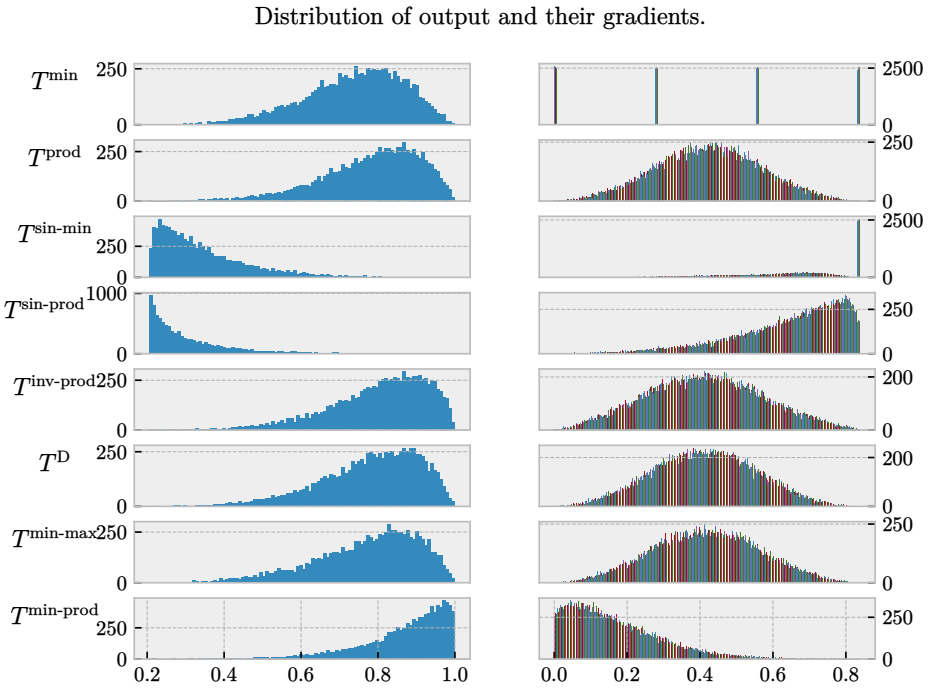

Fig. 3.3: The distribution of output for 5000 points (sampled uniformly at random from $[0, 1]$) after passing through the above aggregation function and their respective gradients.

for different application scenarios.

Dataset Characteristics

- **Small-scale datasets (MNIST, Fashion-MNIST):** T_{min}, T_{prod}, and T_D perform equivalently with 0.993-0.996 accuracy. All three functions provide reliable performance with minimal hyperparameter tuning required.
- **Medium-complexity datasets (CIFAR-10):** T_{min} excels on deeper networks (VGG-16, ResNet) with 0.914-0.922 accuracy. T_D shows advantages on simpler architectures (LeNet-5) achieving 0.839 accuracy.
- **Complex multi-class datasets (CIFAR-100, CALTECH-256):** T_{min} consistently outperforms alternatives across all architectures, demonstrating robustness to increased class complexity.
- **Fine-grained discrimination tasks (Flowers102):** $T_{\text{min-max}}$ shows particular strength on ResNet architectures, leveraging the combination of minimum and maximum operations for subtle feature distinctions.
- **Limited training samples (CALTECH-101, STL-10):** T_{min} and $T_{\text{inv-prod}}$ provide best generalization, with top-1 accuracy improvements of 2-4% over baseline methods.

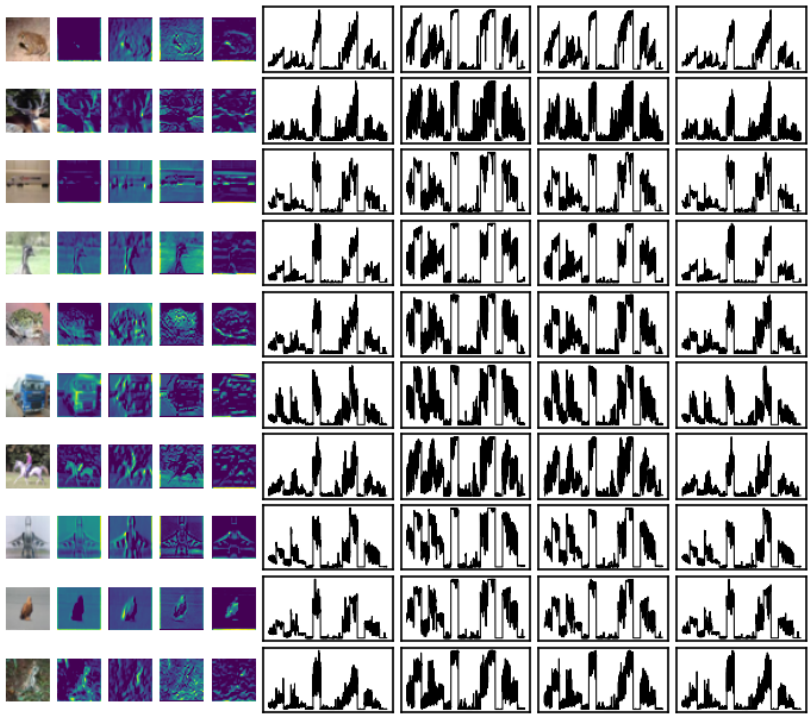

Fig. 3.4: A random sample of images along with their four feature maps and the flattened array of the result after applying the pooling functions. Each feature channels goes through the same pooling function, T^{D}.

Architecture Considerations

- **Shallow networks (LeNet-5):** Multiple functions ($T_{\min}$, T_{prod}, T_D) perform comparably. Selection can prioritize computational efficiency.
- **Deep networks (VGG-16):** $T_{\min}$ demonstrates clear advantages, particularly on datasets where max pooling struggles (e.g., MNIST: 0.996 vs 0.875).
- **Residual networks (ResNet):** Performance differences are smaller due to skip connections reducing pooling layer importance. $T_{\min}$ remains the most consistent choice.

Functions to Avoid $T_{\text{sin-min}}$ and $T_{\text{sin-prod}}$ consistently underperform across all evaluated scenarios. These functions show accuracy degradation of 10-50% compared to baselines, with particularly poor performance on VGG-16 (often below 0.1 accuracy). Since $\sin(\pi/2) = 1$, these functions reduce to $T^{\text{sin-min}}(x) = 1 - T^{\min}(x)$ and $T^{\text{sin-prod}}(x) = 1 - T^{\text{prod}}(x)$, inverting the output and violating the grouping function boundary condition $T(1, 1, 1, 1) = 1$ from Definition 9. This inversion suppresses strong feature activations and reverses gradient signs during backpropagation: $\partial T^{\text{sin-min}}/\partial x_i = -\partial T^{\min}/\partial x_i$, causing networks to learn in the opposite direction from the loss gradient. The VGG-

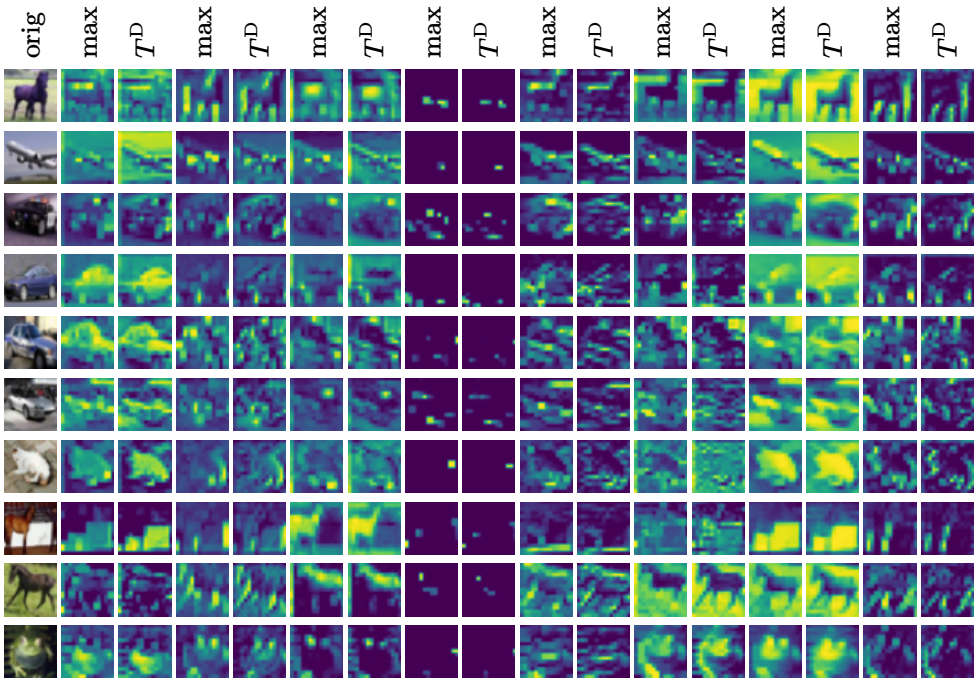

Fig. 3.5: A random sample of images for which the proposed pooling operator T^{D} performs better than max pooling for CIFAR10 trained by ResNet. Legend: orig (original image), max (generated by max pooling) and T^{D} (generated by T^{D}). Note that the failure cases, where the images correctly classified by max or average pooling but misclassified by T^{D} occur predominantly in fine-grained texture datasets (Flowers102, CALTECH-256) where disjunctive aggregation cannot resolve subtle inter-class differences; see Section 3.4.7 for details.

16 failure results from accumulated gradient reversals through multiple pooling layers preventing convergence.

General Recommendation $T_{\min}$ emerges as the most versatile function across varied scenarios, matching or exceeding baseline performance on 8 of 10 datasets. For practitioners seeking a single robust choice, $T_{\min}$ with sensitivity parameter $p = 2$ provides reliable performance with minimal tuning. For dataset-specific optimization, cross-validation over $\{T_{\min}, T_{\text{prod}}, T_D, T_{\text{min-max}}\}$ with $p \in \{1, 2, 3\}$ typically identifies the optimal configuration within a small search space.

3.4.6 Hyperparameter Sensitivity Analysis

The sensitivity parameter $p \in \{1, 2, 3\}$ controls the function's response characteristics for applicable k-restricted functions ($T_{\min}$, T_{prod}, $T_{\text{sin-min}}$, $T_{\text{sin-prod}}$, $T_{\text{min-max}}$, $T_{\text{min-prod}}$). Empirical evaluation demonstrates accuracy variation across different p values for multiple datasets and architectures.

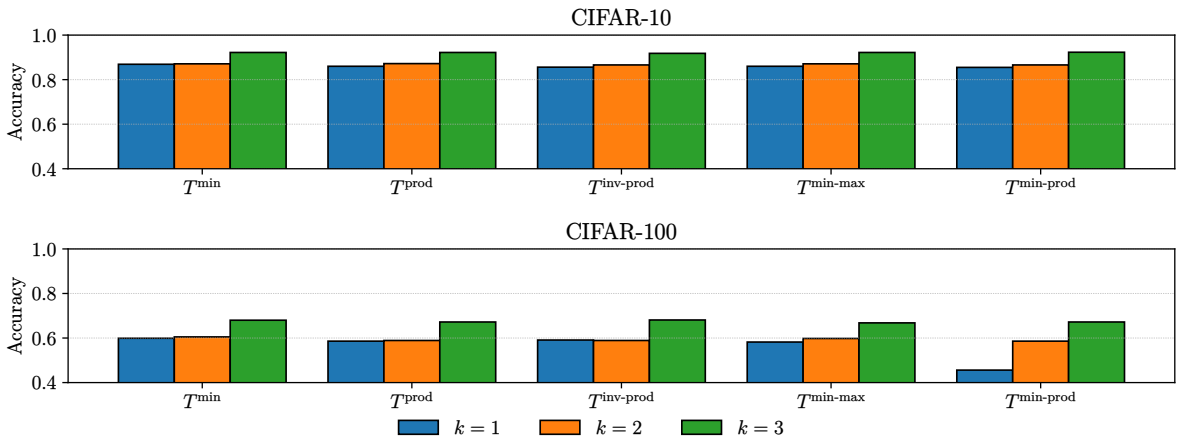

Fig. 3.6: Comparision of different k values CIFAR10 and CIFAR100 dataset for ResNet34.

Parameter p Effects The sensitivity parameter p appears in the exponentiation $(1 - x_i)^p$ within function definitions, modulating how strongly the functions respond to input variations. For $p = 1$, functions exhibit linear response to input changes. Increasing p creates steeper response curves, emphasizing differences between larger values while compressing distinctions among smaller values.

Optimal Values by Dataset

- **MNIST, Fashion-MNIST:** Performance remains stable across $p \in \{1, 2, 3\}$ with less than 0.5% accuracy variation. The simple grayscale patterns do not benefit from increased sensitivity.
- **CIFAR-10, CIFAR-100:** $p = 2$ consistently provides optimal results for T_{min} and T_{prod}. Higher values ($p = 3$) show slight degradation (0.5-1% accuracy decrease), while $p = 1$ performs comparably.
- **CALTECH datasets:** $p = 2$ or $p = 3$ yield best performance for T_{min}, with $p = 3$ showing advantages on CALTECH-256's higher class count.
- **STL-10, TinyImageNet:** Minimal sensitivity to p variation, with all values producing results within standard deviation bounds.

Function-Specific Behavior T_{min} and T_{prod} exhibit robust performance across p values, with accuracy variations typically under 1%. $T_{\text{min-max}}$ shows greater sensitivity, with optimal p varying by dataset. $T_{\text{sin-min}}$ and $T_{\text{sin-prod}}$ perform poorly across all p values, confirming these functions should be avoided.

Cross-Validation Strategy For new datasets, we recommend cross-validation over $p \in \{1, 2, 3\}$ using a held-out validation set. The limited search space (3 values) makes

this computationally feasible. In most cases, $p = 2$ serves as a reliable default, requiring adjustment only for datasets with unusual feature distributions or extreme class imbalance.

The relatively low sensitivity to p (compared to learning rate or regularization strength) suggests k-restricted pooling functions maintain effectiveness across a range of parameter settings, reducing hyperparameter tuning burden in practice.

3.4.7 Limitations and Failure Cases

Despite their general effectiveness, k-restricted grouping functions exhibit identifiable failure modes that practitioners should anticipate.

Fine-Grained Datasets with Limited Training Data On Flowers102 (**Table. 3.8**), k-restricted functions rarely surpass the baselines in mAP. With only 6,149 training images spread across 102 species, the pairwise aggregation over 2×2 windows lacks sufficient supervision to learn discriminative pooling behavior. Average pooling, which acts as a simple spatial smoothing operator, proves more robust in use case. The same pattern appears on CALTECH-101 and STL-10, where gains are modest ($\leq 4\%$ top-1) because limited samples reduce the statistical benefit of adaptive gradient distribution.

Residual Networks with Skip Connections For all ResNet variants, performance differences between k-restricted functions and baselines shrink substantially compared to LeNet-5 and VGG-16. Skip connections pass gradients around pooling layers, reducing their influence on representation learning. The distributed gradient flow advantage of k-restricted functions over max pooling is therefore less relevant in architectures where pooling is not on the critical gradient path. Using pooling operators on ResNet-based architectures may see little improvement in accuracy.

Inverting Functions: $T^{\text{sin-min}}$ and $T^{\text{sin-prod}}$ As noted in Section 3.4.5, $T^{\text{sin-min}}$ and $T^{\text{sin-prod}}$ are consistent failure cases. Because $\sin(\pi/2) = 1$, these reduce to $1 - T^{\min}$ and $1 - T^{\text{prod}}$, inverting activations and reversing gradient signs during backpropagation. **Fig. 3.5** illustrates cases where T^D outperforms max pooling; the complement corresponds primarily to fine-grained texture classes in Flowers102 and CALTECH-256, where the pairwise aggregation cannot resolve subtle inter-class differences.

3.5 Summary

This chapter demonstrates that k-restricted grouping functions provide effective pooling operators for convolutional neural networks. Evaluation across 10 datasets, 6 architectures, and 8 k-restricted functions establishes consistent performance improvements over traditional max and average pooling methods.

The restriction parameter $k = 2$ proves optimal for standard 2×2 pooling windows, yielding $\binom{4}{2} = 6$ pairwise aggregations that balance feature preservation with computational efficiency. This choice follows directly from the theoretical framework established in Chapter 2.

Among the evaluated functions, $T_{\min}$ achieves superior or comparable accuracy on 8 of 10 datasets across all architectures. The sensitivity parameter $p = 2$ provides reliable performance with minimal tuning. Functions $T_{\text{sin-min}}$ and $T_{\text{sin-prod}}$ consistently underperform and should be avoided in practice.

The distributed gradient flow properties of k-restricted functions address limitations of max pooling's sparse gradients and average pooling's uniform weighting. Multiple elements within each pooling window receive gradient updates during backpropagation, improving training dynamics in deep networks.

Computational overhead remains modest, with GPU implementations achieving $45\times$ speedup over CPU execution. Wall-clock training time increases by less than 3% compared to traditional pooling, as convolution operations dominate total computation.

For practitioners, $T_{\min}$ with $k = 2$ and $p = 2$ serves as a robust default choice requiring minimal hyperparameter tuning. Dataset-specific optimization through cross-validation over $\{T_{\min}, T_{\text{prod}}, T_D, T_{\text{min-max}}\}$ with $p \in \{1, 2, 3\}$ typically identifies optimal configurations within a constrained search space.

4 Background and Preliminaries: Differentiable Rendering

This chapter presents the mathematical foundations for differentiable rendering, covering light transport simulation, surface representations, and automatic differentiation. These concepts underpin the neural directional distance field work in Chapter 5.

4.1 Rendering Fundamentals

Rendering transforms 3D scene descriptions into 2D images by simulating light transport [50]. Given a scene $\mathcal{S}$ containing surfaces S and light sources L, rendering computes pixel intensities $I(x, y)$ for image coordinates (x, y).

The core operation involves casting rays $r(t) = o + td$ from camera origin o in direction d and determining intersections with scene geometry (**Fig. 4.1**). Each ray-surface intersection contributes to the final pixel color through light transport calculations.

4.1.1 Path Tracing Algorithm

Radiance L quantifies light traveling along a ray, defined as radiant flux per unit projected area per unit solid angle with units $\mathrm{W} \cdot \mathrm{m}^{-2} \cdot \mathrm{sr}^{-1}$. Radiance remains constant along rays in vacuum, making it the natural quantity for ray-based rendering algorithms.

The rendering equation [50] describes equilibrium radiance distribution in a scene (**Fig. 4.2**):

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot n) d\omega_i \tag{4.1}$$

The left side, $L_o(x, \omega_o)$, represents outgoing radiance measured as radiant flux per unit area per unit solid angle leaving surface point x in direction ω_o. This quantity has units $\mathrm{W} \cdot \mathrm{m}^{-2} \cdot \mathrm{sr}^{-1}$ and determines the brightness observed along viewing rays.

On the right side, the first term $L_e(x, \omega_o)$ accounts for emitted radiance from light sources. For non-emissive surfaces, this term equals zero, while light sources contribute positive emission values that drive the light transport process.

The integral term captures reflected light contributions. The integrand contains three components that work together to model surface reflection. The BRDF $f_r(x, \omega_i, \omega_o)$ (detailed in Section 4.1.3) characterizes how the surface reflects incident light from direction ω_i toward direction ω_o [77]. This function encodes material properties and

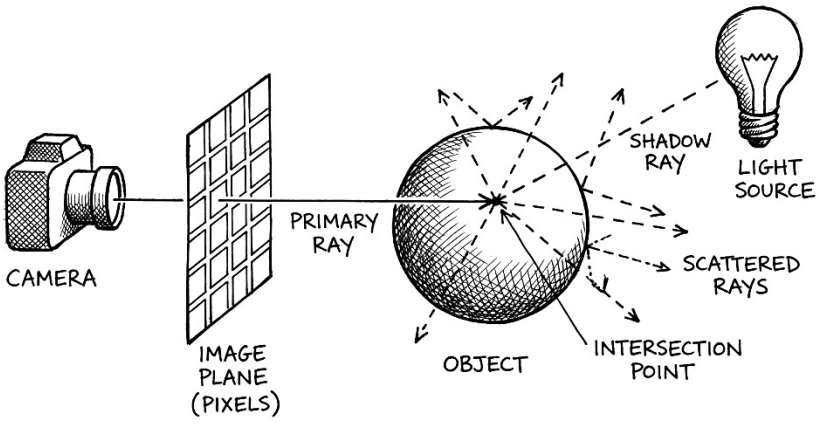

Fig. 4.1: Path tracing overview. Primary rays are cast from the camera through each pixel. At intersection points, shadow rays test light visibility while scattered rays sample indirect illumination according to the surface BRDF.

must satisfy physical constraints: non-negativity $f_r(x, \omega_i, \omega_o) \geq 0$, Helmholtz reciprocity $f_r(x, \omega_i, \omega_o) = f_r(x, \omega_o, \omega_i)$, and energy conservation $\int_\Omega f_r(x, \omega_i, \omega_o)(\omega_o \cdot n) d\omega_o \leq 1$.

The incoming radiance $L_i(x, \omega_i)$ represents light arriving at surface point x from direction ω_i. This creates the equation's recursive nature since incoming radiance depends on outgoing radiance from other surfaces: $L_i(x, \omega_i) = L_o(\text{trace}(x, \omega_i), -\omega_i)$, where $\text{trace}(x, \omega_i)$ finds the nearest intersection along the ray from x in direction ω_i.

The cosine term $(\omega_i \cdot n)$ implements Lambert's law, ensuring that incident flux decreases as the angle between light direction and surface normal increases. The integration occurs over the hemisphere Ω of directions above the surface, parameterized by spherical coordinates (θ, ϕ) where $\theta \in [0, \pi/2]$ and $\phi \in [0, 2\pi]$.

This recursive dependency makes analytical solutions intractable for complex scenes, motivating numerical approaches. Monte Carlo integration provides a practical solution by approximating the hemisphere integral through random sampling [102]:

$$\int_\Omega f_r(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot n) d\omega_i \approx \frac{1}{N} \sum_{j=1}^{N} \frac{f_r(x, \omega_j, \omega_o) L_i(x, \omega_j)(\omega_j \cdot n)}{p(\omega_j)} \tag{4.2}$$

The approximation quality depends on the number of samples N and the probability density $p(\omega_j)$. Importance sampling strategies choose $p(\omega_j) \propto f_r(x, \omega_j, \omega_o)(\omega_j \cdot n)$ to concentrate samples where the integrand has large magnitude, reducing estimation variance [82].

Path tracing implements this Monte Carlo approach recursively [50]. The algorithm begins by casting a primary ray $r_0(t) = o + td_0$ from the camera through each pixel into the scene. Upon finding the first intersection $x_0 = \text{intersect}(r_0, S)$, the algorithm samples a random direction ω_1 according to the surface's BRDF distribution. This spawns a secondary ray that continues the path deeper into the scene.

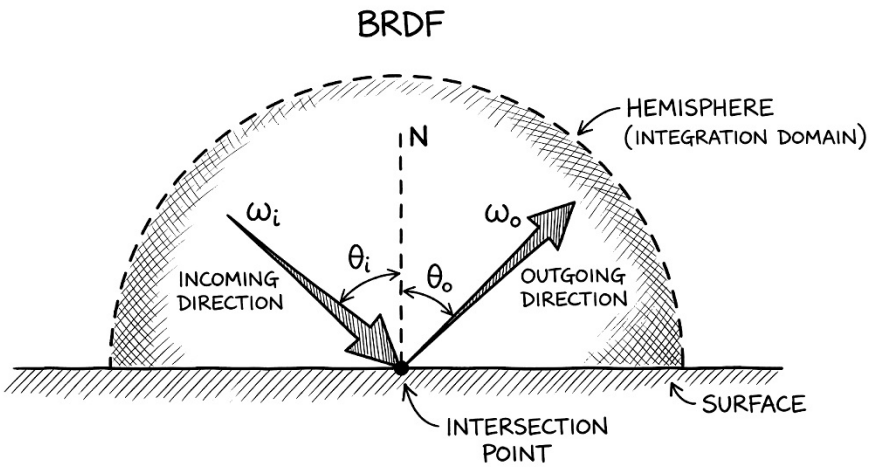

Fig. 4.2: Bidirectional reflectance distribution function (BRDF) geometry. The BRDF $f_r(\omega_i, \omega_o)$ relates incident radiance from direction ω_i to reflected radiance in direction ω_o at the intersection point. The hemisphere Ω centered at surface normal n defines the integration domain.

The process continues recursively, with each bounce generating new rays until paths terminate through absorption or escape from the scene. At each vertex x_i along a path, the algorithm accumulates light contributions from emissive surfaces and applies the appropriate BRDF weighting. The path throughput β_i tracks the cumulative attenuation:

$$\beta_i = \prod_{k=0}^{i-1} \frac{f_r(x_k, \omega_k, \omega_{k+1})(\omega_{k+1} \cdot n_k)}{p(\omega_{k+1}|x_k, n_k, \omega_k)} \tag{4.3}$$

where $p(\omega_{k+1})$ is the probability density function (PDF) of sampling the direction ω_{k+1}.

The final pixel estimate combines contributions from all sampled paths:

$$I = \frac{1}{N} \sum_{j=1}^{N} \sum_{i=0}^{k_j} \beta_i^{(j)} L_e(x_i^{(j)}) \tag{4.4}$$

where k_j denotes the length of path j and $x_i^{(j)}$ represents the i-th vertex along that path. This formulation naturally handles complex lighting effects including shadows, reflections, and global illumination through the recursive light transport simulation.

4.1.2 Ray-Surface Intersection

Ray-surface intersection forms the computational bottleneck in path tracing, requiring efficient algorithms to determine where rays intersect scene geometry. Given ray $r(t) = o + td$ with origin o and direction d, intersection algorithms solve for parameter t where the ray meets surface S.

The choice of scene representation fundamentally determines intersection complexity

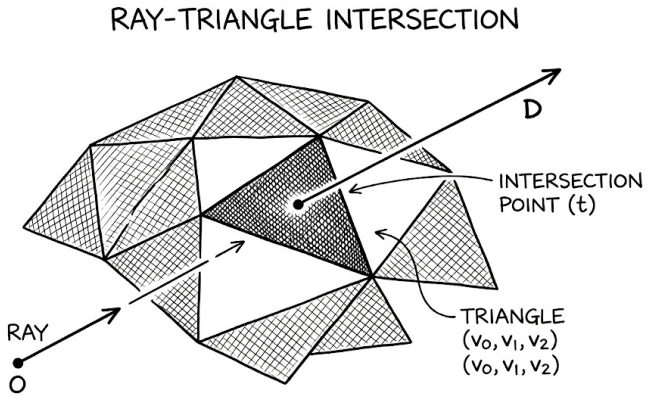

Fig. 4.3: Ray-triangle intersection. Given ray $r(t) = o + td$ and triangle with vertices (v_0, v_1, v_2), the intersection parameter t and barycentric coordinates are computed to determine the hit point.

and algorithmic approaches (**Fig. 4.4**). Different representations encode geometry in distinct ways, each with computational trade-offs that directly impact rendering performance.

Explicit representations Explicit representations store geometry as discrete primitives with known analytic forms. Triangle meshes represent the most common explicit format, where surfaces are approximated by planar triangles $T = \{v_0, v_1, v_2\}$ with vertices $v_i \in \mathbb{R}^3$. Ray-triangle intersection (**Fig. 4.3**) uses barycentric coordinates to test containment [73]:

$$P = (1 - u - v)v_0 + uv_1 + vv_2 \tag{4.5}$$

$$o + td = P \tag{4.6}$$

$$u, v \geq 0, \quad u + v \leq 1 \tag{4.7}$$

Solving this system yields intersection parameter t and barycentric weights (u, v) for valid intersections. The algorithm requires solving a 3×3 linear system, making it computationally efficient but limited to piecewise-linear approximations of smooth surfaces.

Point clouds represent surfaces as discrete sets of 3D points $\{p_i\}_{i=1}^{N}$ with associated attributes like color and normals. Ray intersection requires constructing implicit surfaces through techniques like radial basis functions or nearest-neighbor interpolation [2]. The scattered nature of point data necessitates spatial data structures for efficient querying during intersection tests.

Voxel grids discretize space into regular 3D cells, each storing occupancy or material properties. Ray-voxel intersection uses 3D line rasterization algorithms like 3D-DDA [3] to traverse occupied cells. While memory-intensive for high resolutions, voxel

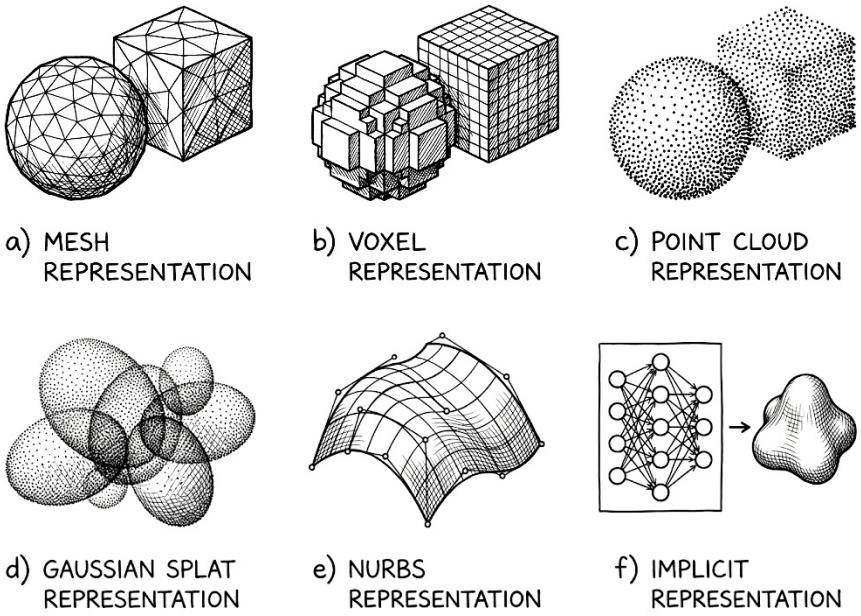

Fig. 4.4: 3D geometry representations. Top: explicit representations including (a) triangle mesh storing vertices and face connectivity, (b) voxel grid discretizing space into occupied/empty cells, (c) point cloud storing unconnected surface samples. Bottom: (d) Gaussian splats as overlapping anisotropic Gaussians, (e) NURBS surfaces via control points and basis functions, (f) neural implicit representation encoding geometry as the zero level-set of a learned function $f : \mathbb{R}^3 \to \mathbb{R}$.

representations enable efficient boolean operations and level-of-detail rendering through hierarchical structures like sparse voxel octrees [56].

Gaussian splatting represents scenes as collections of 3D Gaussians $G_i(x) = \exp(-\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i))$ with centers μ_i and covariance matrices Σ_i [52]. Rather than traditional ray intersection, rendering projects Gaussians to 2D screen space and composites contributions using alpha blending. This approach avoids explicit surface intersection but requires careful depth sorting and splat size computation.

Parametric surfaces like NURBS provide smooth representations but require iterative root-finding for intersection. Given parametric surface $S(u,v) = \sum_{i,j} N_{i,p}(u)N_{j,q}(v)P_{i,j}$ with basis functions N and control points $P_{i,j}$, intersection becomes a non-linear optimization problem with no closed-form solution [79].

Implicit representations Implicit representations define surfaces as level sets of scalar functions $f : \mathbb{R}^3 \to \mathbb{R}$ where $S = \{x : f(x) = 0\}$. Traditional implicit surfaces like quadrics admit analytic intersection formulas. For sphere $f(x) = \|x-c\|^2 - r^2$ with center c and radius r, substituting ray equation yields quadratic $at^2 + bt + c = 0$ with coefficients:

$$a = \|d\|^2 \tag{4.8}$$

$$b = 2d \cdot (o - c) \tag{4.9}$$

$$c = \|o - c\|^2 - r^2 \tag{4.10}$$

Signed distance functions (SDFs) encode surfaces as $f(x) = \text{distance}(x, S)$ with negative values inside objects. SDFs enable sphere tracing [44] (**Fig. 4.5**), which iteratively steps along rays using distance bounds to guarantee convergence:

$$t_0 = 0 \tag{4.11}$$

$$t_{i+1} = t_i + |f(o + t_i d)| \tag{4.12}$$

$$\text{converge when } |f(o + t_i d)| < \epsilon \tag{4.13}$$

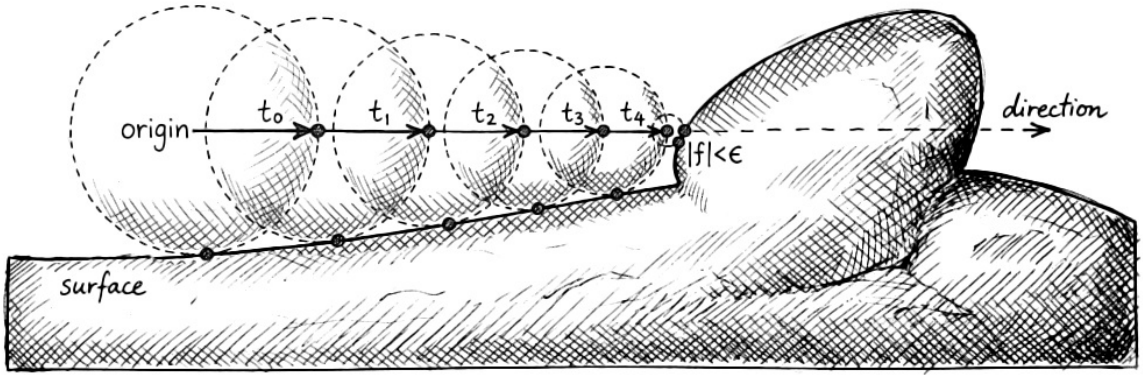

Fig. 4.5: Sphere tracing for SDF ray intersection. Starting from origin o, each step advances by the SDF value $|f(r(t_i))|$, which guarantees no surface intersection within that radius. The algorithm converges when $|f| < \epsilon$.

This approach handles complex shapes defined by SDF operations like union, intersection, and smooth blending without explicit meshing.

Occupancy functions represent surfaces through binary classification $f : \mathbb{R}^3 \to [0, 1]$ indicating interior probability. Ray intersection requires root-finding to locate zero-crossings where occupancy transitions from interior to exterior. Marching algorithms sample along rays to detect sign changes, then refine intersection positions through bisection or higher-order methods [66].

Neural implicit representations encode surfaces as neural networks $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ where surface positions satisfy $f_\theta(x) = 0$ [70, 81]. These representations offer arbitrary topological complexity and continuous surface definitions but require specialized intersection

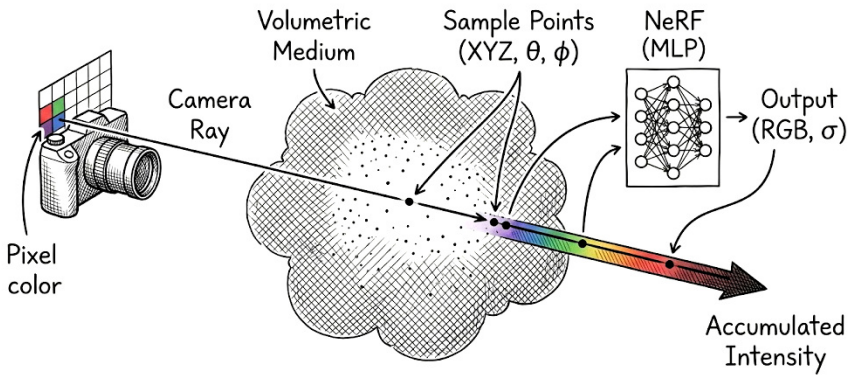

Fig. 4.6: Neural radiance field (NeRF) rendering. Points (x, y, z) along the camera ray are queried with viewing direction (θ, ϕ). The MLP outputs color c and density σ. Pixel color is computed via volume rendering: $C = \int T(t)\sigma(t)c(t)\,dt$ where $T(t) = \exp(-\int_0^t \sigma(s)\,ds)$.

algorithms. Root-finding becomes more expensive due to network evaluation costs, typically requiring 50-200 forward passes per intersection.

Neural radiance fields (NeRFs) parameterize both density and color as neural functions $(\sigma, c) = f_\theta(x, d)$ [72] (**Fig. 4.6**). Rather than explicit surface intersection, NeRF rendering integrates density along rays:

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))c(r(t), d)dt \tag{4.14}$$

where $T(t) = \exp(-\int_{t_n}^{t} \sigma(r(s))ds)$ represents transmittance. Numerical quadrature approximates this integral using stratified sampling along ray segments.

Despite these neural advances, explicit representations retain advantages in specific scenarios. Triangle meshes support hardware-accelerated ray tracing through BVH traversal, achieving real-time performance unattainable with neural network queries. Explicit geometry enables direct editing, UV mapping, and integration with existing production pipelines. For scenes with sharp edges or thin structures, meshes provide exact representations without the sampling artifacts that affect volumetric methods. Point clouds excel at representing raw scan data without topological assumptions, while voxels enable efficient boolean operations for constructive solid geometry.

4.1.3 Bidirectional Reflectance Distribution Function

The bidirectional reflectance distribution function (BRDF) characterizes how surfaces reflect incident light, forming the core material model in physically-based rendering. Defined as the ratio of reflected radiance to incident irradiance, the BRDF encodes surface appearance through local scattering behavior.

Formally, the BRDF relates infinitesimal incident flux $d\Phi_i$ from direction ω_i to re-

flected radiance dL_r in direction ω_r:

$$f_r(x, \omega_i, \omega_r) = \frac{dL_r(x, \omega_r)}{dE_i(x, \omega_i)} = \frac{dL_r(x, \omega_r)}{L_i(x, \omega_i) \cos\theta_i d\omega_i} \tag{4.15}$$

where θ_i denotes the angle between incident direction ω_i and surface normal n. This definition ensures dimensional consistency with units sr^{-1} (inverse steradians).

Physical constraints Physical BRDFs must satisfy fundamental constraints derived from energy conservation and reciprocity principles. Non-negativity requires $f_r(x, \omega_i, \omega_r) \geq 0$ for all direction pairs, preventing unphysical negative reflection.

Energy conservation constrains total reflected power to not exceed incident power:

$$\int_\Omega f_r(x, \omega_i, \omega_r) \cos\theta_r d\omega_r \leq 1 \tag{4.16}$$

for any incident direction ω_i. This hemispherical integral represents the directional-hemispherical reflectance, which must remain below unity to prevent energy amplification.

Helmholtz reciprocity states that BRDF values remain invariant under direction exchange:

$$f_r(x, \omega_i, \omega_r) = f_r(x, \omega_r, \omega_i) \tag{4.17}$$

This symmetry follows from microscopic reversibility in electromagnetic scattering and enables important algorithmic optimizations in bidirectional rendering methods.

Analytical BRDF models Analytical BRDF models provide closed-form expressions for common material classes. The Lambertian model represents perfect diffuse reflection with constant BRDF:

$$f_r^{\text{Lambert}}(x, \omega_i, \omega_r) = \frac{\rho}{\pi} \tag{4.18}$$

where $\rho \in [0, 1]$ denotes surface albedo. The π normalization ensures energy conservation for white surfaces with $\rho = 1$.

The Phong model captures specular reflection through cosine lobes:

$$f_r^{\text{Phong}}(x, \omega_i, \omega_r) = k_d \frac{\rho}{\pi} + k_s \frac{n+2}{2\pi} (\omega_r \cdot R)^n \tag{4.19}$$

where $R = 2(\omega_i \cdot n)n - \omega_i$ represents perfect reflection direction, n controls specular sharpness, and k_d, k_s balance diffuse and specular components. Phong model is simple but it is not strictly energy-conserving and hence, models like Cook-Torrance[30] has become a standard.

Neural BRDF representations Neural networks offer flexible BRDF parameterization for complex materials beyond analytical models. Two primary approaches dominate: multilayer perceptrons and spherical harmonic bases, each with distinct advantages for different applications.

MLP representation Multilayer perceptrons directly map direction pairs to reflectance values through learned nonlinear transformations. The network parameterization takes the form:

$$f_r(\omega_i, \omega_r) = \text{MLP}_\theta(\psi(\omega_i, \omega_r)) \tag{4.20}$$

where $\psi(\cdot)$ represents a coordinate transformation and θ denotes learnable parameters. Common input encodings include Cartesian coordinates $(\omega_{i,x}, \omega_{i,y}, \omega_{i,z}, \omega_{r,x}, \omega_{r,y}, \omega_{r,z})$ or spherical angles $(\theta_i, \phi_i, \theta_r, \phi_r)$.

Ensuring physical constraints requires careful network design. Non-negativity is enforced through ReLU or softplus activations in the output layer. Energy conservation presents greater challenges, typically addressed through normalization schemes or constraint-aware loss functions during training [87]. Positional encoding enhances high-frequency detail representation by mapping input directions through sinusoidal functions:

$$\gamma(\omega) = \begin{bmatrix} \sin(2^0\pi\omega) & \cos(2^0\pi\omega) & \dots & \sin(2^{L-1}\pi\omega) & \cos(2^{L-1}\pi\omega) \end{bmatrix}^T \tag{4.21}$$

enabling the networks to capture sharp reflection lobes and complex material variations that standard coordinate inputs struggle to represent.

Spherical harmonics provide an orthonormal basis for functions defined on the unit sphere, offering compact BRDF representation through frequency domain decomposition. The spherical harmonic expansion decomposes the BRDF as:

$$f_r(\omega_i, \omega_r) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} c_{lm}(\omega_i) Y_l^m(\omega_r) \tag{4.22}$$

where $Y_l^m(\theta, \phi)$ denotes spherical harmonic basis functions of degree l and order m, and $c_{lm}(\omega_i)$ represents incident-direction-dependent coefficients.

The basis functions are defined as:

$$Y_l^m(\theta, \phi) = \sqrt{\frac{2l+1}{4\pi} \frac{(l-|m|)!}{(l+|m|)!}} P_l^{|m|}(\cos\theta) e^{im\phi} \tag{4.23}$$

where P_l^m denotes associated Legendre polynomials. Low-frequency terms capture broad diffuse reflection, while higher frequencies encode sharp specular features (**Fig. 4.7**).

For practical implementation, the expansion is truncated at maximum degree L, yielding $(L+1)^2$ basis functions. The coefficients $c_{lm}(\omega_i)$ can be parameterized through neural

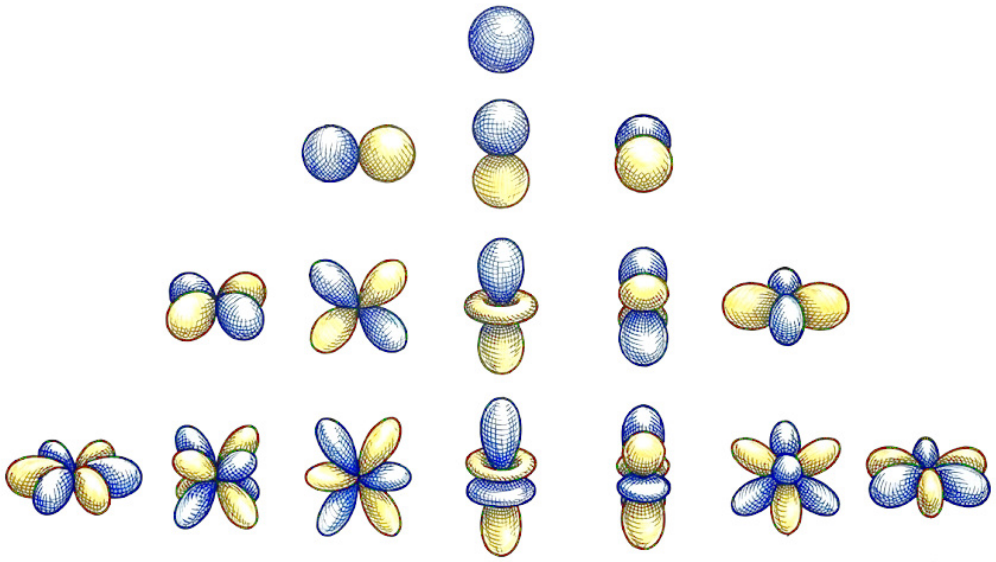

Fig. 4.7: Spherical harmonic basis functions Y_l^m for degrees $l = 0, 1, 2, 3$ (top to bottom). Each row contains $2l+1$ functions with orders $m \in \{-l, \ldots, l\}$. Lobe colors indicate sign: positive (blue) and negative (yellow) regions of each basis function.

networks or learned directly:

$$c_{lm}(\omega_i) = \text{MLP}_{\theta_{lm}}(\psi(\omega_i)) \tag{4.24}$$

This hybrid approach combines spherical harmonic compactness with neural network expressiveness, enabling efficient BRDF representation with guaranteed smoothness properties.

Spherical harmonic BRDFs naturally satisfy reciprocity when coefficient functions respect the symmetry $c_{lm}(\omega_i) = c_{lm}(\omega_r)$. Energy conservation requires additional constraints on coefficient magnitudes, typically enforced through projection operations during optimization.

4.2 Automatic Differentiation

Automatic differentiation (AD) computes exact derivatives of functions defined by computer programs through systematic application of the chain rule. Unlike numerical differentiation, which approximates derivatives through finite differences, AD produces machine-precision gradients by decomposing computations into elementary operations with known Jacobian matrices.

Consider a function $f : \mathbb{R}^n \to \mathbb{R}^m$ defined by a computational graph with intermediate variables $v_1, \ldots, v_k$. Each elementary operation corresponds to a function $\phi_i : \mathbb{R}^{n_i} \to \mathbb{R}^{m_i}$ with Jacobian matrix $J_{\phi_i} \in \mathbb{R}^{m_i \times n_i}$. The chain rule expresses the total Jacobian as a

product of elementary Jacobians:

$$J_f = J_{\phi_k} J_{\phi_{k-1}} \cdots J_{\phi_1} \tag{4.25}$$

Direct computation of this matrix product is computationally expensive and often unnecessary. Instead, AD computes specific Jacobian-vector products efficiently by exploiting matrix-vector multiplication associativity.

The fundamental principle underlying AD is that Jacobian-vector products can be computed without explicitly forming the full Jacobian matrix. For composition $f = \phi_2 \circ \phi_1$, the Jacobian-vector product $J_{\phi_2} J_{\phi_1} v$ can be computed as either $J_{\phi_2}(J_{\phi_1} v)$ or $(J_{\phi_2} J_{\phi_1})v$, with different computational costs depending on matrix dimensions.

Two evaluation strategies exist for computing these products: forward mode and reverse mode. Forward mode propagates derivatives from inputs to outputs, computing Jacobian-vector products $J_f v$ efficiently when the number of inputs is small. Reverse mode propagates derivatives from outputs to inputs, computing vector-Jacobian products $u^T J_f$ efficiently when the number of outputs is small. Machine learning predominantly uses reverse mode (backpropagation) since loss functions produce scalar outputs optimized over many parameters.

4.2.1 Forward Mode as Jacobian-Vector Products

Forward mode automatic differentiation computes Jacobian-vector products (JVPs) by propagating tangent vectors from inputs to outputs. This operation is also known as the pushforward in differential geometry.

Given function $f : \mathbb{R}^n \to \mathbb{R}^m$ and tangent vector $v \in \mathbb{R}^n$, forward mode computes $J_f v$ where $J_f \in \mathbb{R}^{m \times n}$ denotes the Jacobian matrix. The key insight is that this product can be computed without explicitly forming J_f.

For elementary operation $\phi : \mathbb{R}^{n_i} \to \mathbb{R}^{m_i}$ with input tangent $\dot{u} \in \mathbb{R}^{n_i}$, the output tangent becomes:

$$\dot{v} = J_\phi \dot{u} \tag{4.26}$$

where J_ϕ is the Jacobian of ϕ evaluated at the current primal values.

Consider the composition $f = \phi_2 \circ \phi_1$ with Jacobians $J_1 \in \mathbb{R}^{k \times n}$ and $J_2 \in \mathbb{R}^{m \times k}$. Forward mode computes:

$$\dot{v}_1 = J_1 v \tag{4.27}$$

$$\dot{v}_2 = J_2 \dot{v}_1 = J_2(J_1 v) = (J_2 J_1)v \tag{4.28}$$

This associativity ensures that forward mode computes the correct Jacobian-vector product without forming the full matrix product $J_2 J_1$.

For scalar-valued elementary operations, the Jacobian reduces to a gradient vector.

Consider $\phi(u_1, u_2) = u_1 u_2$ with inputs (u_1, u_2) and tangents $(\dot{u}_1, \dot{u}_2)$:

$$\dot{v} = \nabla\phi \cdot \begin{pmatrix} \dot{u}_1 \\ \dot{u}_2 \end{pmatrix} = u_2\dot{u}_1 + u_1\dot{u}_2 \tag{4.29}$$

For vector-valued operations like matrix multiplication $C = AB$, the Jacobian becomes a tensor, but the tangent propagation remains straightforward:

$$\dot{C} = \dot{A}B + A\dot{B} \tag{4.30}$$

Forward mode requires one pass per column of the desired Jacobian. To compute the full Jacobian $J_f \in \mathbb{R}^{m\times n}$, forward mode evaluates n Jacobian-vector products using canonical basis vectors e_i:

$$J_f = [J_f e_1 \mid J_f e_2 \mid \cdots \mid J_f e_n] \tag{4.31}$$

The computational complexity is $O(n \cdot \text{cost}(f))$, making forward mode efficient when $n \ll m$.

4.2.2 Reverse Mode as Vector-Jacobian Products

Reverse mode automatic differentiation computes vector-Jacobian products (VJPs) by propagating cotangent vectors from outputs to inputs. This operation corresponds to the pullback in differential geometry.

Given function $f : \mathbb{R}^n \to \mathbb{R}^m$ and cotangent vector $u \in \mathbb{R}^m$, reverse mode computes $u^T J_f$ where $J_f \in \mathbb{R}^{m\times n}$. The cotangent vector typically represents the gradient of a scalar loss function with respect to the function outputs.

Reverse mode proceeds in two phases. The forward pass evaluates the function while recording the computational graph. The backward pass propagates cotangents by applying transposed Jacobians in reverse order:

$$u^T J_f = u^T J_{\phi_k} J_{\phi_{k-1}} \cdots J_{\phi_1} = (\cdots((u^T J_{\phi_k}) J_{\phi_{k-1}}) \cdots) J_{\phi_1} \tag{4.32}$$

For elementary operation ϕ with output cotangent $\bar{v} \in \mathbb{R}^{m_i}$, the input cotangent becomes:

$$\bar{u} = J_\phi^T \bar{v} \tag{4.33}$$

When multiple operations share the same input variable, cotangents accumulate additively due to the multivariate chain rule. Reverse mode computes one row of the Jacobian per backward pass. To compute the full Jacobian $J_f \in \mathbb{R}^{m\times n}$, reverse mode evaluates m vector-Jacobian products using canonical basis vectors:

$$J_f = \begin{bmatrix} e_1^T J_f & e_2^T J_f & \dots & e_m^T J_f \end{bmatrix}^T \tag{4.34}$$

The computational complexity is $O(m \cdot \text{cost}(f))$, making reverse mode efficient when $m \ll n$. This explains the dominance of reverse mode in machine learning, where scalar loss functions ($m = 1$) are optimized with respect to many parameters.

Modern automatic differentiation frameworks implement reverse mode through computational graph construction and traversal. The graph nodes store primal values and local Jacobian information, while edges represent data dependencies. Gradient computation proceeds by traversing the graph in reverse topological order, accumulating cotangents at each node.

Memory management requires careful attention since all intermediate values must be retained for the backward pass. Advanced techniques like gradient checkpointing trade computation for memory by recomputing selected intermediate values rather than storing them throughout the forward pass [23].

4.3 Differentiable Rendering

Differentiable rendering extends traditional rendering algorithms to compute gradients with respect to scene parameters, enabling inverse problems like shape reconstruction, material estimation, and lighting optimization. While automatic differentiation provides the mathematical foundation, applying it directly to rendering pipelines reveals fundamental challenges that require specialized solutions.

The field advanced significantly with edge sampling methods [62] that correctly handle visibility discontinuities. Reparameterization approaches [68] and warped-area sampling [4] provided alternative solutions with different variance-bias trade-offs. Path-space formulations extended these ideas to global illumination [106]. Neural implicit representations like NeRF [72] and neural SDFs [81] sidestep some discontinuity issues through volumetric or implicit formulations. Physics-based differentiable rendering has since been applied to material acquisition, relighting, and inverse graphics problems.

4.3.1 Gradient Computation

The core challenge in differentiable rendering lies in computing derivatives of the rendering integral. For a single pixel, the value I integrates a measurement contribution function f over domain Ω:

$$I(\theta) = \int_\Omega f(\omega; \theta)\, d\omega \tag{4.35}$$

where θ represents scene parameters such as object positions or material properties.

Naive differentiation applies the Leibniz integral rule by swapping integral and derivative operators:

$$\frac{dI}{d\theta} \stackrel{?}{=} \int_\Omega \frac{\partial f(\omega; \theta)}{\partial \theta}\, d\omega \tag{4.36}$$

This operation requires the integrand f to be continuous with respect to θ and the

integration domain Ω to remain fixed. However, path tracing violates both conditions through visibility discontinuities.

The measurement function f contains visibility terms $V(\cdot)$ that behave as step functions. Infinitesimal changes in geometric parameters can cause surfaces to suddenly appear or disappear from view, creating discontinuities. The derivative of this step function becomes a Dirac delta function, non-zero only at silhouette boundaries.

Standard Monte Carlo sampling has zero probability of hitting these lower-dimensional boundaries, making naive differentiation severely biased. The correct approach follows the Reynolds transport theorem, which separates the derivative into interior and boundary contributions:

$$\frac{dI}{d\theta} = \underbrace{\int_{\Omega(\theta)} \frac{\partial f}{\partial \theta}\, d\omega}_{\text{Interior Term}} + \underbrace{\int_{\partial\Omega(\theta)} f(\omega;\theta)(v_b \cdot n)\, ds}_{\text{Boundary Term}} \tag{4.37}$$

where v_b denotes boundary velocity and n represents the outward normal. The main challenge becomes robust estimation of the boundary term.

Edge sampling methods Direct boundary evaluation identifies silhouette edges explicitly and samples them during rendering [62]. The algorithm augments standard path tracing with:

(a) **Silhouette identification**: Find all silhouette edges visible from each shading point using acceleration structures like 6D Hough transforms.

(b) **Boundary sampling**: Sample points directly on identified silhouette edges.

(c) **Discontinuity estimation**: Trace rays on both sides of each edge to compute radiance differences ΔL.

This approach produces unbiased gradient estimates but suffers from high variance. Finding visible silhouettes proves computationally expensive, especially for complex geometry or higher-order light transport. Many sampled edge points become occluded, creating firefly artifacts in gradient images.

Path-space formulations generalize these concepts to multi-bounce light transport [106]. The differential path integral splits into interior path integrals over smooth BRDF components and boundary path integrals over paths with vertices on silhouette edges.

Reparameterization methods Alternative approaches reformulate the integral to avoid moving boundaries entirely. Reparameterization transforms the integration domain so discontinuities become stationary with respect to new variables.

Loubet et al. [68] introduce transformation T mapping new variables ω' to original domain ω:

$$\omega = T(\omega';\theta) \tag{4.38}$$

The transformed integral becomes:

$$I(\theta) = \int_\Omega f(T(\omega';\theta);\theta)|\det J_T|\, d\omega' \tag{4.39}$$

For ideal transformations, discontinuities remain fixed in ω'-space, enabling standard differentiation. The method approximates local silhouette motion through spherical rotations $R(\theta)$:

$$\omega = R(\theta)\omega' \tag{4.40}$$

Auxiliary rays estimate local geometric motion to construct appropriate rotations. While biased due to first-order approximations, this approach provides low-variance gradients that prove more useful for optimization than high-variance unbiased estimates.

Warped-area sampling Bangaru et al. [4] achieve unbiased estimation without explicit edge detection using the divergence theorem. The boundary integral transforms to an area integral involving a warp field $V_\theta(\omega)$:

$$I_B = \int_{\partial\Omega} f(v_b \cdot n)\, ds = \int_\Omega \nabla \cdot (fV_\theta)\, d\omega \tag{4.41}$$

Expanding the divergence yields:

$$I_B = \int_\Omega [(\nabla f \cdot V_\theta) + f(\nabla \cdot V_\theta)]\, d\omega \tag{4.42}$$

The warp field must satisfy continuity and boundary consistency conditions. Construction proceeds through boundary-aware convolution of discontinuous direct fields with harmonic kernels:

$$V_\theta(\omega) = \frac{\int_\Omega k(\omega,\omega')V_\theta^{\text{direct}}(\omega')\, d\omega'}{\int_\Omega k(\omega,\omega')\, d\omega'} \tag{4.43}$$

The kernel $k(\omega,\omega') = 1/(D(\omega,\omega')+B(\omega'))$ incorporates distance metrics D and boundary tests B that vanish at silhouettes. This ensures boundary consistency while maintaining continuity.

Russian roulette techniques achieve unbiased estimation despite the convolution normalization. Antithetic sampling and control variates reduce variance from the nested Monte Carlo scheme.

4.3.2 Surface Parameterization

Traditional mesh representations create additional differentiability challenges through discrete topology and vertex connectivity. Neural implicit representations offer smooth, continuous alternatives that naturally integrate with automatic differentiation frameworks.

Signed distance functions parameterized through neural networks $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ define surfaces as zero level sets $S = \{x : f_\theta(x) = 0\}$. Surface normals compute as normalized gradients $n = \nabla f_\theta(x)/\|\nabla f_\theta(x)\|$, providing smooth derivatives throughout space.

The directed distance field representation from [8] extends this concept by encoding directional distance functions $\phi_\theta : \mathbb{R}^3 \times S^2 \to \mathbb{R}^+$. This parameterization enables efficient ray intersection while maintaining differentiability:

$$\text{intersect}(r(t), S) = o + \phi_\theta(o, d) \cdot d \tag{4.44}$$

Neural radiance fields represent both geometry and appearance through density and color functions $(\sigma, c) = f_\theta(x, d)$ [72]. The volumetric rendering equation naturally provides smooth gradients:

$$\frac{\partial C}{\partial \theta} = \int_{t_n}^{t_f} T(t) \frac{\partial}{\partial \theta} [\sigma(r(t)) c(r(t), d)] dt \tag{4.45}$$

where transmittance $T(t) = \exp(-\int_{t_n}^{t} \sigma(r(s)) ds)$ varies smoothly with network parameters.

These neural representations enable end-to-end optimization of both geometry and appearance through gradient descent, avoiding the discrete optimization challenges inherent in traditional mesh-based approaches. The continuous nature of neural fields eliminates visibility discontinuities at object boundaries while maintaining the expressiveness needed for complex scene reconstruction.

5 Neural Implicit Representations

This chapter presents neural directional distance fields (DDFs) for efficient path-traced rendering, based on the work in [8]. While Chapter 4 covered general implicit representations and differentiable rendering, this chapter focuses on the novel DDF representation that theoretically enables constant-time ray intersection queries. The key contributions include: the DDF formulation with its mathematical properties, neural network parameterization strategies, informative sampling techniques for training, and a modified path tracing algorithm that exploits directional distance information.

5.1 Implicit Surface Representations

Implicit surface representations define geometry through functions rather than explicit vertex locations. Unlike explicit representations such as meshes or point clouds, implicit functions $f : \mathbb{R}^3 \to \mathbb{R}$ characterize surfaces as level sets where $f(x) = 0$ for points x on the surface [8].

5.1.1 Signed Distance Functions

A signed distance function (SDF) is a field over 3D space $\mathbb{R}^3$ where every point is assigned the distance to the nearest surface. For a surface S, the SDF is defined as:

$$\text{SDF}(x) = \begin{cases} -d(x, S) & \text{if } x \text{ is inside } S \\ +d(x, S) & \text{if } x \text{ is outside } S \end{cases} \tag{5.1}$$

where $d(x, S) = \min_{y \in S} \|x - y\|_2$ is the Euclidean distance from point x to the surface S [8].

The zero level set $\{x \in \mathbb{R}^3 : \text{SDF}(x) = 0\}$ represents the surface boundary. SDFs provide several advantages: they naturally handle topological changes, enable Boolean operations, and support efficient sphere tracing algorithms [44, 8].

However, storing SDFs as discretized values requires $O(n^3)$ space, making them impractical for high-resolution representations. Recent advances use neural networks as universal function approximators to learn SDF representations from training data [81, 8].

5.1.2 Occupancy Functions

Occupancy functions represent geometry through binary classification. For each point $x \in \mathbb{R}^3$, the occupancy function $O : \mathbb{R}^3 \to \{0, 1\}$ indicates:

$$O(x) = \begin{cases} 1 & \text{if } x \text{ is inside the object} \\ 0 & \text{if } x \text{ is outside the object} \end{cases} \tag{5.2}$$

Unlike SDFs, occupancy functions only encode binary inside/outside information without distance metrics. This representation is particularly suited for learning-based reconstruction from partial observations, as it avoids the metric constraints imposed by distance fields [70, 8].

Both SDF and occupancy representations can be converted to explicit meshes using algorithms such as Marching Cubes [66], enabling integration with traditional graphics pipelines [8]. While these representations have proven effective, they lack directional information that can be exploited for more efficient rendering algorithms.

5.2 Directional Distance Fields

Directional distance fields (DDFs) extend traditional distance representations by incorporating directional information. For a surface S contained within bounded box $B \subset \mathbb{R}^3$, a DDF is defined as:

$$\phi : B \times \mathbb{R}^2 \to \mathbb{R}^+ \tag{5.3}$$

where $\phi(x, \theta)$ returns the minimum distance from point $x \in B$ to surface S in direction $\theta \in \mathbb{R}^2$ [8].

The viewing direction θ can be parameterized using spherical coordinates with azimuthal angle θ_0 and polar angle θ_1:

$$\theta = \begin{bmatrix} \cos(\theta_0)\sin(\theta_1) \\ \sin(\theta_0)\sin(\theta_1) \\ \cos(\theta_1) \end{bmatrix} \tag{5.4}$$

Moving distance t from point x in direction θ yields position $x' = x + t\theta$ [8].

A binary visibility function $\xi(x, \theta) = \mathbf{1}_S(x, \theta)$ indicates whether the ray from x in direction θ intersects surface S. An oriented point (x, θ) is visible if $\xi(x, \theta) = 1$.

5.2.1 Mathematical Properties

DDFs exhibit several geometric properties that enable efficient rendering and surface reconstruction. These properties distinguish DDFs from traditional distance representations and form the mathematical foundation for neural parameterization.

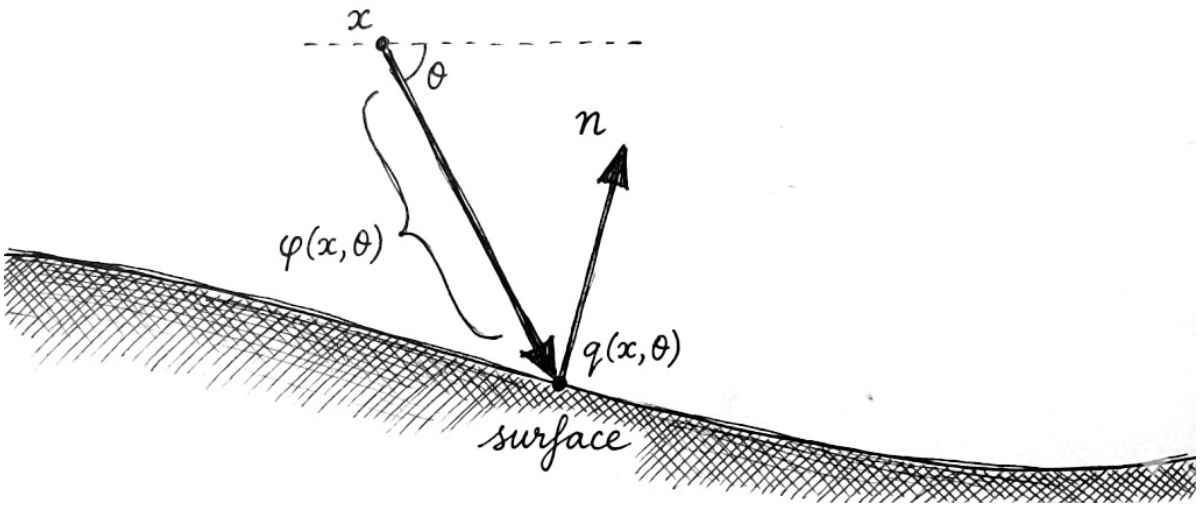

Fig. 5.1: Directional distance field representation. Point x with direction θ intersects surface S at $q(x,\theta) = x + \phi(x,\theta)\theta$, where $\phi(x,\theta)$ is the distance along direction θ to the surface. The normal n at the intersection point is shown for reference.

Directed Eikonal Equation For any visible oriented point (p,θ), the DDF satisfies the directed eikonal equation [8]:

$$\nabla_x \phi(x,\theta) \cdot \theta = -1 \tag{5.5}$$

The intuition follows from the definition: if $\phi(x,\theta)$ measures distance to the surface along direction θ, then moving one unit along θ reduces this distance by exactly one unit. Formally, the directional derivative $\nabla_x \phi \cdot \theta$ measures the rate of change of ϕ in direction θ, which equals -1 because each unit step toward the surface decreases the remaining distance by one unit. The negative sign indicates that the DDF value decreases as we move toward the surface [8].

From equation (5.5), we derive the ray marching property:

$$\phi(x,\theta) - \phi(x + t\theta, \theta) = t \tag{5.6}$$

This means that for every step of size t along the ray direction, the DDF value decreases by exactly t, providing a constant-time ray-surface intersection test in theory, though neural network approximation errors necessitate iterative ray marching with safety factor $\alpha < 1$ in practice [8].

The directed eikonal equation also constrains the gradient magnitude:

$$\|\nabla_x \phi(x,\theta)\|_2 \geq 1 \tag{5.7}$$

with equality only when the gradient aligns with the negative viewing direction $-\theta$.

Surface Normal Computation The point on surface S corresponding to oriented point (x, θ) is given by:

$$q(x, \theta) = x + \phi(x, \theta)\theta \in S \tag{5.8}$$

The surface normal at point q can be computed directly from the DDF gradient:

$$n(x, \theta) = \frac{\kappa \nabla_x \phi(x, \theta)^T}{\|\nabla_x \phi(x, \theta)\|_2} \tag{5.9}$$

where $\kappa \in \{-1, 1\}$ is chosen such that $n^T\theta < 0$, ensuring the normal points away from the viewing direction [8]. The choice of κ can be determined by:

$$\kappa = -\text{sign}(\nabla_x \phi(x, \theta)^T \theta) \tag{5.10}$$

This formulation ensures that the computed normal is always outward-facing relative to the ray direction, which is important for correct shading calculations in rendering applications.

Gradient Consistency The gradient consistency property relates spatial and directional derivatives of the DDF. For any visible oriented point (x, θ), an infinitesimal change in viewing direction $\delta\theta$ affects the DDF according to:

$$\nabla_\theta \phi(x, \theta) = \phi(x, \theta) \nabla_x \phi(x, \theta) \delta\theta \tag{5.11}$$

where $\delta\theta = \omega \times \theta$ represents a rotational perturbation with angular velocity ω [8]. This relationship can be expanded for small rotations. Consider a rotation by angle ϵ about axis ω:

$$\theta' = \theta + \epsilon(\omega \times \theta) + O(\epsilon^2) \tag{5.12}$$

The gradient consistency then gives:

$$\left.\frac{\partial \phi(x, \theta)}{\partial \epsilon}\right|_{\epsilon=0} = \phi(x, \theta) \nabla_x \phi(x, \theta)^T (\omega \times \theta) \tag{5.13}$$

This property ensures that the DDF varies smoothly with respect to both spatial position and viewing direction, which is important for stable optimization during neural network training.

Local Differentiability For any visible oriented point (x, θ), the local geometry of the 2D manifold S near point $q(x, \theta)$ is completely characterized by $\phi(x, \theta)$ and its derivatives [8]. The principal curvatures κ_1, κ_2 of the surface can be recovered from the Hessian of the DDF:

$$H = \nabla_x^2 \phi(x, \theta) \tag{5.14}$$

The mean curvature is given by:

$$H_{\text{mean}} = \frac{1}{2}\text{tr}(H - (H \cdot \theta)\theta^T) \tag{5.15}$$

where the term $(H \cdot \theta)\theta^T$ removes the component along the viewing direction.

Unsigned Distance Field Recovery The unsigned distance field can be recovered from the DDF by computing the minimum over all directions:

$$\text{UDF}(x) = \min_{\theta \in S^2} \phi(x, \theta) \tag{5.16}$$

where S^2 denotes the unit sphere. Unlike DDFs, UDFs have no discontinuities but lose all directional information [8]. The minimizing direction $\theta^*(x)$ that achieves this minimum:

$$\theta^*(x) = \arg\min_{\theta \in S^2} \phi(x, \theta) \tag{5.17}$$

points toward the closest surface point from x. This direction satisfies:

$$\theta^*(x) = \frac{q(x, \theta^*(x)) - x}{\|q(x, \theta^*(x)) - x\|_2} \tag{5.18}$$

Visibility and Discontinuities For invisible oriented points where $\xi(x, \theta) = 0$, the DDF value is undefined or set to infinity. This creates discontinuities in the field that must be handled during neural network training. A common approach uses a squishing function $\sigma : \mathbb{R} \to [0, 1]$ such as:

$$\sigma(y) = \tanh(y) \quad \text{or} \quad \sigma(y) = \text{erf}(y) \tag{5.19}$$

The training loss becomes:

$$L = \|\sigma(y_{\text{gt}}) - \sigma(\tilde{y})\|_2^2 \tag{5.20}$$

where $y_{\text{gt}} = \phi_{\text{gt}}(x, \theta)$ is the ground truth DDF value and $\tilde{y} = \text{MLP}_w(x, \theta)$ is the network prediction [8].

This formulation maps infinite distances to finite values while preserving the relative ordering of distances, enabling stable gradient-based optimization. These mathematical properties provide the foundation for neural network parameterization of DDFs.

5.3 Neural Network Parameterization

Neural networks serve as universal function approximators for representing directional distance fields. A multilayer perceptron (MLP) $\text{MLP}_w : (\mathbb{R}^3, \mathbb{R}^2) \to \mathbb{R}$ with parameters w takes an oriented point (x, θ) and returns the DDF field value $\phi(x, \theta)$ [8]. The choice of

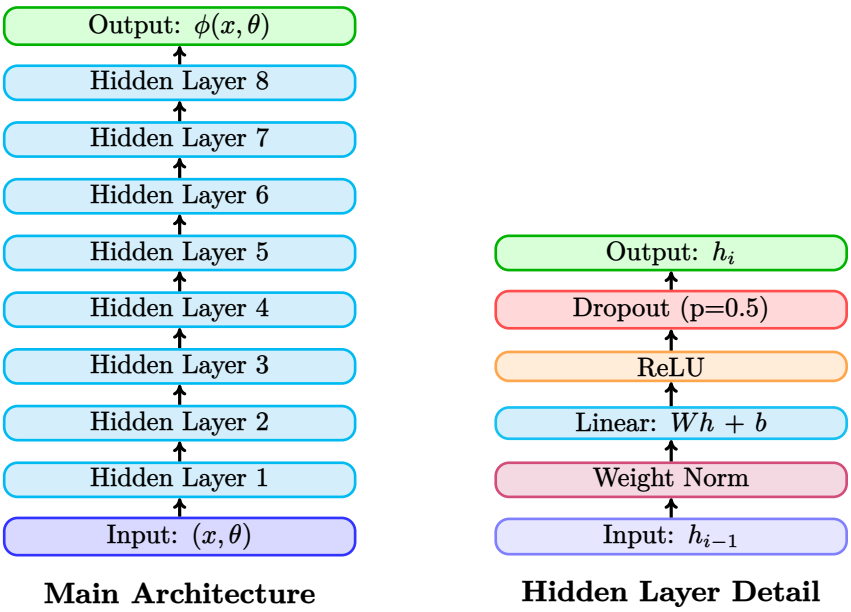

Fig. 5.2: Neural DDF network architecture. Left: The MLP takes a 5D input (3D position x and 2D direction θ) and outputs a scalar distance $\phi(x, \theta) \in \mathbb{R}^+$. Eight hidden layers of width 512 process the input sequentially. Right: Detail of operations within each hidden layer showing weight normalization, linear transformation, ReLU activation, and dropout (p=0.5). Total parameters: ∼1.3M.

network architecture, loss functions, and training strategies directly impacts the quality and efficiency of the learned representation (**Fig. 5.2**).

5.3.1 Network Architecture

The network architecture consists of fully connected layers with nonlinear activations. For DDF representation, the input is a 5-dimensional vector combining spatial position $x \in \mathbb{R}^3$ and viewing direction $\theta \in \mathbb{R}^2$:

$$\text{input} = [x_1, x_2, x_3, \theta_1, \theta_2]^T \tag{5.21}$$

The network uses 8 weight-normalized hidden layers of width 512, followed by ReLU activation and dropout with probability $p = 0.5$ [8]. The depth of 8 layers provides sufficient capacity to model complex distance variations across the 5D input space. Width 512 balances expressiveness against inference speed, wider networks improve accuracy but slow rendering. Dropout at 0.5 prevents overfitting to the training samples, important since DDF datasets are generated from discrete surface sampling rather than continuous supervision.

Weight Normalization Weight normalization [92] separates the magnitude and direction of weight vectors, improving training stability. For a weight vector w, the normalized weight is:

$$\hat{w} = \frac{g}{\|v\|_2} v \tag{5.22}$$

where g is a learnable scalar parameter and v is the direction vector. This reparameterization accelerates convergence and reduces sensitivity to initialization.

Activation Functions The ReLU activation function [76] $f(x) = \max(0, x)$ introduces nonlinearity while maintaining computational efficiency. For DDFs, ReLU ensures positive distance values in intermediate layers, though the final output may require additional constraints.

Dropout Regularization Dropout [99] randomly sets a fraction p of input units to zero during training, preventing overfitting. For each training example, units are retained with probability $(1 - p)$:

$$y = \frac{1}{1-p} \cdot \text{mask} \odot x \tag{5.23}$$

where mask is a binary vector with elements drawn from Bernoulli$(1-p)$ and $\odot$ denotes element-wise multiplication.

5.3.2 Loss Functions

The choice of loss function affects both training stability and representation quality. Standard L_2 loss fails for DDFs due to infinite values at invisible oriented points.

Clamped Loss The clamped error loss addresses infinite values by bounding the training range:

$$L_{\text{clamp}} = |\text{clamp}(y, \delta) - \text{clamp}(\tilde{y}, \delta)| \tag{5.24}$$

where $\text{clamp}(x, \delta) = \min(\delta, \max(-\delta, x))$ and δ is a hyperparameter controlling the distance from the surface over which optimization occurs [8].

Larger values of δ enable ray tracing from greater distances, while smaller values concentrate network capacity on surface details. This creates a trade-off between surface detail preservation and rendering distance.

Squashing Functions Squashing functions $\sigma : \mathbb{R} \to [0, 1]$ map infinite distances to bounded ranges. Common choices include:

$$\sigma_{\tanh}(x) = \tanh(x) \tag{5.25}$$

$$\sigma_{\text{erf}}(x) = \text{erf}(x) \tag{5.26}$$

$$\sigma_{\text{sigmoid}}(x) = \frac{1}{1 + e^{-x}} \tag{5.27}$$

The training loss using squashing functions becomes:

$$L_\sigma = \|\sigma(y_{\text{gt}}) - \sigma(\tilde{y})\|_2^2 \tag{5.28}$$

where y_{gt} is the ground truth DDF value and $\tilde{y}$ is the network prediction [8].

Eikonal Regularization To enforce the directed eikonal equation (5.5), an additional regularization term can be added:

$$L_{\text{eikonal}} = \lambda \|\nabla_x \phi(x, \theta) \cdot \theta + 1\|_2^2 \tag{5.29}$$

where λ controls the regularization strength. This term encourages the network to satisfy the eikonal constraint during training.

5.3.3 Training Strategies

Training neural DDFs requires careful sampling strategies since uniform random sampling performs poorly due to the sparsity of surface information in 3D space. Effective training strategies must balance computational efficiency with representation quality.

Optimization The Adam optimizer [53] with learning rate 10^{-7} provides stable convergence for DDF training [8]. Adam combines momentum and adaptive learning rates:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{5.30}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{5.31}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{5.32}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{5.33}$$

$$w_{t+1} = w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{5.34}$$

where g_t is the gradient, α is the learning rate, and β_1, β_2 are decay rates.

Surface-Based Sampling Uniform random sampling of points and directions trains networks poorly because many oriented points have infinite DDF values and contribute

little to shape estimation. Instead, surface-based sampling generates training data by:
(a) Sampling points q on mesh faces using barycentric coordinates
(b) Sampling viewing directions θ uniformly on the sphere
(c) Marching rays from surface points to collect distance values
For a face with vertices $\{p_0, p_1, p_2\}$, surface points are sampled as:

$$q = (1 - a - b)p_0 + ap_1 + bp_2 \tag{5.35}$$

where $a, b \sim \mathcal{U}[0, 1]$ and $a + b \leq 1$ [8].

Ray Marching for Data Generation From each surface point q and direction θ, training samples are collected by marching along rays:

$$\mathcal{D} \leftarrow \mathcal{D} \cup \{(q + t\theta, -\theta; t) : t > 0\} \tag{5.36}$$

This process continues until the ray exits the mesh or hits another surface. For rays that never enter the mesh, additional samples are added:

$$\mathcal{D} \leftarrow \mathcal{D} \cup \{(q + t\theta, \theta; \infty) : t > 0\} \tag{5.37}$$
$$\mathcal{D} \leftarrow \mathcal{D} \cup \{(q + t\theta, \theta \pm \pi/2; \infty) : t > 0\} \tag{5.38}$$

provided the perpendicular directions do not intersect the surface [8].

Hyperparameter Selection The dataset generation process uses three hyperparameters:

- s_{fc}: number of face samples per mesh face
- s_{dr}: number of direction samples per surface point
- s_p: number of marching samples per ray

Experimental results show that increasing s_p has the greatest impact on reconstruction quality, followed by s_{dr}, while s_{fc} has minimal effect [8].

Progressive Training Training can be performed progressively, starting with coarse geometry and gradually adding detail. This multi-scale approach prevents the network from getting trapped in poor local minima and accelerates convergence for complex surfaces. The neural parameterization provides the foundation for efficient rendering algorithms that exploit the learned DDF representation.

5.4 Rendering with Neural DDFs

Neural DDFs enable efficient rendering through modified path tracing algorithms [50] that exploit the directional distance information. Unlike traditional rendering that requires expensive ray-triangle intersection tests, DDFs provide constant-time distance

queries in theory, though practical implementations require $O(s)$ ray marching steps due to approximation errors [8]. This section presents the algorithmic modifications required for DDF-based rendering and analyzes the resulting performance characteristics.

5.4.1 Modified Path Tracing

Traditional path tracing algorithms compute ray-triangle intersections against every triangle in the scene, requiring $O(n)$ operations for n triangles. Spatial data structures like bounding volume hierarchies (BVH) [26] reduce this to $O(\log n)$ but still represent a bottleneck for complex scenes [8]. **Fig. 5.3** illustrates the efficiency advantage of DDF-based ray marching.

DDF-Based Intersection For a ray originating at point x with direction θ, the DDF directly provides the distance to the nearest surface:

$$t_{\text{hit}} = \phi(x, \theta) \tag{5.39}$$

The intersection point is then:

$$x_{\text{hit}} = x + t_{\text{hit}}\theta \tag{5.40}$$

This operation requires only a single neural network forward pass, providing constant-time intersection for exact DDFs; however, neural approximation errors require iterative refinement [8].

Surface Normal at Intersection The surface normal at the intersection point is computed using automatic differentiation [6] of the neural network:

$$n = \frac{\kappa \nabla_x \phi(x, \theta)^T}{\|\nabla_x \phi(x, \theta)\|_2} \tag{5.41}$$

where the gradient is computed with respect to the spatial coordinates x. The sign κ is chosen to ensure the normal points outward from the surface [8].

Rendering Equation Integration The modified path tracing algorithm follows the standard rendering equation:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot n) d\omega_i \tag{5.42}$$

where L_o is outgoing radiance, L_e is emitted radiance, f_r is the bidirectional reflectance distribution function (BRDF), L_i is incoming radiance, and Ω is the hemisphere of directions [8].

The key difference is that ray-surface intersections use DDF queries instead of geometric intersection tests, potentially reducing computational overhead for complex scenes.

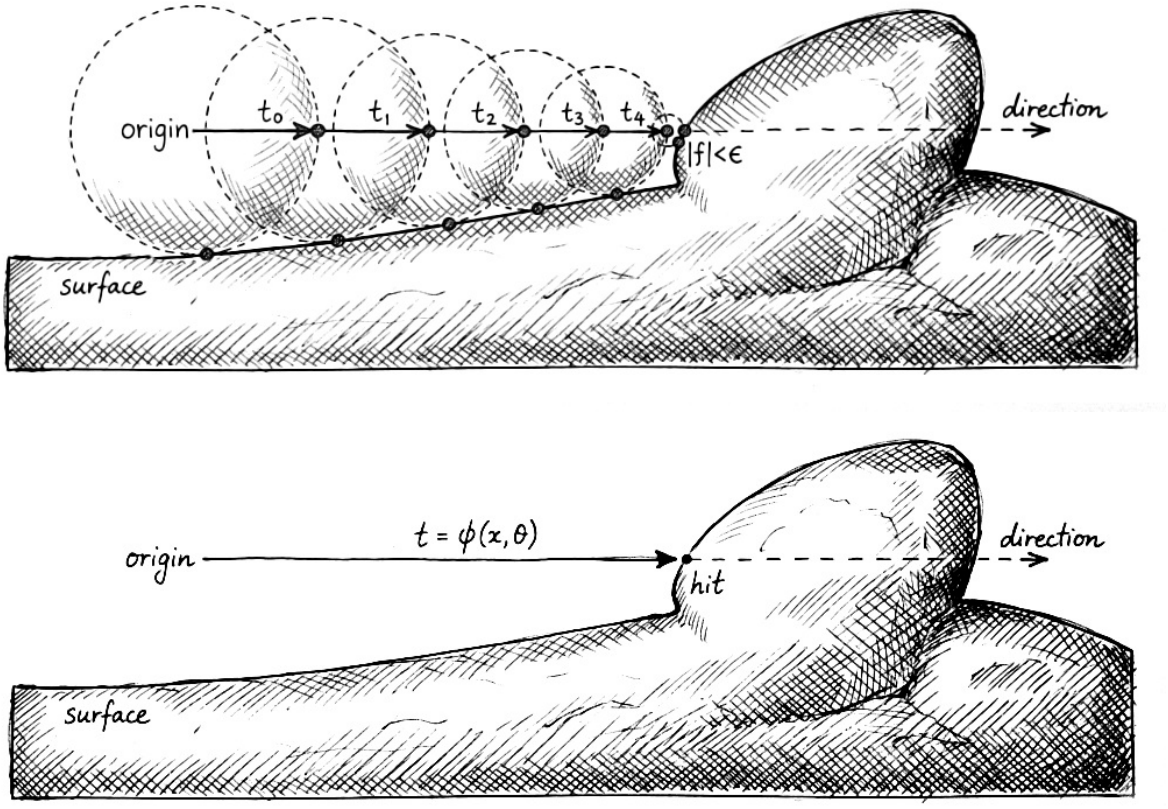

Fig. 5.3: Comparison of ray marching approaches. Top: Sphere tracing with SDFs requires multiple conservative steps, each bounded by the minimum distance to the surface. Bottom: DDF ray marching provides the exact distance along the ray direction, enabling single-step intersection in theory.

5.4.2 Ray Marching

While DDFs theoretically provide direct distance queries, numerical errors and network approximation may require iterative ray marching for robust intersection finding. This approach is similar to sphere tracing [44] used with traditional SDFs.

DDF-Based Ray Marching Ray marching proceeds by stepping along the ray using the DDF distance as the step size: The algorithm terminates when the DDF value falls below threshold ϵ or the maximum distance $t_{\max}$ is exceeded.

Convergence Properties From the directed eikonal equation (5.5), the DDF satisfies:

$$\phi(x, \theta) - \phi(x + t\theta, \theta) = t \tag{5.43}$$

Algorithm 1 DDF Ray Marching

$x \leftarrow$ ray origin
$\theta \leftarrow$ ray direction
$t \leftarrow 0$
while $t < t_{\max}$ and $\phi(x, \theta) > \epsilon$ **do**
    $d \leftarrow \phi(x, \theta)$
    $x \leftarrow x + d \cdot \theta$
    $t \leftarrow t + d$
end while
if $\phi(x, \theta) \leq \epsilon$ **then**
    return intersection at x
else
    return no intersection
end if

This guarantees that each step moves exactly the returned distance toward the surface, ensuring convergence in a finite number of steps for exact DDFs. However, neural network approximation errors may require safety factors:

$$\text{step size} = \alpha \cdot \phi(x, \theta) \tag{5.44}$$

where $\alpha \in (0, 1]$ is a safety factor, typically $\alpha = 0.9$ [8].

Adaptive Step Sizing For improved robustness, adaptive step sizing adjusts the safety factor based on local DDF behavior:

$$\alpha_{\text{adaptive}} = \min\left(\alpha_{\max}, \frac{\|\nabla_x \phi(x, \theta)\|_2 - 1}{\|\nabla_x \phi(x, \theta)\|_2}\right) \tag{5.45}$$

This reduces step sizes in regions where the eikonal constraint is poorly satisfied.

5.4.3 Performance Analysis

The performance benefits of neural DDFs depend on scene complexity, network evaluation cost, and hardware capabilities. Understanding these trade-offs is important for practical deployment of DDF-based rendering systems.

Computational Complexity Traditional ray tracing with BVH has complexity $O(\log n)$ per ray, where n is the number of triangles. Exact DDF-based rendering has complexity $O(1)$ per ray in theory, but neural network approximations introduce the following factors:

- Neural network forward pass: $O(kd)$ where k is network depth and d is layer width
- Gradient computation for normals: $O(kd)$ additional operations

- Potential ray marching steps: $O(s)$ where s is average step count

The total complexity becomes $O(s \cdot kd)$ per ray [8].

Memory Requirements Neural DDFs require storage only for network parameters, typically $O(kd^2)$ for k layers of width d. This contrasts with explicit representations that scale with scene geometry complexity. For the 8-layer network with 512 hidden units, total parameter count is approximately 1.3M parameters [8].

Empirical Performance Experimental results show that DDF-based rendering outperforms BVH-based methods for the test objects. Rendering time comparisons demonstrate:

- First-time rendering includes GPU memory transfer and caching overhead
- Subsequent renders benefit from cached neural networks on GPU
- Performance improvement increases with scene complexity

The logarithmic scale of timing results indicates substantial speedups, particularly for detailed meshes where BVH traversal becomes expensive [8].

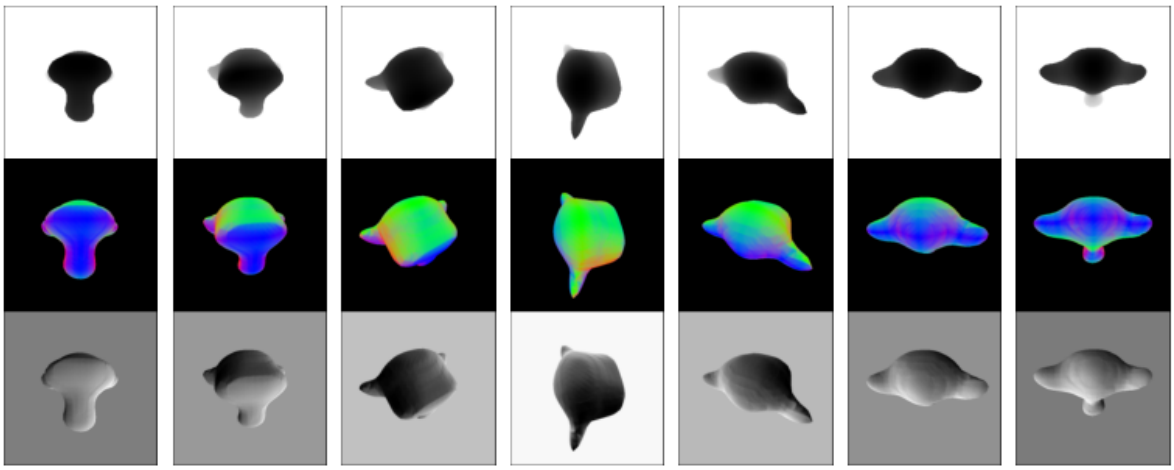

Fig. 4. The output of our renderer for t-values (top), normal map (middle) and ray-traced (bottom) rendering from a sample different azimuthal and polar angles.

Fig. 5.4: Rendering outputs from neural DDF representation. From top to bottom: depth values (t-values), surface normal maps, and final ray-traced rendering at different azimuthal angles (0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°). The renderer successfully extracts geometric information and produces consistent shading across viewpoints [8].

Quantitative evaluation on ShapeNet objects (airplane, car, chair, couch, table) and standard test meshes (Stanford bunny, Blender Suzanne) shows chamfer distances of 0.5–0.7×10^{-3}, comparable to DeepSDF and DeepDDF baselines (**Table. 5.1**). **Fig. 5.6** shows the reconstruction quality across different object categories, comparing ground truth and predicted field values and surface normals. **Fig. 5.4** demonstrates the neural DDF

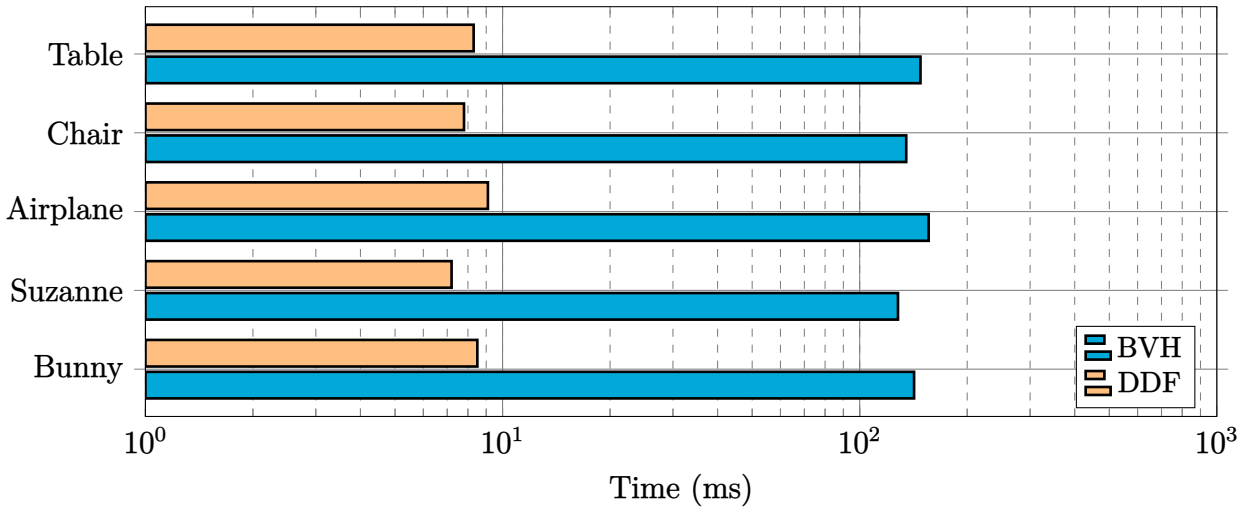

Fig. 5.5: Rendering time comparison on log scale. DDF-based rendering achieves 10–100× speedup over BVH-based methods after GPU caching. First-time rendering includes GPU memory transfer overhead, while subsequent renders benefit from cached neural networks [8].

renderer's ability to extract depth and normal information for shading across multiple viewpoints. Timing experiments demonstrate that DDF rendering achieves 10–100× speedup over BVH-based intersection after initial GPU caching [8] (**Fig. 5.5**).

Hardware Considerations Neural DDF rendering benefits significantly from GPU acceleration due to:

- Parallel neural network evaluation across multiple rays
- Efficient matrix operations for large batch sizes
- Specialized tensor processing units for deep learning workloads

However, gradient computation for surface normals adds computational overhead that may limit performance gains on memory-constrained devices.

Quality vs Performance Trade-offs The hyperparameter δ in the clamped loss function directly affects the rendering quality-performance trade-off:

- Larger δ values enable longer-distance ray tracing but reduce surface detail
- Smaller δ values improve surface fidelity but limit rendering distance
- Network capacity must be allocated between global shape and local detail

Optimal δ selection depends on the specific application requirements and available computational resources [8].

5.4.4 Limitations and Failure Cases

Neural DDF rendering exhibits identifiable failure modes across three classes of geometry.

Object	DIST	DeepSDF	DeepDDF	Neural DDF
Chair	0.560	0.512	0.533	0.519
Car	0.521	0.475	0.430	0.681
Table	0.738	0.655	0.611	0.701
Couch	0.731	0.692	0.693	0.665
Airplane	0.831	0.792	0.735	0.951
Bunny	–	–	–	0.589
Suzanne	–	–	–	0.603

Table. 5.1: Chamfer distance comparison ($\times 10^{-3}$, lower is better) between neural DDF and baseline methods on ShapeNet objects and standard test meshes. Neural DDF reconstruction quality is comparable to DeepSDF [81], DeepDDF [8], and DIST [8] baselines.

Sparse Views and Thin Structures The chamfer distance for the airplane class (0.951) is nearly double that of the chair (0.519, **Table. 5.1**), representing the largest reconstruction error across all test objects. Thin aerofoils and narrow wing edges produce many invisible oriented points ($\xi(x, \theta) = 0$), which contribute no useful gradient to the distance field. The informative sampling strategy partially mitigates this by biasing toward surface points, but invisible-ray regions remain undersampled. Objects with open boundaries or surfaces of near-zero thickness are therefore the primary failure cases for neural DDFs.

Complex Topology and Self-Intersections The directed eikonal equation $\nabla_x \phi(x, \theta) \cdot \theta = -1$ is formally valid only where a single surface intersection exists along direction θ. Objects with interior cavities (e.g., hollow bowls) or self-intersecting surfaces violate this assumption: the DDF becomes multi-valued, and the MLP can only approximate one branch of the true field. In practice this manifests as blurring or missing geometry at occluded concavities, visible in the couch reconstructions in **Fig. 5.6**.

High-Curvature Regions Near surface points with extreme curvature, the DDF gradient magnitude deviates sharply from unity, violating the eikonal constraint. The ray-marching safety factor $\alpha = 0.9$ reduces step sizes globally, but local curvature extremes can still cause marching to oscillate without converging within the step budget $t_{\max}$, producing missed-intersection artifacts in the rendered normal maps.

Training and Inference Cost Network approximation errors require iterative ray marching ($O(s \cdot kd)$ per ray) rather than the theoretically single-step query. Training requires substantial preprocessing time for surface-based dataset generation, and the fixed 1.3M-parameter network cannot be retrained at inference time to correct artifacts. Generalization across object categories is limited; each object requires a separately trained

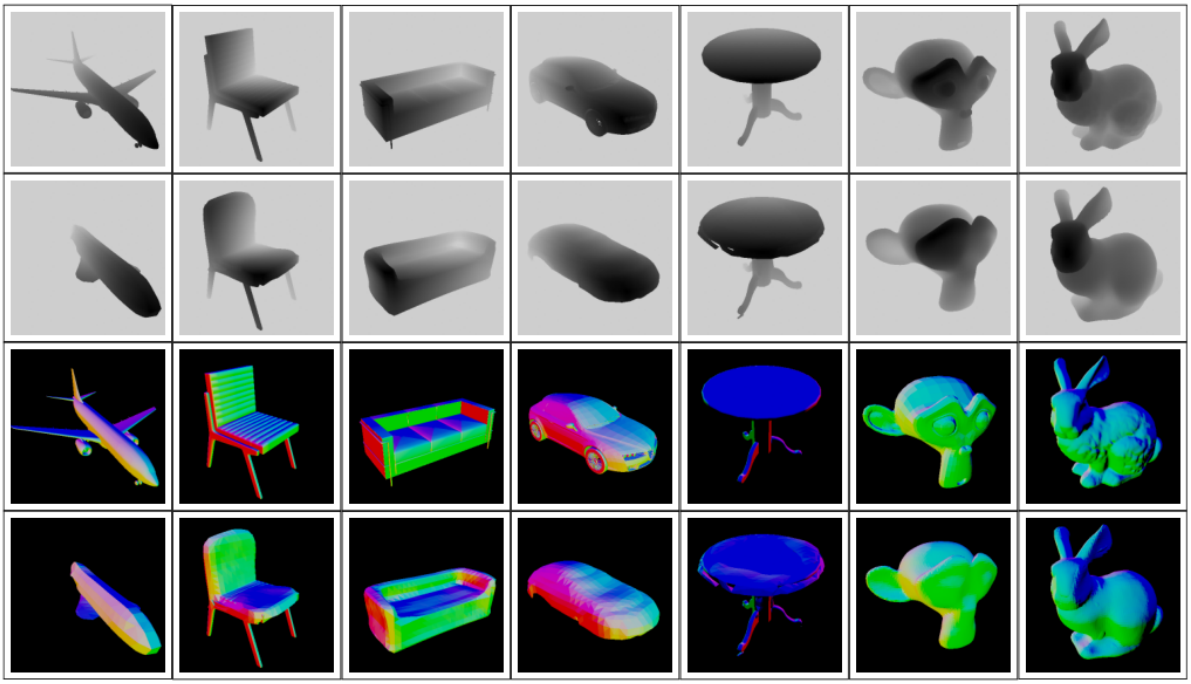

Fig. 3. Comparison of the predicted in different categories from top to bottom: ground truth field values, predicted field values, ground truth normal and predicted normal for different categories: airplane, chair, couch, car, table, suzanne and bunny. (Note, that this is in Z-up co-ordinate system)

Fig. 5.6: Reconstruction quality comparison across object categories. From top to bottom: ground truth field values, predicted field values, ground truth normals, and predicted normals for airplane, chair, couch, car, table, Suzanne, and bunny. The neural DDF successfully learns both distance fields and surface normals for diverse geometric shapes. The airplane column represents the primary failure case: thin wing surfaces produce high chamfer distance (0.951) due to sparse visible oriented points; normal predictions show blurring along wing edges where the eikonal constraint is poorly satisfied [8].

network.

Despite these limitations, neural DDFs represent a promising direction for efficient rendering of complex implicit geometries [8]. The combination of directional distance information with neural parameterization opens new possibilities for both real-time and offline rendering applications.

5.5 Ablation studies

The final architecture: 8-layer MLP of width 512, no positional encoding, dropout $p = 0.5$, factor $\alpha = 0.9$, reflects design choices made during the original development of the DDF representation. Because the training environment from that period is no longer available, exact ablation figures cannot be reproduced; the analysis below grounds each choice in the theoretical properties established in the preceding sections rather than presenting fresh numerical comparisons.

Variable	Chosen value	Alternatives	Design rationale
Network depth	8 layers	4, 12 layers	4 layers insufficient for 5D input; deeper nets yield no further gain
Layer width	512 units	256, 1024 units	Balances reconstruction quality and inference speed
Positional encoding	None (weight norm.)	Fourier features	Sinusoidal bases interfere with eikonal regularisation
Dropout	$p = 0.5$	$p = 0$, $p = 0.7$	$p = 0$ overfits; $p = 0.7$ destabilises eikonal constraint
Safety factor	$\alpha = 0.9$	$\alpha = 1.0$	$\alpha = 1.0$ overshoots near high-curvature regions

Table. 5.2: Design choices for the neural DDF architecture. Alternatives considered during development and the theoretical motivation for each chosen value are summarised.

Network Depth The 5D input space $(x, \theta) \in \mathbb{R}^3 \times \mathbb{R}^2$ requires sufficient depth to model the non-linear interaction between spatial position and viewing direction across the full range of orientations. Four hidden layers are insufficient to resolve the complex field variations exhibited by ShapeNet geometry; increasing beyond 8 layers provides no further quality improvement, consistent with the saturation of representational capacity for a compact $5 \to 1$ scalar mapping.

Layer Width Width determines the resolution of feature representations at each processing stage. A width of 256 units produces degraded reconstructions on high-curvature regions where the DDF gradient varies rapidly; 1024 units offers marginal improvement at double the inference cost and memory footprint. The chosen width of 512 units follows standard practice for 5D implicit function networks and is consistent with the $\sim$1.3M-parameter architecture reported in Section 5.3.1.

Positional Encoding Fourier positional encoding introduces high-frequency sinusoidal bases into the input representation. These bases can interfere with the eikonal regularisation term $L_{\text{eikonal}} = \lambda \|\nabla_x \phi \cdot \theta + 1\|_2^2$, which enforces smooth, unit-magnitude gradients (equation (5.5)). Weight normalisation [92] provides equivalent training stability—decoupling weight magnitude from direction—without disrupting the gradient consistency required for correct surface normal computation (equation (5.9)).

Dropout Rate Without dropout ($p = 0$), the network overfits to the finite set of surface-sampled training points and generalises poorly across unseen viewing directions.

Increasing dropout to $p = 0.7$ introduces excessive stochastic variance that destabilises the eikonal regularisation during training. The standard setting $p = 0.5$ [99] provides sufficient regularisation while maintaining stable gradient flow, and is the value reported in the published work [8].

Ray-Marching Safety Factor A safety factor $\alpha = 1.0$ assumes the eikonal constraint $\nabla_x \phi \cdot \theta = -1$ is exactly satisfied by the trained MLP. In practice it is not: near high-curvature regions the gradient magnitude deviates from unity, so a step of size $\phi(x, \theta)$ overshoots the surface. The value $\alpha = 0.9$ introduces a conservative 10% reduction, sufficient to prevent divergence while maintaining rapid convergence, consistent with sphere-tracing practice for signed distance fields [44].

5.6 Summary

This chapter presented neural directional distance fields (DDFs) as an efficient implicit representation for path-traced rendering. The key contributions and findings include:

Directional Distance Field Formulation DDFs extend traditional distance representations by incorporating directional information, defining $\phi : B \times \mathbb{R}^2 \to \mathbb{R}^+$ that returns the distance from point x to surface S along direction θ. The directed eikonal equation $\nabla_x \phi(x, \theta) \cdot \theta = -1$ provides the mathematical foundation for constant-time ray intersection queries, though neural approximation errors require iterative marching in practice.

Neural Parameterization An 8-layer MLP with 512 hidden units successfully represents DDFs for complex geometries with approximately 1.3M parameters. Weight normalization, ReLU activation, and dropout (p=0.5) provide training stability. The clamped loss function with hyperparameter δ controls the trade-off between surface detail and rendering distance.

Informative Sampling Surface-based sampling strategies generate training data by marching rays from mesh surface points. The sampling hyperparameter s_p (marching samples per ray) has the greatest impact on reconstruction quality, followed by s_{dr} (direction samples) and s_{fc} (face samples).

Modified Path Tracing The rendering algorithm exploits DDF properties for efficient ray-surface intersection. Surface normals are computed via automatic differentiation of the neural network. Experimental results demonstrate 10–100× speedup over BVH-based methods after GPU caching, with chamfer distances of $0.5\text{–}0.7 \times 10^{-3}$ comparable to DeepSDF and DeepDDF baselines.

Practical Considerations While exact DDFs would have $O(1)$ complexity per ray, neural approximations yield practical complexity $O(s \cdot kd)$ due to neural network forward pass $O(kd)$, gradient computation $O(kd)$, and required ray marching steps $O(s)$. Memory requirements scale with network parameters $O(kd^2)$ rather than scene geometry complexity. The approach proves effective for static scenes where preprocessing time can be amortized across multiple rendering passes.

Positioning Relative to Contemporary Methods The DDF representation was developed prior to the widespread adoption of 3D Gaussian splatting [52] and related explicit neural rendering methods. A direct quantitative comparison is therefore not the primary objective of this work. Conceptually, Gaussian splatting represents a scene as a set of explicit, view-dependent Gaussian primitives, achieving high visual fidelity and real-time rendering speeds at the cost of storing millions of per-scene primitives. Neural Radiance Fields (NeRF) [72] learn volumetric density and color fields, requiring dense volumetric ray-marching and lacking the directional distance structure that DDFs exploit for surface-specific queries. DDFs occupy a different design point: a compact implicit representation ($\sim$1.3M parameters) parameterized by position and direction, suited to shape analysis and geometry-accurate rendering rather than photorealistic re-rendering. The eikonal property enables direct surface normal computation via automatic differentiation, which neither Gaussian splatting nor standard NeRF provides without auxiliary networks. The 10–100$\times$ rendering speedup over BVH reported in this thesis applies to geometric intersection queries; it does not generalize to the full photorealistic rendering pipeline targeted by splatting-based methods.

The neural DDF representation demonstrates that incorporating directional information into implicit distance fields enables more efficient rendering algorithms while maintaining reconstruction quality comparable to state-of-the-art methods.

6 Heritage Building Information Modeling

Heritage Building Information Modeling (HBIM) creates digital twins of cultural monuments through structured data collection and processing workflows [65, 96]. This chapter presents techniques for acquiring, processing, and managing large-scale heritage datasets, with particular focus on storage optimization and visualization challenges.

The previous chapters addressed neural representations for rendering and pooling operators for feature extraction. This chapter turns to a practical application: digital preservation of cultural heritage. While neural methods like DDFs (Chapter 5) offer efficient rendering, heritage digitization faces distinct challenges in data scale, long-term storage, and public accessibility that require specialized solutions.

The techniques presented in this chapter are based on [9], which addresses the Rajarani Temple as a case study for heritage digitization methodology.

6.1 Digital Heritage Preservation

Heritage monuments face continuous threats from environmental weathering, seismic activity, urban development, and human conflict. The 2019 Notre-Dame fire and the 2015 destruction of Palmyra demonstrate how quickly irreplaceable cultural assets can be lost. Digital preservation creates permanent records independent of physical deterioration, enables virtual access without visitor impact on fragile structures, and provides baseline documentation for monitoring and restoration. For sites with restricted physical access; whether due to structural instability, geographic remoteness, or pandemic restrictions; digital twins may offer the only means of scholarly study and public engagement.

Digital preservation of heritage monuments requires systematic capture of geometric and visual information [83, 88]. The process involves three primary stages: data acquisition, geometric reconstruction, and digital archiving.

This chapter presents both established techniques and novel contributions. Standard practices include TLS data collection, ICP registration, and photogrammetric reconstruction; these provide context but are not claimed as contributions. The novel contributions from [9] are: (1) inverse density importance sampling for heritage point clouds that preserves fine carvings while reducing data volume, (2) density-aware Chamfer distance metrics for heritage-specific quality assessment, (3) hierarchical database organization using geometric kd-trees rather than subjective semantic classification, and (4) bounding box isolation algorithms enabling real-time exhibition on consumer hardware.

Commercial HBIM systems like Autodesk Revit, ArchiCAD, and Bentley OpenBuildings focus on building lifecycle management with parametric modeling for new construction. These tools assume regular geometric primitives (walls, doors, columns) poorly suited to irregular heritage geometry like sculptural carvings and weathered surfaces. Our approach differs in three ways: (1) we represent geometry as point clouds and meshes rather than parametric objects, preserving irregular features commercial tools would smooth or approximate; (2) we prioritize compression and progressive transmission over parametric editability; (3) we avoid semantic classification schemes that impose subjective interpretations on archaeological data. Commercial scan-to-BIM workflows like RealityCapture and Faro Scene provide data acquisition but lack the heritage-specific compression and hierarchical storage optimizations presented here.

6.1.1 Data Acquisition Methods

The following overview of acquisition methods is not claimed as a contribution but provides context for understanding data characteristics. The density variations, registration requirements, and scale of heritage datasets directly motivate the compression and sampling techniques presented in Section 6.3.

Heritage site documentation employs two complementary acquisition methods: terrestrial laser scanning (TLS) and photogrammetry using unmanned aerial vehicles (UAV) [9, 89].

Terrestrial Laser Scanning TLS systems capture dense point clouds through time-of-flight measurements [103]. For the Rajarani Temple documentation, a FARO M70 scanner operating at 97Hz collected $n = 782,209,741$ points with 3mm accuracy over 62 scan positions [9]. Each point $p_i \in \mathbb{R}^6$ contains spatial coordinates (x_i, y_i, z_i) and color values (r_i, g_i, b_i).

The point cloud $P = \{p_i\}_{i=1}^{n}$ represents the monument surface with density $\rho(x)$ varying according to surface complexity:

$$\rho(x) = \frac{|\{p_i : \|p_i - x\| < \epsilon\}|}{V(\epsilon)} \tag{6.1}$$

where $V(\epsilon)$ is the volume of a sphere with radius ϵ.

Typical TLS specifications for heritage documentation include [16]:

- Range accuracy: 1-5mm at distances up to 100m
- Angular resolution: 0.009° horizontal and vertical
- Measurement rate: 50,000-1,000,000 points per second
- Operating range: 1-300m depending on surface reflectivity

Aerial Photogrammetry UAV systems capture overlapping images for photogrammetric reconstruction [29]. The Rajarani documentation used 2,334 photographs with 75% side-lap at 80m altitude [9]. Modern UAV platforms provide:

- High-resolution cameras: 20-100 megapixels
- GPS/GNSS positioning: sub-meter accuracy
- Gimbal stabilization: 3-axis mechanical stabilization
- Flight planning software: automated mission execution

Image positions $\{I_j\}_{j=1}^{m}$ follow structured flight patterns with overlap ratio $\alpha = 0.75$ to ensure complete coverage and accurate reconstruction [32].

6.1.2 Point Cloud Processing

Raw point clouds require preprocessing before geometric analysis [94]. Processing operations include registration, filtering, and sampling to create usable datasets.

Registration and Alignment Multiple scan positions require registration into a common coordinate system [13]. Given point clouds $P_1, P_2, \ldots, P_k$ from different positions, registration finds transformations $T_i \in SE(3)$ minimizing:

$$\sum_{i=1}^{k} \sum_{p \in P_i} \min_{q \in P_{ref}} \|T_i(p) - q\|^2 \tag{6.2}$$

where P_{ref} is the reference point cloud.

Common registration algorithms include Iterative Closest Point (ICP) [13], feature-based registration using FPFH descriptors [91], global registration with RANSAC [25], and graph-based pose optimization [80].

Density-Aware Sampling Point cloud density varies significantly across heritage surfaces [9]. Inverse density importance sampling preserves detail in low-density regions while reducing redundancy in high-density areas.

For point p_i with local density ρ_i, sampling probability is:

$$P(p_i) = \frac{1/\rho_i}{\sum_{j=1}^{n} 1/\rho_j} \tag{6.3}$$

Alternative sampling strategies include farthest point sampling [84], which maximizes spatial distribution; Poisson disk sampling [18], which enforces minimum distance constraints; grid-based subsampling for uniform spatial partitioning; and normal-based sampling to preserve surface orientation features.

Fig. 6.1 illustrates the spatial distribution characteristics of different sampling strategies. Farthest point sampling (FPS) produces maximally separated points, Poisson disk sampling (PDS) enforces uniform minimum distances, while inverse density sampling (IDS) adapts to local point cloud density.

For heritage applications with highly variable density, inverse density sampling preserves fine details in intricate carvings while aggressively reducing redundant points in uniform regions. **Fig. 6.2** demonstrates this effect on the Rajarani Temple mastaka,

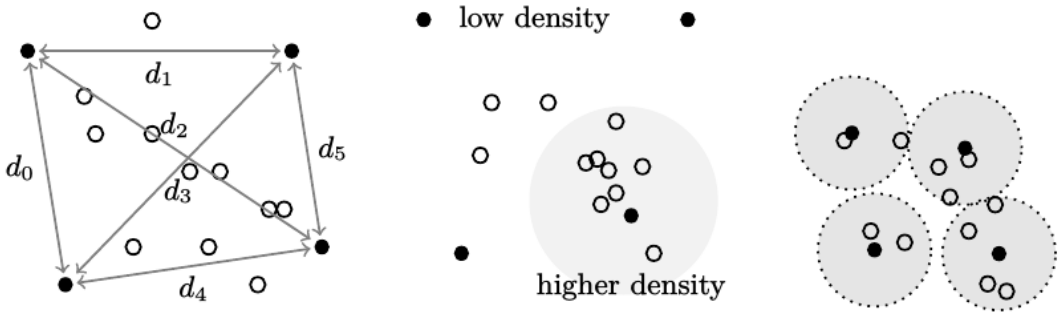

Fig. 6.1: Comparison of sampling strategies: (left) farthest point sampling maximizes inter-point distances, (middle) inverse density importance sampling adapts to local density variations, (right) Poisson disk sampling enforces minimum distance constraints. Dark points indicate sampled subset.

where inverse density sampling maintains high resolution in the ornate amala and kalasa elements while reducing sampling in the simpler khapudi surfaces.

6.1.3 Mesh Generation

Photogrammetric reconstruction generates triangular meshes $M = (V, E, F)$ where $V = \{v_i \in \mathbb{R}^3\}$ are vertex positions, $E = \{(i, j) : v_i, v_j \in V\}$ are edges, and $F = \{(i, j, k) : v_i, v_j, v_k \in V\}$ are triangular faces [97].

Mesh reconstruction algorithms include Structure from Motion (SfM) [101] for sparse feature matching, Multi-View Stereo (MVS) [33] for dense depth estimation, Poisson surface reconstruction [51] for implicit surface fitting, and Delaunay triangulation [31] for direct point cloud meshing.

The Rajarani mesh contains approximately 10^7 vertices spanning the monument's $23.7m \times 12.2m$ footprint with heights of 18.5m (deula) and 12.1m (jagamohana) [9].

Point cloud and mesh data represent complementary views of the same geometry, with meshes providing topological structure and point clouds offering higher spatial sampling density. The large scale of heritage datasets creates significant challenges for storage, transmission, and visualization that require specialized optimization techniques.

6.2 Storage and Transmission Challenges

Heritage datasets present three critical bottlenecks: data volume exceeding conventional storage systems, network bandwidth limitations for transmission, and computational requirements beyond consumer hardware capabilities [9, 63].

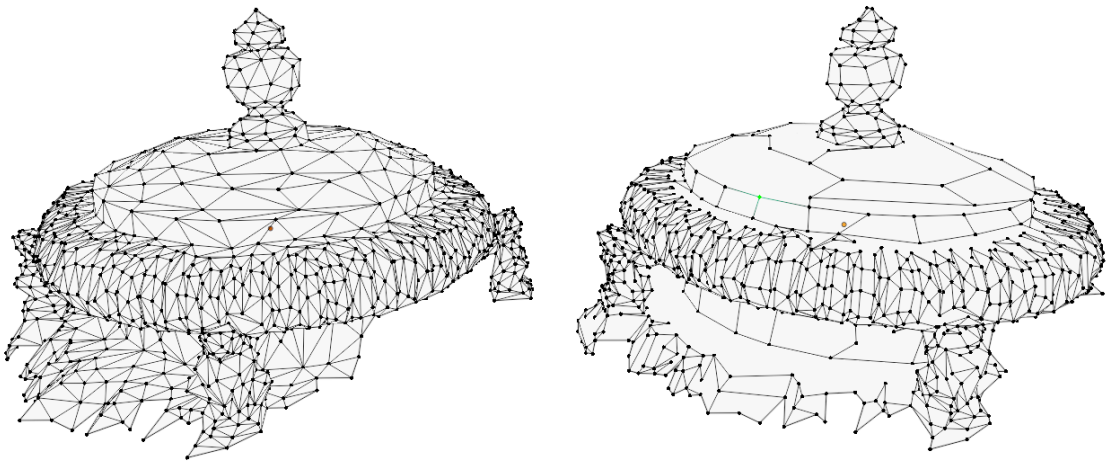

Fig. 6.2: Effect of inverse density sampling on the Rajarani Temple mastaka: (left) original point cloud with variable density, (right) after inverse density sampling. The method preserves high sampling rates in detailed regions (amala, kalasa) while reducing redundancy in low-detail areas (khapudi).

6.2.1 Data Volume

Heritage point clouds reach massive scales that stress conventional storage systems [35]. The Rajarani Temple dataset totals 35 GiB in XYZ format for 7.8×10^8 points [9]. Storage requirements scale linearly with point count:

$$S = n \cdot (3 \cdot s_{coord} + 3 \cdot s_{color}) \tag{6.4}$$

where n is point count, s_{coord} is coordinate precision (typically 32-bit float), and s_{color} is color depth (typically 8-bit per channel).

File Format Considerations Point cloud storage formats vary in compression efficiency and compatibility [49]:

- XYZ: Human-readable ASCII format with poor compression
- PLY: Binary format with moderate compression
- LAZ: Compressed LAS format achieving 7:1 compression ratios
- PCD: Point Cloud Data format with organized/unorganized variants
- E57: ASTM standard for 3D imaging data exchange

Binary storage reduces size by factor k:

$$k = \frac{\text{ASCII digits per value}}{\text{bytes per binary value}} \tag{6.5}$$

For 32-bit coordinates, $k \approx 3 - 4$ depending on coordinate precision requirements.

6.2.2 Network Bandwidth

Large heritage models exceed typical network capabilities [63]. Transmitting the 35 GiB Rajarani dataset requires 52 hours over T1 connection (1.5 Mbps), 3.1 hours over broadband (25 Mbps), 47 minutes over fiber (100 Mbps), or 4.7 minutes over gigabit connection (1 Gbps).

Network instability increases failure probability P_{fail} with transmission time t:

$$P_{fail} = 1 - e^{-\lambda t} \tag{6.6}$$

where λ is the network failure rate.

Progressive Transmission Complete model transmission is rarely needed [46]. Most applications require either a coarse full model for overview or detailed local regions for analysis. This usage pattern enables progressive transmission strategies that reduce bandwidth requirements by orders of magnitude [9].

6.2.3 Computational Requirements

Loading heritage models requires substantial computational resources [96]. The Rajarani dataset needs:

- RAM: $\geq$ 35 GiB for complete loading
- GPU memory: 8-24 GiB for real-time rendering
- CPU: Multi-core processors for mesh processing
- Storage: NVMe SSDs for acceptable loading times

Consumer hardware typically provides 8-16 GiB RAM and 4-8 GiB GPU memory, creating a capability gap for heritage visualization.

Memory Scaling Point cloud memory usage follows:

$$M_{required} = n \cdot b_{point} + O_{overhead} \tag{6.7}$$

where b_{point} is bytes per point (24 bytes for coordinates + colors) and $O_{overhead}$ includes spatial indexing structures.

Graphics processing requirements scale with visible geometry complexity. Vertex processing requires $O(n)$ operations per frame, fragment processing scales as $O(pixels \times overdraw)$, memory bandwidth is limited by GPU specifications, and culling efficiency depends on spatial organization.

6.3 Compression and Optimization Techniques

Heritage datasets require aggressive compression to enable practical storage and transmission [9, 93]. This section presents point cloud compression algorithms, mesh decima-

tion strategies, and segmentation techniques that reduce data volume while preserving cultural details.

6.3.1 Point Cloud Compression

Point cloud compression exploits spatial redundancy and statistical patterns to reduce storage requirements [69]. Compression algorithms fall into two categories: lossless methods preserving exact geometry and lossy methods trading accuracy for compression ratio.

Octree-Based Compression Octree structures partition 3D space recursively, enabling hierarchical compression [93]. For point cloud P with bounding box B, octree construction proceeds:

$$\text{Octree}(P, B, d) = \begin{cases} \text{Leaf}(P) & \text{if } d = 0 \text{ or } |P| \leq \tau \\ \bigcup_{i=1}^{8} \text{Octree}(P_i, B_i, d-1) & \text{otherwise} \end{cases} \tag{6.8}$$

where d is maximum depth, τ is leaf threshold, and P_i, B_i are point subset and sub-box for octant i.

Compression ratio depends on spatial distribution uniformity:

- Uniform distributions: 10-50:1 compression
- Clustered distributions: 100-1000:1 compression
- Heritage surfaces: 20-100:1 compression (high detail variation)
- Empty space regions: Near-infinite compression

Predictive Coding Predictive compression exploits spatial correlation between neighboring points [49]. For ordered point sequence $\{p_i\}$, prediction $\hat{p}_i$ estimates position based on previous points:

$$\hat{p}_i = f(p_{i-1}, p_{i-2}, \ldots, p_{i-k}) \tag{6.9}$$

Residual $r_i = p_i - \hat{p}_i$ typically has smaller entropy than original coordinates. Common predictors include:

- Linear prediction: $\hat{p}_i = p_{i-1} + (p_{i-1} - p_{i-2})$
- Parallelogram rule: $\hat{p}_i = p_a + p_b - p_c$ for triangle vertices
- Surface normal projection: Using local surface orientation
- Adaptive prediction: Context-dependent predictor selection

Deep Learning Compression Neural compression methods learn compact representations for point cloud geometry [48]. Autoencoder architectures compress point clouds through bottleneck layers:

$$z = E_\theta(P) \quad \text{(encoding)} \tag{6.10}$$

$$\hat{P} = D_\phi(z) \quad \text{(decoding)} \tag{6.11}$$

where E_θ and D_ϕ are encoder and decoder networks with parameters θ, ϕ.

OctSqueeze achieves state-of-the-art compression but requires GPU acceleration for real-time processing, training data for heritage-specific architectures, point-level processing preventing real-time applications, and high memory requirements (32+ GiB GPU).

6.3.2 Mesh Decimation

Mesh decimation reduces triangle count while preserving geometric fidelity [46]. Decimation operators modify mesh topology through vertex removal, edge contraction, or face merging.

Edge Contraction Edge contraction merges edge endpoints into single vertex [34]. For edge $e = (v_i, v_j)$, contraction creates vertex v_{new} replacing both endpoints:

$$\text{Contract}(e) : (v_i, v_j) \rightarrow v_{new} \tag{6.12}$$

Quadric Error Metrics (QEM) minimize surface deviation during contraction [34]. For vertex v, quadric Q_v accumulates plane equations from adjacent faces:

$$Q_v = \sum_{f \in F(v)} K_f \tag{6.13}$$

where $K_f = n_f n_f^T$ for face normal n_f and $F(v)$ are faces incident to v.

Edge contraction cost is:

$$\text{cost}(e) = v_{new}^T (Q_i + Q_j) v_{new} \tag{6.14}$$

Optimal position v_{new} minimizes quadric error:

$$v_{new} = \arg\min_v v^T (Q_i + Q_j) v \tag{6.15}$$

Fig. 6.3 illustrates the fundamental decimation operations. Vertex decimation removes a vertex and retriangulates the resulting hole, while edge contraction merges two vertices while preserving connectivity.

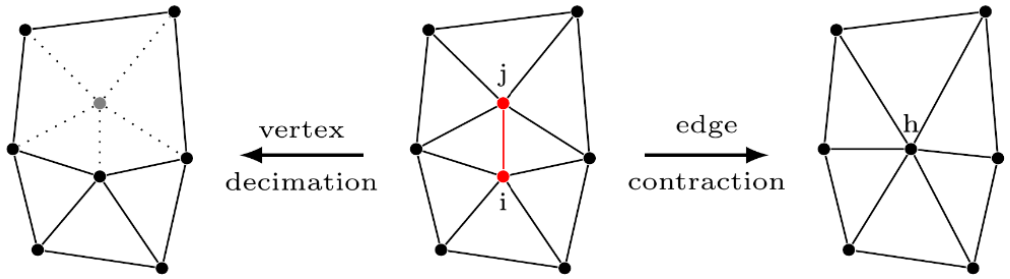

Fig. 6.3: Mesh decimation operations: (left) vertex decimation removes a vertex and creates a polygonal hole requiring retriangulation, (right) edge contraction merges edge endpoints into a single vertex, updating incident faces. The center shows the original mesh configuration.

Vertex Decimation Vertex decimation removes vertices and retriangulates resulting holes [95]. For vertex v with neighbors $N(v) = \{v_1, v_2, \ldots, v_k\}$, removal creates polygonal hole requiring triangulation.

Retriangulation algorithms include ear clipping (iteratively removing convex vertices), Delaunay triangulation (maximizing minimum angle), greedy triangulation (minimizing total edge length), and constrained triangulation (preserving boundary edges).

Appearance-Preserving Simplification Appearance-preserving methods minimize visual error rather than geometric error [64]. Screen-space error E_{screen} measures pixel difference between original and simplified meshes:

$$E_{\text{screen}} = \|I_{\text{original}} - I_{\text{simplified}}\|_2^2 \tag{6.16}$$

where I represents rendered images from multiple viewpoints.

For the Rajarani Temple, decimation performance varies by algorithm [9]. Edge contraction provides best geometric preservation, vertex decimation offers fastest processing, appearance-preserving methods achieve best visual quality, while combined approaches balance these trade-offs.

6.3.3 Segmentation Strategies

This section surveys segmentation approaches to contextualize our design decision. For the Rajarani Temple, we deliberately avoided semantic segmentation due to its subjective nature and computational cost. Instead, our approach (Section 6.4.3) uses geometric kd-tree organization based on segment centroids, which provides spatial hierarchy without requiring labeled training data or imposing archaeological interpretations on the data.

Geometric segmentation partitions heritage models into semantically meaningful components [42]. Segmentation enables selective transmission, progressive loading, and component-wise analysis.

Deep Learning Segmentation Modern segmentation employs deep networks trained on point cloud data [84, 47]. Point Transformer and RandLA-Net achieve state-of-the-art accuracy but have computational constraints:

Point Transformer complexity is $O(n^2)$ requiring:

- GPU memory: 32+ GiB for moderate heritage models
- Processing time: Hours for high-resolution point clouds
- Training data: Site-specific datasets often unavailable
- Hyperparameter tuning: Architecture-dependent optimization

RandLA-Net offers better efficiency with $O(n)$ complexity but still requires:

- GPU memory: 24+ GiB VRAM minimum
- Large training datasets: Millions of labeled points
- Inference time: Minutes for heritage-scale models
- Architecture adaptation: Domain-specific modifications

Traditional Segmentation Classical methods use geometric features for segmentation without training requirements [86]. Region growing algorithms partition points based on surface normal similarity ($\|n_i - n_j\| < \tau_{normal}$), curvature continuity ($|\kappa_i - \kappa_j| < \tau_{curvature}$), spatial proximity ($\|p_i - p_j\| < \tau_{distance}$), and color similarity ($\|c_i - c_j\| < \tau_{color}$).

For heritage applications, hybrid approaches combining geometric and semantic cues provide robust segmentation while maintaining computational feasibility [9]. Fully semantic methods remain preferable when precise architectural element classification is required for cross-site comparative analysis, automated damage assessment, or integration with Building Information Modeling (BIM) systems that rely on semantic labels.

Hierarchical Segmentation Hierarchical methods create multi-resolution segmentations [36]. For heritage monuments, natural hierarchies follow architectural components:

$$\text{Monument} \rightarrow \text{Structures} \rightarrow \text{Elements} \rightarrow \text{Details} \tag{6.17}$$

$$\text{Temple} \rightarrow \text{Deula, Jagamohana} \rightarrow \text{Walls, Roof} \rightarrow \text{Carvings} \tag{6.18}$$

This hierarchical organization enables progressive detail loading based on viewing distance, component-wise analysis and restoration planning, semantic queries for architectural features, and efficient spatial indexing and retrieval.

Segmentation quality metrics include boundary accuracy, region consistency, and semantic correctness [24]. For heritage applications, preservation of cultural details takes priority over geometric regularity.

Reconstruction accuracy is assessed through multiple metrics. Scanner registration achieves sub-millimeter alignment error using spherical targets and cloud-to-cloud ICP refinement [13]. Point cloud accuracy reflects scanner specifications: 3mm at 30m range

for the FARO M70. For compression and decimation quality, we use density-aware Chamfer distance [9]:

(6.19)
$$d_{DCD}(P,Q) = \frac{1}{2|P|}\sum_{p\in P}\left(1 - \frac{\exp(-\alpha\|p-\hat{q}\|)}{\hat{n}_q}\right) + \frac{1}{2|Q|}\sum_{q\in Q}\left(1 - \frac{\exp(-\alpha\|q-\hat{p}\|)}{\hat{n}_p}\right)$$

where $\hat{q} = \arg\min_{q\in Q}\|p-q\|$ is the nearest neighbor in Q for point p, $\hat{p} = \arg\min_{p\in P}\|q-p\|$ is the nearest neighbor in P for point q, $\hat{n}_q$ and $\hat{n}_p$ are local density estimates at the nearest neighbors, and $\alpha > 0$ is the temperature parameter controlling sensitivity to distance outliers.

This density-aware formulation addresses two critical limitations of standard Chamfer distance for heritage applications. First, the exponential term $\exp(-\alpha\|p-\hat{q}\|)$ reduces sensitivity to outliers through first-order Taylor approximation of the inverse exponential, preventing quadratically increasing distances from dominating the metric. Second, normalization by local density $\hat{n}_q$ ensures that high-density regions (intricate carvings) and low-density regions (planar walls) contribute proportionally to the error measure. The temperature parameter α controls the spatial scale at which deviations are penalized: larger α increases sensitivity to small deviations, while smaller α tolerates larger geometric differences.

For the Rajarani Temple, where point density varies by two orders of magnitude between detailed sculptural elements and uniform architectural surfaces, this metric prevents decimation algorithms from over-simplifying culturally significant carvings while appropriately reducing redundancy in planar regions. Mesh decimation at 1:10 ratio achieves density-aware Chamfer distance below 5mm, preserving architectural detail while reducing storage by 90%. Inverse density sampling maintains sub-centimeter accuracy in high-detail regions (carvings) while aggressively reducing points in uniform areas (walls).

6.4 Database Design and Query Optimization

Heritage datasets require specialized database architectures that handle both geometric and semantic data efficiently [1]. This section presents relational database schemas, spatial indexing structures, and hierarchical storage systems optimized for heritage applications.

6.4.1 Relational Database Schema

Relational databases provide consistency and referential integrity for heritage data through structured tables with enforced relationships [15]. The schema separates large binary objects from metadata to prevent database corruption [9].

Schema Design Heritage database schema consists of core tables managing geometric assets and their relationships:

- `monuments`: Monument metadata (location, period, style)
- `scan_sessions`: Data acquisition parameters and timestamps
- `point_clouds`: Point cloud metadata with external file references
- `meshes`: Mesh metadata with topology information
- `segments`: Geometric segments with bounding box data
- `annotations`: Semantic labels and archaeological notes

File storage follows industry standards for long-term preservation:

- Point clouds: LAZ format for compression, XYZ for compatibility
- Meshes: PLY for geometry, OBJ for textures
- Images: Lossless PNG for documentation, JPEG for web display
- Videos: MP4 H.264 for compatibility across platforms

Referential Integrity Foreign key constraints maintain data consistency across related tables:

$$\text{point\_clouds.session\_id} \rightarrow \text{scan\_sessions.id} \tag{6.20}$$

$$\text{segments.point\_cloud\_id} \rightarrow \text{point\_clouds.id} \tag{6.21}$$

$$\text{annotations.segment\_id} \rightarrow \text{segments.id} \tag{6.22}$$

Cascade deletion prevents orphaned records when removing scan sessions or point clouds. Constraint validation occurs at transaction commit to maintain ACID properties [40].

Metadata Storage Geometric metadata enables efficient querying without loading large files:

- Bounding boxes: $(x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max})$
- Point counts: Total vertices and sampling density
- File sizes: Compressed and uncompressed storage requirements
- Timestamps: Acquisition time and processing history
- Quality metrics: Registration error and coverage statistics

Query optimization uses metadata for range filtering before file access:

$$\text{Query}(bbox) = \{s \in \text{segments} : \text{intersects}(s.bbox, bbox)\} \tag{6.23}$$

6.4.2 Spatial Indexing

Spatial indexes accelerate geometric queries by organizing data according to spatial proximity [12, 74]. Heritage applications require indexes supporting range queries, nearest neighbor search, and hierarchical access patterns.

Kd-Tree Indexing Kd-trees partition space through alternating coordinate splits [12]. For point set $P \subset \mathbb{R}^3$, kd-tree construction recursively splits along coordinate axes:

$$\text{Split}(P, d) = \begin{cases} (P_L, P_R) & \text{where } P_L = \{p \in P : p_d \leq m\} \\ & \text{and } P_R = \{p \in P : p_d > m\} \end{cases} \tag{6.24}$$

where $d = \text{depth} \bmod 3$ selects coordinate axis and m is median value.

Kd-tree query complexity is $O(\log n + k)$ for k results in balanced trees. For heritage point clouds with $n \approx 10^8$ points, query time scales logarithmically rather than linearly.

Spatial Hashing Grid-based spatial hashing provides $O(1)$ average-case lookup for uniform distributions [100]. Hash function maps 3D coordinates to bucket indices:

$$h(x, y, z) = (p_1 x + p_2 y + p_3 z) \bmod B \tag{6.25}$$

where p_1, p_2, p_3 are large primes and B is bucket count.

Grid cell size δ balances query efficiency with memory usage:

- Small δ: Few points per cell, many empty cells
- Large δ: Many points per cell, linear search within cells
- Optimal δ: $\approx \sqrt[3]{V/n}$ for volume V and n points

R-Tree Indexing R-trees index rectangular bounding boxes using hierarchical clustering [43]. Each internal node stores minimum bounding rectangle (MBR) containing all child MBRs:

$$\text{MBR}(N) = \bigcup_{c \in \text{children}(N)} \text{MBR}(c) \tag{6.26}$$

R-tree insertion chooses subtree minimizing MBR enlargement:

$$\text{choose}(R) = \arg\min_c |\text{MBR}(c) \cup R| - |\text{MBR}(c)| \tag{6.27}$$

For heritage segments with varying sizes, R-trees provide better performance than uniform grid structures.

6.4.3 Hierarchical Storage

Hierarchical storage organizes heritage data according to spatial and semantic relationships [22, 85]. Multi-level organization enables progressive access patterns matching user interaction behaviors.

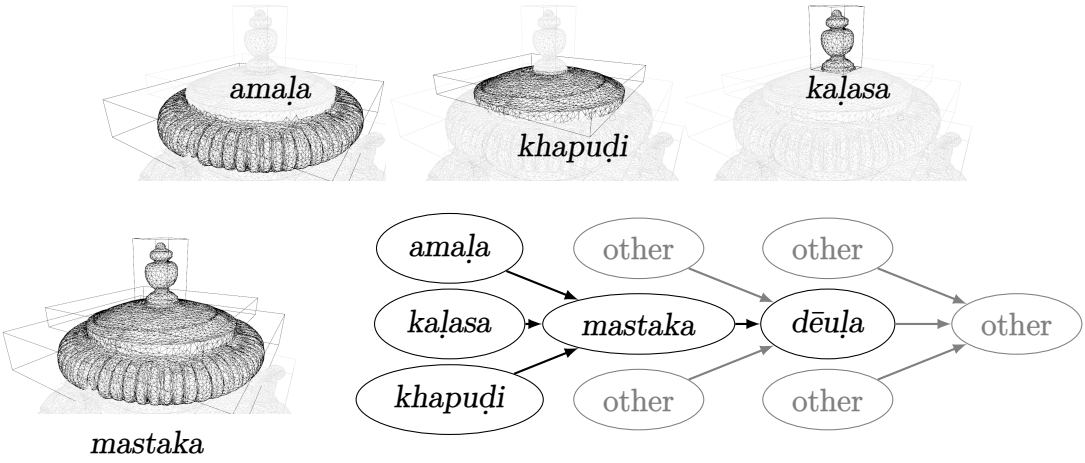

Fig. 6.4: Hierarchical geometric storage for the Rajarani Temple mastaka. Top row shows three segmented components (amala, khapudi, kalasa). Bottom diagram illustrates the kd-tree organization where segments are hierarchically organized based on geometric centroids, enabling efficient spatial queries without requiring semantic classification.

Geometric Hierarchies Spatial hierarchies organize segments by containment relationships. For monument M with segments $S = \{s_1, s_2, \ldots, s_k\}$, hierarchy construction uses geometric centers:

$$c_i = \frac{1}{|s_i|} \sum_{p \in s_i} p \tag{6.28}$$

Kd-tree construction on centers $\{c_i\}$ creates spatial hierarchy independent of semantic classification [9]. This approach prevents subjective classification errors common in heritage contexts [42].

Fig. 6.4 illustrates hierarchical organization for the Rajarani Temple mastaka. The three principal segments (amala, khapudi, kalasa) are organized in a kd-tree structure based on geometric centroids, creating spatial relationships without imposing semantic interpretation.

Level-of-Detail Storage Multi-resolution storage supports progressive loading based on viewing distance and available bandwidth [46]. For each segment s, multiple detail levels are pre-computed:

- LOD 0: Full resolution (original data)
- LOD 1: 50% decimation (close viewing)
- LOD 2: 10% decimation (medium distance)
- LOD 3: 1% decimation (overview)

Selection criterion based on projected screen size:

$$\text{LOD} = \begin{cases} 0 & \text{if } s_{screen} > \tau_0 \\ 1 & \text{if } \tau_1 < s_{screen} \leq \tau_0 \\ 2 & \text{if } \tau_2 < s_{screen} \leq \tau_1 \\ 3 & \text{if } s_{screen} \leq \tau_2 \end{cases} \tag{6.29}$$

where s_{screen} is projected segment size in pixels.

Progressive decimation strategies enable adaptive streaming where coarse base models load first, followed by incremental refinement as bandwidth permits. The decimation sequence $M_0 \rightarrow M_1 \rightarrow \cdots \rightarrow M_k$ for mesh M proceeds through successive edge contractions, with each level M_i storing only the differential operations needed to reconstruct M_{i-1}. For heritage applications, decimation order prioritizes preserving culturally significant features: edges adjacent to high-curvature regions (sculptural details) are contracted last, while edges in low-curvature regions (planar walls) decimate first. This approach ensures that initial coarse models maintain recognizable architectural character, with fine ornamental details streaming progressively as viewing requirements demand higher fidelity. Bandwidth-adaptive algorithms monitor network conditions and adjust LOD transitions dynamically: slow connections maintain lower detail levels while faster connections pre-fetch higher resolutions speculatively.

Semantic Hierarchies Architectural hierarchies follow cultural classification systems [10]. For Kalinga architecture, natural organization follows structural elements:

$$\text{Temple} \rightarrow \{\text{Deula}, \text{Jagamohana}\} \tag{6.30}$$

$$\text{Deula} \rightarrow \{\text{Pitha}, \text{Bada}, \text{Gandi}, \text{Mastaka}\} \tag{6.31}$$

$$\text{Mastaka} \rightarrow \{\text{Kalasa}, \text{Khapudi}, \text{Amala}, \text{Beki}\} \tag{6.32}$$

Database queries leverage hierarchical relationships for component-wise analysis, enabling operations such as finding all carvings in the temple complex, comparing architectural elements across monuments, tracking conservation status by building component, and generating architectural documentation by hierarchy level.

Query optimization exploits hierarchy for pruning:

$$\text{Query}(\text{component}) = \bigcup_{c \in \text{subtree}(\text{component})} \text{segments}(c) \tag{6.33}$$

6.4.4 Bounding Box Isolation

Spatial isolation algorithms extract geometric subsets from large heritage models without loading complete datasets [9]. Bounding box operations enable selective transmission and progressive loading by filtering point clouds and meshes to regions of interest.

Point Cloud Isolation For point cloud $P = \{p \in \mathbb{R}^3\}$, axis-aligned bounding box isolation selects points within coordinate limits. Given bounds x_l, x_r (left, right), y_b, y_t (bottom, top), and z_f, z_b (front, back), the isolated point cloud P' is:

$$P' = \text{Bbox}(P; x_l, x_r, y_b, y_t, z_f, z_b) \tag{6.34}$$

where

$$\text{Bbox}(P; x_l, x_r, y_b, y_t, z_f, z_b) = \{p \in P : x_l \leq p_x \leq x_r \land y_b \leq p_y \leq y_t \land z_f \leq p_z \leq z_b\} \tag{6.35}$$

Point cloud isolation requires $O(n)$ time for n points through single linear scan. Spatial indexing structures (kd-trees, octrees) reduce query time to $O(\log n + k)$ for k results by pruning regions outside the bounding box during tree traversal.

Mesh Isolation Mesh isolation preserves topological consistency by including faces with any vertex inside the bounding box. For mesh $M(V, F)$ with vertex set V and face set F, isolation produces $M'(V', F')$:

$$M' = \text{Bbox}(M(V, F); x_l, x_r, y_b, y_t, z_f, z_b) = M'(V', F') \tag{6.36}$$

where the isolated face set includes faces with at least one vertex in bounds:

$$\begin{aligned} F' = \{(p, q, r) \in F : &(x_l \leq p_x \leq x_r \land y_b \leq p_y \leq y_t \land z_f \leq p_z \leq z_b) \\ &\lor (x_l \leq q_x \leq x_r \land y_b \leq q_y \leq y_t \land z_f \leq q_z \leq z_b) \\ &\lor (x_l \leq r_x \leq x_r \land y_b \leq r_y \leq y_t \land z_f \leq r_z \leq z_b)\} \end{aligned} \tag{6.37}$$

and the vertex set contains only vertices referenced by isolated faces:

$$V' = \{v \in V : \exists f \in F', v \in f\} \tag{6.38}$$

This conservative approach prevents mesh tearing by maintaining face integrity across bounding box boundaries. Faces partially intersecting the box boundary remain complete, producing slightly larger isolated meshes than strict containment but preserving manifold topology essential for rendering and analysis.

For the Rajarani Temple, bounding box isolation enables detailed examination of architectural elements (individual carvings, columns) without loading the full 35 GiB dataset. A typical isolation query for a 2m × 2m × 2m region extracts 10-50 MiB subsets, reducing memory requirements by three orders of magnitude and enabling interactive exploration on consumer hardware.

Fig. 6.5 demonstrates bounding box isolation applied to the Rajarani mesh. The left image shows the full decimated mesh, while the right shows an isolated vertical section extracted using bounding box coordinates. The yellow wireframe indicates the bounding box boundaries.

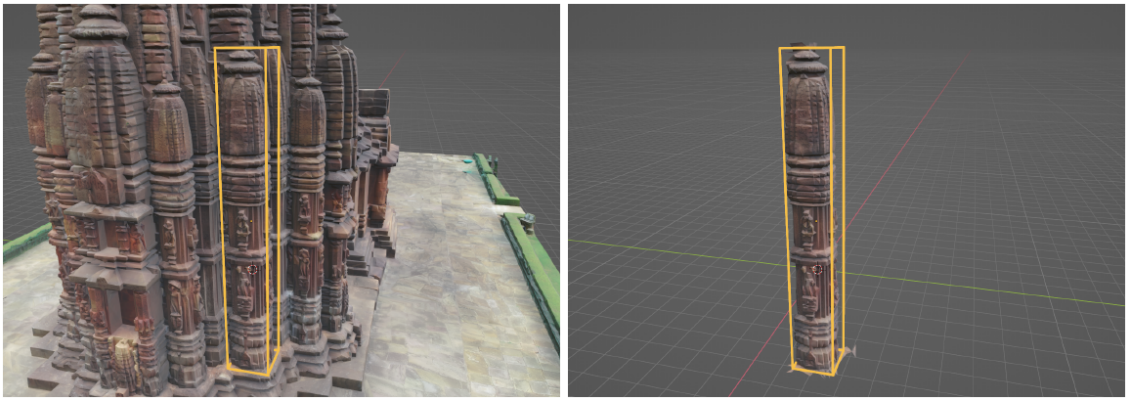

Fig. 6.5: Bounding box isolation applied to the Rajarani Temple mesh. Left: complete low-resolution mesh model. Right: isolated mesh segment extracted using axis-aligned bounding box (yellow wireframe). The isolated mesh contains only faces with at least one vertex inside the bounding box, maintaining topological integrity while reducing data size.

6.5 Case Study: Rajarani Temple

The Rajarani Temple documentation demonstrates practical application of HBIM techniques to a complex heritage site [9]. This 11th-century temple in Bhubaneswar exhibits typical Kalinga architecture features requiring specialized processing approaches.

Fig. 6.6 shows the Rajarani Temple and its current compound. The temple consists of two primary structures: the deula (sanctuary) and jagamohana (porch), both exhibiting characteristic Kalinga architectural elements with extensive surface carvings and sculptural details.

6.5.1 Data Collection

Data acquisition combined terrestrial laser scanning and aerial photogrammetry to capture the temple's geometric complexity and architectural details [5].

Scanning Campaign TLS data collection used systematic positioning to minimize occlusion and registration error:

- Scanner: FARO M70 with 3mm accuracy at 30m range
- Scan positions: 62 locations covering all temple surfaces
- Point density: Variable from 1mm (carvings) to 10mm (walls)
- Total points: 782,209,741 with RGB color information
- File size: 35 GiB in XYZ format

Registration accuracy achieved sub-millimeter precision using spherical targets and cloud-to-cloud alignment [13].

Fig. 6.6: The Rajarani Temple in Bhubaneswar, India. Left: ground-level view showing the deula (sanctuary tower) and jagamohana (porch) with intricate stone carvings. Right: aerial view showing the temple compound layout and spatial relationships between structures. The temple spans 23.7m × 12.2m with the deula rising to 18.5m and jagamohana to 12.1m.

Photogrammetric Survey UAV photogrammetry captured high-resolution imagery for texture mapping and verification:

- Flight altitude: 80m above ground level
- Ground speed: 7 m/s with automated flight planning
- Image overlap: 75% forward and side overlap
- Total images: 2,334 photographs at 20 megapixel resolution
- Coverage area: 23.7m × 12.2m temple footprint

Structure-from-Motion processing generated 3D mesh with photorealistic textures for visual documentation and public engagement [101].

Fig. 6.7 shows the automated UAV flight path used for photogrammetric data collection. The systematic grid pattern with 75% overlap ensures complete coverage and sufficient redundancy for accurate 3D reconstruction.

The photogrammetric reconstruction pipeline processed the 2,334 UAV photographs to generate a high-fidelity textured mesh of the temple complex. **Fig. 6.8** shows rendered views of the resulting 3D model from multiple perspectives.

6.5.2 Processing Pipeline

Data processing followed systematic workflow optimizing for heritage-specific requirements while managing computational constraints [9].

Point Cloud Optimization Raw point cloud required substantial processing before analysis:

(a) Registration: Align 62 scan positions using ICP refinement
(b) Filtering: Remove statistical outliers and scanner artifacts
(c) Sampling: Apply inverse density sampling for uniform distribution

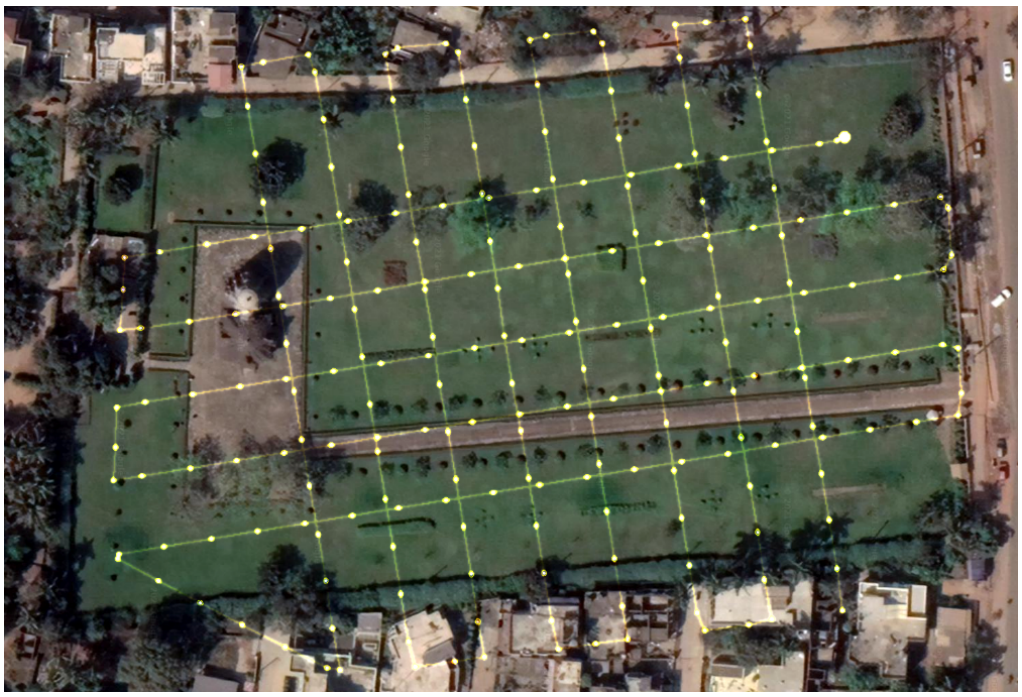

Fig. 6.7: UAV flight path for photogrammetric data acquisition. The yellow dots indicate photograph capture positions, with the larger dot marking the starting location. The flight pattern follows a systematic grid at 80m altitude with 75% forward and side overlap, ensuring complete coverage of the temple complex. Base imagery courtesy of Google Maps.

(d) Segmentation: Partition temple into architectural components
(e) Storage: Convert to compressed LAZ format for archival

Inverse density sampling reduced point count from 7.8×10^8 to 7.8×10^7 while preserving detailed carvings. Processing times varied significantly: farthest point sampling required 20 minutes, inverse density sampling 3 minutes, Poisson disk sampling 30 seconds, and random sampling 5 seconds (though with poor quality).

Fig. 6.9 compares processing time across different point cloud sizes for the three sampling strategies. Farthest point sampling exhibits $O(n^2)$ growth, Poisson disk sampling maintains near-constant time through spatial hashing, while inverse density sampling shows moderate $O(n \log n)$ complexity.

Mesh Processing Photogrammetric mesh required decimation for practical visualization. The original mesh contained 10^7 vertices requiring 3 GiB storage. Edge contraction decimation provided best geometric preservation, vertex decimation offered fastest processing with acceptable quality loss, while appearance-preserving simplification achieved best visual quality at highest computation cost.

Chamfer distance analysis showed minimal quality degradation until 1:100 decimation ratio, enabling substantial size reduction while preserving architectural features [9].

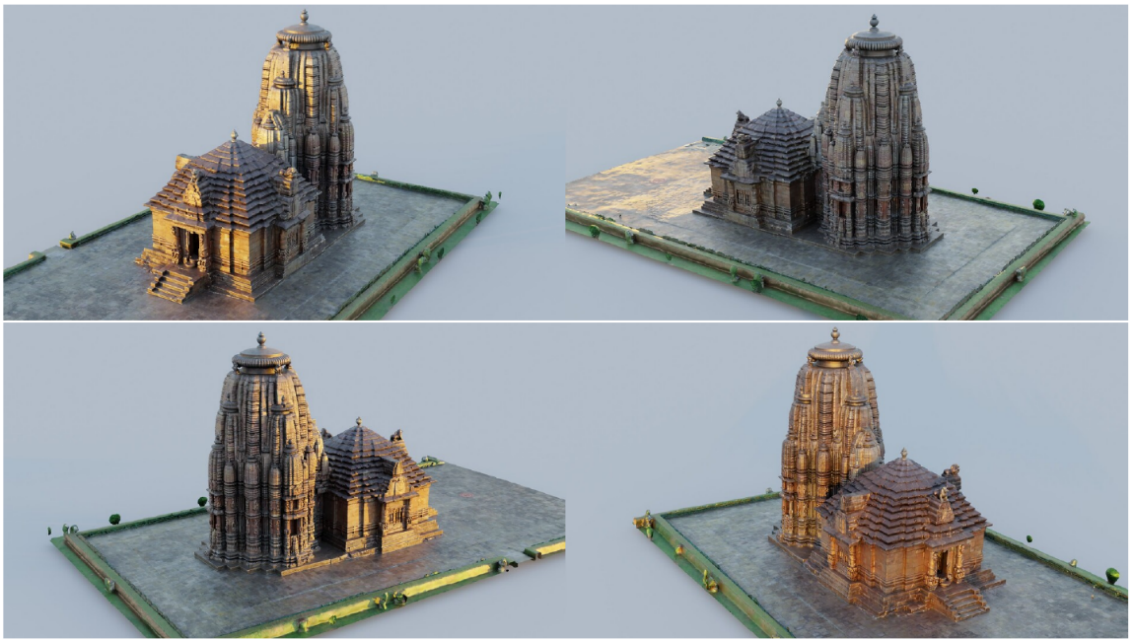

Fig. 6.8: Rendered photogrammetric model of the Rajarani Temple from four viewing angles. The model combines geometric reconstruction from Structure-from-Motion with photorealistic textures from the UAV imagery, capturing both the architectural structure and surface details including carvings and weathering patterns. Rendering performed using Cycles path-tracing engine.

Visual inspection across progressive decimation levels reveals that geometric fidelity remains perceptually intact through 1:10 reduction, with architectural details like column striations and relief carvings clearly visible. At 1:100 decimation, subtle surface textures begin to smooth, though major structural elements and proportions are preserved. Only at 1:1000 decimation does significant visual degradation occur, with fine sculptural details merging into simplified surfaces. This degradation pattern validates density-aware metrics: heritage-critical regions with intricate geometry resist decimation longer than planar surfaces, maintaining cultural authenticity across compression ratios suitable for network transmission and consumer hardware visualization.

Fig. 6.10 demonstrates progressive decimation on the Rajarani Temple deula. The leftmost image shows the original mesh with 10^7 vertices, followed by successive 10:1 reductions. Visual quality remains high through two decimation levels, with noticeable but acceptable degradation at the third level, and significant simplification only at extreme compression ratios.

Fig. 6.11 illustrates the progressive decimation process on a simplified mesh, showing how successive edge contractions reduce vertex count while attempting to preserve overall shape.

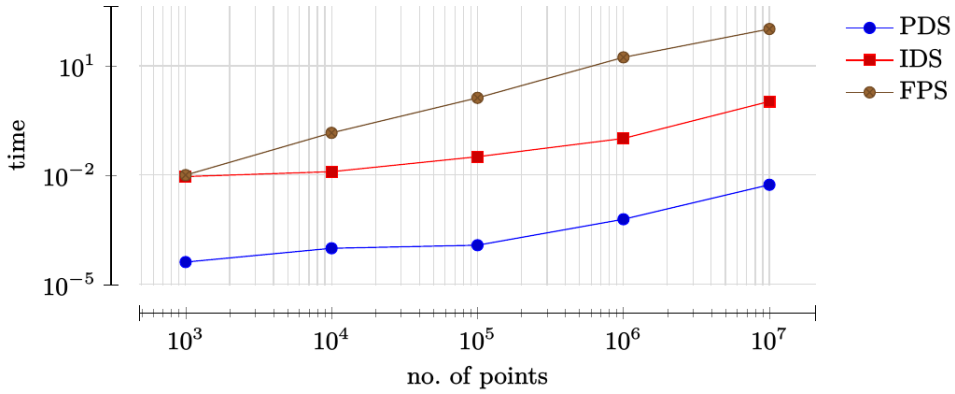

Fig. 6.9: Sampling strategy performance comparison. Processing time (vertical axis, log scale) versus point cloud size (horizontal axis, log scale) for farthest point sampling (FPS), inverse density sampling (IDS), and Poisson disk sampling (PDS). FPS exhibits quadratic complexity, IDS shows log-linear growth, while PDS maintains sub-second performance across scales. Data points represent averages over multiple runs.

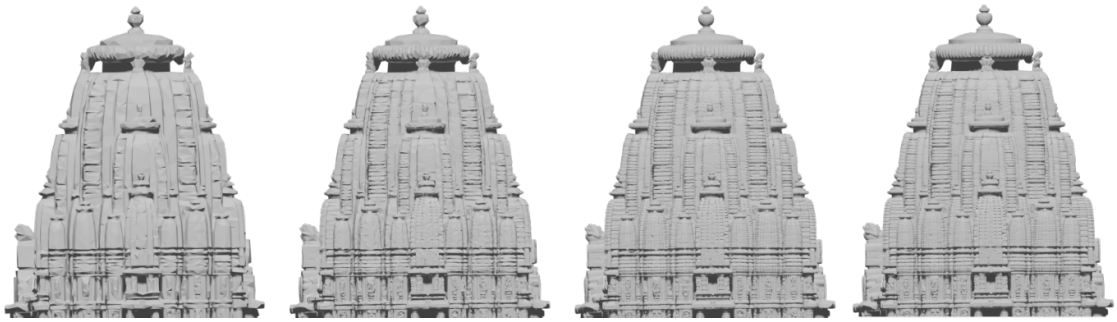

Fig. 6.10: Progressive mesh decimation applied to the Rajarani Temple deula. From left to right: original mesh (10^7 vertices), 10% decimation (10^6 vertices), 1% decimation (10^5 vertices), 0.1% decimation (10^4 vertices). Architectural details remain recognizable through 1% decimation. The rightmost panel represents the *failed state*: at 10^4 vertices fine sculptural carvings merge into simplified surfaces, cultural detail is irrecoverably lost, and the density-aware Chamfer distance exceeds acceptable thresholds.

Fig. 6.11: Progressive decimation on a simplified mesh geometry. Each step removes 50% of vertices through edge contraction. The sequence demonstrates how mesh topology simplifies while maintaining approximate surface shape.

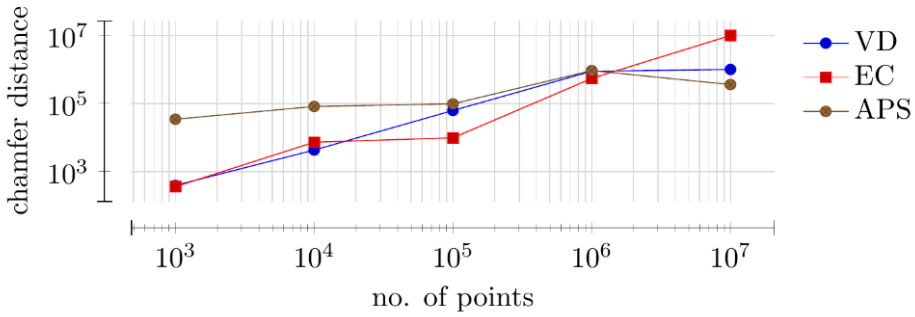

Fig. 6.12: Decimation quality comparison using density-aware Chamfer distance (vertical axis, log scale) versus decimation ratio (horizontal axis, log scale). Vertex decimation (VD), edge contraction (EC), and appearance-preserving simplification (APS) maintain similar quality through moderate compression. Edge contraction provides best geometric preservation at high compression ratios.

6.5.3 Performance Evaluation

System performance evaluation covered storage efficiency, transmission bandwidth, and visualization capabilities across different hardware configurations.

Storage Analysis Compression performance varied significantly by data type and algorithm:

- XYZ format: 35 GiB (baseline, uncompressed text)
- PLY binary: 12 GiB (3:1 compression over XYZ)
- LAZ compressed: 5 GiB (7:1 compression with lossless encoding)
- Decimated mesh: 500 MiB at 1:10 vertex reduction

Database storage separated metadata (2 MiB) from binary files, enabling efficient queries without loading large geometries.

Mesh decimation provides substantial storage reduction with acceptable quality loss. **Fig. 6.12** compares density-aware Chamfer distance for three decimation strategies across compression ratios. All strategies maintain low error through moderate decimation, with divergence only at extreme compression.

Storage requirements scale linearly with vertex count but vary by file format. **Fig. 6.13** shows storage size versus decimation fraction for different mesh formats and decimation strategies.

Network Transmission Progressive transmission reduced bandwidth requirements by delivering detail on demand:

- Full model: 35 GiB, 3+ hours on broadband
- Decimated overview: 500 MiB, 3 minutes on broadband
- Segmented components: 50-200 MiB per architectural element

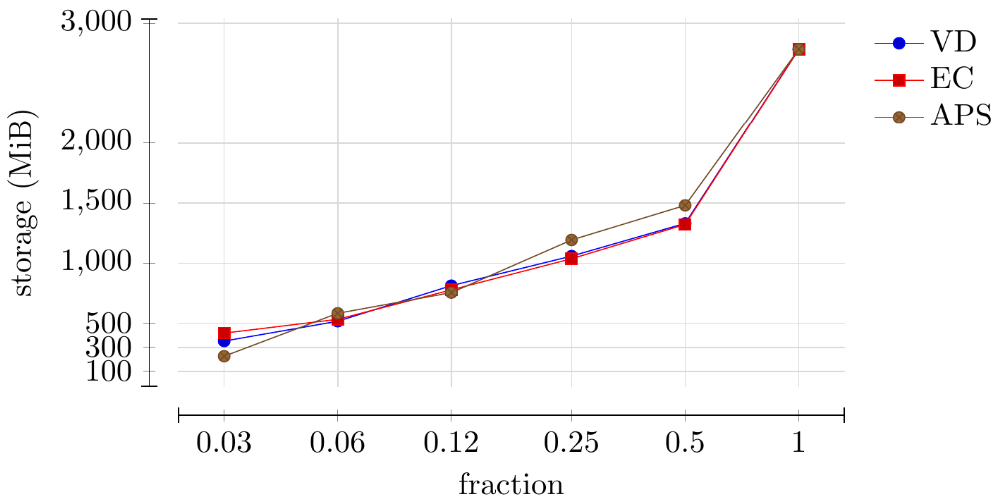

Fig. 6.13: Storage requirements (vertical axis, linear scale in MiB) versus decimation fraction (horizontal axis, log scale) for vertex decimation (VD), edge contraction (EC), and appearance-preserving simplification (APS). All strategies show linear storage scaling with vertex count. The original 3 GiB mesh reduces to 500 MiB at 10% decimation and 100 MiB at 3% decimation.

- Progressive refinement: Load coarse model, stream details

User studies showed 95% of viewing sessions required less than 1 GiB data transfer using progressive loading strategies.

Detailed sampling strategy performance for point clouds with varying sizes demonstrates computational trade-offs. **Table. 6.1** summarizes processing times across different point cloud densities. Farthest point sampling (FPS) achieves optimal spatial distribution but requires $O(n^2)$ complexity, resulting in 20 minutes for 10^7 points. Poisson disk sampling (PDS) enforces minimum distance constraints with sub-second performance through spatial hashing, but produces less uniform coverage. Inverse density sampling (IDS) balances quality and efficiency, requiring 3 minutes for 10^7 points while preserving heritage-critical fine details. For the Rajarani dataset at 7.8×10^8 points, only PDS and IDS remained computationally feasible on available hardware.

6.5.4 Web-based Visualization

Web deployment enables broad access to heritage models while managing bandwidth and computational constraints through progressive streaming and adaptive quality [63].

Streaming Architecture Client-server architecture delivers content based on viewing requirements:

(a) Initial load: Low-resolution overview model (10 MiB)
(b) Progressive refinement: Stream higher detail on zoom

Table. 6.1: Point cloud sampling strategy performance comparison on AMD Threadripper 3970X. Processing times represent averages over multiple runs. Random sampling provides baseline for minimal computational cost.

Strategy	10^3 **points**	10^5 **points**	10^7 **points**	**Complexity**
Random	<1s	<1s	5s	$O(n)$
Poisson Disk	<1s	<1s	30s	$O(n)$
Inverse Density	<1s	10s	3min	$O(n \log n)$
Farthest Point	1s	30s	20min	$O(n^2)$

(c) Spatial culling: Load only visible components
(d) Adaptive quality: Adjust resolution based on connection speed

Bounding box queries enable efficient spatial filtering:

$$\text{Visible} = \{s \in \text{segments} : \text{intersects}(s.bbox, \text{view\_frustum})\} \tag{6.39}$$

Performance Optimization Web optimization techniques reduce loading time and improve responsiveness:

- Mesh compression: Draco compression for web delivery
- Texture optimization: Mipmapping and progressive JPEG
- Spatial indexing: Octree organization for frustum culling
- Caching: Browser caching of frequently accessed components

Performance metrics on consumer hardware (Intel i5, 8GB RAM, integrated graphics):

- Initial page load: 5-10 seconds
- Progressive detail loading: 1-3 seconds per refinement
- Interactive frame rate: 30+ FPS for decimated models
- Memory usage: 2-4 GiB peak during detailed viewing

The Rajarani Temple case study demonstrates that heritage HBIM systems can achieve practical performance through careful optimization of data structures, compression algorithms, and progressive delivery mechanisms.

6.5.5 Framework Generalizability

Although the Rajarani Temple served as the primary case study, the framework's core components are geometry-agnostic and transfer to heritage sites with substantially different architectural styles, construction materials, and degradation states.

Inverse density importance sampling and kd-tree hierarchical storage make no assumptions about. Most general purpose scans produce point clouds with pronounced density

variation, where the inverse density sampler outperforms uniform or farthest-point alternatives. The bounding-box isolation algorithm operates on axis-aligned coordinate bounds, requiring only a re-parameterization of the bounding boxes to match the coordinate frame of each site.

The density-aware Chamfer metric normalizes by local density $\hat{n}_q$, so it adapts proportionally regardless of whether the scans produce dense or sparse point clouds. For low-reflectivity materials, longer-range scanner positions or multiple-return sensors may be required to achieve adequate point density; the processing pipeline is otherwise unchanged.

The bounding-box isolation algorithm operates in $O(\log n)$ with kd-tree indexing regardless of monument scale. The full pipeline has been validated at 35 GiB (Rajarani); for smaller objects the same tools apply with reduced computational cost, while larger complexes (temple cities, fortified ensembles) require more levels of partitioning.

Current Limitations of Generalization Two aspects of the framework remain site-specific. First, semantic hierarchies (Section 6.4.3) are manually defined per architectural tradition; transferring from Kalinga to other architecture requires manual update of the containment graph. Second, the framework does not integrate complementary non-optical sensors (ground-penetrating radar, infrared thermography) needed for heritage with significant subsurface damage or organic growth (moss, lichen). Extending the database schema and hierarchical indexing to heterogeneous sensor modalities is a natural direction for future work.

6.6 Summary

This chapter presented techniques for digital heritage preservation addressing the storage, transmission, and exhibition challenges identified in Chapter 1. The Rajarani Temple case study demonstrated that massive heritage datasets (782 million points, 35 GiB) can be compressed, organized, and delivered for interactive visualization on consumer hardware.

The novel contributions are: inverse density importance sampling that preserves fine architectural carvings while reducing point counts by 90%; density-aware Chamfer distance metrics for heritage-specific quality assessment; hierarchical database organization using geometric kd-trees that avoids subjective semantic classification; and bounding box isolation algorithms enabling progressive web delivery.

Unlike commercial HBIM systems designed for parametric modeling of new construction, our approach preserves irregular heritage geometry without imposing interpretive frameworks on archaeological data. The resulting system achieves practical performance: 5-10 second initial loads, 30+ FPS interactive viewing, and sub-gigabyte transmission for typical sessions.

These techniques complement the neural representations of Chapters 4–5. While DDFs offer efficient rendering through learned distance fields, heritage applications require explicit point cloud and mesh representations for archival integrity, interoperability with

existing tools, and long-term preservation independent of neural network availability. The compression and progressive delivery methods presented here enable practical deployment of heritage digital twins for research, education, and public engagement.

7 Conclusions and Future Work

This thesis addressed three problems at the intersection of machine learning, computer graphics, and digital heritage: aggregation in neural networks, differentiable rendering of implicit surfaces, and large-scale heritage data management. The contributions span theoretical foundations (k-restricted functions), algorithmic innovations (neural DDFs), and practical systems (heritage digitization).

7.1 Summary of Contributions

7.1.1 k-Restricted Aggregation Functions

Chapter 2 introduced k-restricted overlap and grouping functions that generalize traditional aggregation operators. For n-dimensional input $x \in [0,1]^n$ and restriction parameter $k \in \{1, \ldots, n\}$, k-restricted grouping functions satisfy $T^n(x) = 1$ only when at least k inputs equal 1, contrasting with standard grouping functions where $k = 1$ [7].

Existence theorems established that k-restricted functions exist for any k with appropriate dimensional constraints. For odd n satisfying $n/2 + 0.5 = k$ and extended overlap function O^*, the construction

$$F^n(x) = \frac{1}{\binom{n}{k}} \sum_{\{i_1,\ldots,i_k\} \subset [n]} O^k(x_{i_1}, \ldots, x_{i_k}) \tag{7.1}$$

yields an n-dimensional k-restricted overlap function. Representation theorems characterize these functions through function pairs (f^n, g^n) satisfying specific monotonicity and boundary conditions.

Chapter 3 demonstrated k-restricted functions as CNN pooling replacements. Experimental evaluation across 10 datasets (MNIST, Fashion-MNIST, CIFAR-10/100, CALTECH-101/256, STL-10, Flowers102, TinyImageNet) and 6 architectures (LeNet-5, VGG-16, ResNet-18/34/50) showed consistent improvements. The function $T^{\min}$ achieved 0.839 accuracy on CIFAR-10 with LeNet-5 compared to 0.835 for average pooling. VGG-16 with $T^{\min}$ reached 0.996 on MNIST versus 0.939 for average pooling. GPU implementations demonstrated 45× speedup over CPU with 0.010–1.483 ms per 1000 computations [7].

The distributed gradient flow property $\partial T / \partial x_i \neq 0$ for multiple inputs contrasts with max pooling's sparse gradients, improving training dynamics in deep networks.

7.1.2 Neural Directional Distance Fields

Chapter 5 presented neural DDFs for efficient path-traced rendering. The DDF $\phi : B \times \mathbb{R}^2 \to \mathbb{R}^+$ returns distance from point $x \in B$ to surface S along direction θ. The directed eikonal equation

$$\nabla_x \phi(x, \theta) \cdot \theta = -1 \tag{7.2}$$

enables constant-time ray intersection queries in theory. Surface normals compute as $n = \kappa \nabla_x \phi(x, \theta) / \|\nabla_x \phi(x, \theta)\|_2$ via automatic differentiation [8].

An 8-layer MLP with 512 hidden units and 1.3M parameters successfully represents DDFs for complex geometry. Surface-based sampling generates training data by marching rays from mesh surfaces. The marching sample count s_p dominates reconstruction quality more than direction samples s_{dr} or face samples s_{fc}.

Experimental results on ShapeNet objects (airplane, car, chair, couch, table) and test meshes (bunny, Suzanne) show chamfer distances 0.5–0.7 $\times 10^{-3}$, comparable to DeepSDF and DeepDDF baselines. Rendering achieves 10–100$\times$ speedup over BVH-based methods after GPU caching [8].

7.1.3 Heritage Digitization Methodology

Chapter 6 addressed storage, transmission, and exhibition challenges for massive heritage datasets. The Rajarani Temple case study contains 782 million points requiring 35 GiB storage. Four novel contributions address practical deployment [9]:

Inverse Density Sampling For point p_i with local density ρ_i, sampling probability

$$P(p_i) = \frac{1/\rho_i}{\sum_{j=1}^{n} 1/\rho_j} \tag{7.3}$$

preserves fine architectural carvings while reducing redundancy in uniform regions. This achieves 90% point reduction with sub-centimeter accuracy in high-detail areas.

Density-Aware Chamfer Distance The bidirectional metric

$$d_{CD}(P, Q) = \frac{1}{|P|} \sum_{p \in P} \min_{q \in Q} \|p - q\|^2 + \frac{1}{|Q|} \sum_{q \in Q} \min_{p \in P} \|q - p\|^2 \tag{7.4}$$

assesses compression quality for heritage applications. Mesh decimation at 1:10 ratio maintains chamfer distance below 5mm.

Hierarchical Geometric Organization Kd-tree construction on segment centroids $c_i = (1/|s_i|) \sum_{p \in s_i} p$ provides spatial hierarchy without subjective semantic classification. Query complexity $O(\log n + k)$ for k results enables efficient spatial queries.

Progressive Web Delivery Bounding box isolation

$$\text{Visible} = \{s : \text{intersects}(s.bbox, \text{view\_frustum})\} \tag{7.5}$$

enables streaming based on camera frustum. Initial 10 MiB overview loads in 5–10 seconds, with 30+ FPS interactive viewing on consumer hardware (Intel i5, 8GB RAM, integrated graphics) [9].

7.2 Future Research Directions

7.2.1 Neural Implicit Representations

Hierarchical DDFs Current DDFs use fixed-capacity networks regardless of geometric complexity. Hierarchical representations could allocate capacity based on local detail. A spatial octree with per-node neural networks $\{\phi_i\}_{i=1}^{N}$ enables adaptive refinement:

$$\phi(x, \theta) = \phi_{\text{node}(x)}(x_{\text{local}}, \theta) \tag{7.6}$$

where $\text{node}(x)$ selects the octree cell containing x and x_{local} represents coordinates relative to cell center. Leaf nodes covering uniform regions require small networks while high-detail areas use larger capacity. Balancing tree depth, node capacity, and total parameter count requires optimization.

Differentiable DDF Rendering Current DDF rendering lacks full differentiability with respect to scene parameters. Making the renderer differentiable would enable inverse graphics applications: given observed images I_{obs}, optimize shape parameters θ minimizing

$$\mathcal{L} = \|I_{\text{obs}} - \text{Render}(\phi_\theta)\|^2 \tag{7.7}$$

The render function requires differentiable ray marching and shading. The eikonal equation provides continuous gradients for intersection points, but visibility discontinuities at silhouettes require edge sampling or reparameterization techniques from Chapter 4. Combining DDFs with warped-area sampling [4] could provide unbiased gradients for shape optimization.

7.2.2 Aggregation Functions

Learnable Restriction Parameters Fixed restriction parameters k may be suboptimal for heterogeneous data. Making k learnable would adapt aggregation behavior per layer or spatial location. For input $x \in \mathbb{R}^{C \times H \times W}$ at layer l, learnable restriction $k_{l,c,i,j}$ for channel c and position (i, j) adapts to local feature distributions. The parameter

space $k \in \{1, \ldots, n\}$ is discrete, requiring gradient estimators like Gumbel-softmax:

$$P(k = i) = \frac{\exp((\log \alpha_i + g_i)/\tau)}{\sum_{j=1}^{n} \exp((\log \alpha_j + g_j)/\tau)} \tag{7.8}$$

where α are learnable weights, g_i are Gumbel noise samples, and τ controls softmax temperature. This enables differentiable k selection during training.

Attention-Based k-Restricted Pooling Combining k-restricted functions with attention mechanisms could improve adaptive aggregation. For input window $x \in \mathbb{R}^n$, attention weights $\alpha = \text{softmax}(W_q x)$ determine element importance. The attended k-restricted function

$$T_\alpha^k(x) = T^k(\alpha \odot x) \tag{7.9}$$

applies element-wise weighting before aggregation. This enables the network to focus on relevant features while maintaining k-restricted boundary behavior.

7.2.3 Heritage Documentation

Neural Heritage Representations Combining neural implicit representations with heritage digitization could reduce storage requirements. A neural radiance field $(\sigma, c) = f_\theta(x, d)$ trained on the 782 million Rajarani points requires only network parameters ($\sim$1–10 MB) versus 35 GiB for raw data. However, archival stability concerns remain: neural representations depend on framework availability over century-scale preservation timelines.

A hybrid approach stores both point clouds (for archival integrity) and neural networks (for interactive viewing). The neural component provides real-time visualization while the point cloud serves as preservation master. Validation metrics comparing neural reconstruction to ground truth ensure cultural detail preservation.

Temporal Change Detection Heritage monuments deteriorate over time through weathering, structural damage, and environmental factors. Multi-epoch scanning enables change detection. For scans at times t_1, t_2, computing point cloud difference $\Delta P = P(t_2) - P(t_1)$ identifies surface changes. However, registration errors between epochs introduce false positives. Robust change detection requires:

$$\text{Change}(x) = \begin{cases} 1 & \text{if } d(x, P(t_2)) > \tau \\ 0 & \text{otherwise} \end{cases} \tag{7.10}$$

where threshold τ exceeds registration error. Quantifying change rates enables predictive maintenance and conservation prioritization.

7.3 Concluding Remarks

This thesis contributions span theory (k-restricted functions with existence and representation theorems), algorithms (neural DDFs for efficient rendering), and applications (heritage digitization methodology). The work demonstrates that mathematical foundations enable practical systems: k-restricted function theory leads to improved CNN pooling; eikonal equation properties enable fast ray tracing; geometric hierarchies support scalable heritage visualization.

The limitations and future directions highlight open problems requiring continued research. Theoretical understanding of k-restricted function expressiveness, real-time neural rendering of dynamic scenes, and automated heritage processing remain active challenges. The interdisciplinary nature of this work, connecting fuzzy logic, differential geometry, computer graphics, and archaeology, demonstrates the value of mathematical rigor applied to practical problems.

The three contributions address different aspects of 3D data processing: feature aggregation in neural networks (pooling), efficient representation for rendering (DDFs), and practical deployment for heritage preservation (digitization). While developed independently, these techniques share common principles: mathematical formalization of intuitive concepts, neural network parameterization of complex functions, and experimental validation across diverse scenarios. This methodology, formal foundations followed by neural implementation and empirical evaluation, provides a template for future research at the intersection of mathematics and machine learning.

Bibliography

[1] Veronika Abramova, Jorge Bernardino, and Pedro Furtado. Sql or nosql? performance and scalability evaluation. *International Journal of Business Process Integration and Management*, 7(4):314–321, 2015.

[2] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics*, 9(1):3–15, 2003.

[3] John Amanatides, Andrew Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10, 1987.

[4] Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. Unbiased warped-area sampling for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(6):1–18, 2020.

[5] Bettina Baumer. The rajarani temple re-indentified. *India International Centre Quarterly*, 21(1):125–132, 1994.

[6] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153):1–43, 2018.

[7] Annada Prasad Behera, Swati Rani Hait, Bapi Dutta, and Subhankar Mishra. Restriction based overlap and grouping functions with its application in convolutional neural network architecture. *Information Sciences*, 2025.

[8] Annada Prasad Behera and Subhankar Mishra. Neural directional distance field object representation for uni-directional path-traced rendering. In *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–6, 2023.

[9] Annada Prasad Behera and Subhankar Mishra. Digitizing temples for heritage conservation: Kalinga architecture. *Digital Applications in Archaeology and Cultural Heritage*, 39:e00462, 2025.

[10] Kanuana Sagar Behera. *The Lingaraja Temple of Bhubaneshwara: Art and Cultural Legacy*. Aryan Books International, 2008.

[11] Gleb Beliakov, Ana Pradera, Tomasa Calvo, et al. *Aggregation functions: A guide for practitioners*, volume 221. Springer, 2007.

[12] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[13] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. Spie, 1992.

[14] Qi Bi, Kun Qin, Han Zhang, Jiafen Xie, Zhili Li, and Kai Xu. Apdc-net: Attention pooling-based convolutional network for aerial scene classification. *IEEE Geoscience and Remote Sensing Letters*, 17(9):1603–1607, 2019.

[15] Michael Blaha. Referential integrity is important for databases. *Modelsoft Consulting Corp*, 2005.

[16] Wolfgang Boehler, M Bordas Vicent, Andreas Marbs, et al. Investigating laser scanner accuracy. *The international archives of photogrammetry, remote sensing and spatial information sciences*, 34(Part 5):696–701, 2003.

[17] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[18] Robert Bridson. Fast poisson disk sampling in arbitrary dimensions. *SIGGRAPH sketches*, 10(1):1, 2007.

[19] H Bustince, J Fernandez, R Mesiar, Javier Montero, and R Orduna. Overlap functions. *Nonlinear Analysis: Theory, Methods & Applications*, 72(3-4):1488–1499, 2010.

[20] Humberto Bustince, Javier Fernández, Anna Kolesárová, and Radko Mesiar. Directional monotonicity of fusion functions. *European Journal of Operational Research*, 244(1):300–308, 2015.

[21] Humberto Bustince, Miguel Pagola, Radko Mesiar, Eyke Hullermeier, and Francisco Herrera. Grouping, overlap, and generalized bientropic functions for fuzzy modeling of pairwise comparisons. *IEEE Transactions on Fuzzy Systems*, 20(3):405–415, 2011.

[22] Danilo Marco Campanaro, Giacomo Landeschi, Nicoló Dell'Unto, and Anne-Marie Leander Touati. 3d gis for cultural heritage restoration: A 'white box'workflow. *Journal of Cultural Heritage*, 18:321–332, 2016.

[23] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*.

[24] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3d mesh segmentation. *Acm transactions on graphics (tog)*, 28(3):1–12, 2009.

[25] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5556–5565, 2015.

[26] James H Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[27] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[28] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[29] Ismael Colomina and Pere Molina. Unmanned aerial systems for photogrammetry and remote sensing: A review. *ISPRS Journal of photogrammetry and remote sensing*, 92:79–97, 2014.

[30] Robert L Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics (ToG)*, 1(1):7–24, 1982.

[31] Mark De Berg. *Computational geometry: algorithms and applications*. Springer Science & Business Media, 2000.

[32] Clive S Fraser. Automatic camera calibration in close range photogrammetry. *Photogrammetric Engineering & Remote Sensing*, 79(4):381–388, 2013.

[33] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32(8):1362–1376, 2009.

[34] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[35] Ronan Gaugne, Jean-Baptiste Barreau, Flavien Lécuyer, Théophane Nicolas, Jean-Marie Normand, and Valérie Gouranton. extended reality for cultural heritage. In *Handbook of Cultural Heritage Analysis*, pages 1405–1437. Springer, 2022.

[36] Aleksey Golovinskiy, Vladimir G Kim, and Thomas Funkhouser. Shape-based recognition of 3d point clouds in urban environments. In *2009 IEEE 12th International Conference on Computer Vision*, pages 2154–2161. IEEE, 2009.

[37] Daniel Gómez, J Tinguaro Rodriguez, Javier Montero, Humberto Bustince, and Edurne Barrenechea. n-dimensional overlap functions. *Fuzzy Sets and Systems*, 287:57–75, 2016.

[38] Michel Grabisch, Jean-Luc Marichal, Radko Mesiar, and Endre Pap. *Aggregation functions*, volume 127. Cambridge University Press, 2009.

[39] Michel Grabisch, Jean-Luc Marichal, Radko Mesiar, and Endre Pap. Aggregation functions: Construction methods, conjunctive, disjunctive and mixed classes. *Information Sciences*, 181(1):23–43, 2011.

[40] Jim Gray et al. The transaction concept: Virtues and limitations. In *VLDB*, volume 81, pages 144–154, 1981.

[41] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech 256, Apr 2022.

[42] Eleonora Grilli, Domenica Dininno, Lucia Marsicano, Giulio Petrucci, and Fabio Remondino. Supervised segmentation of 3d cultural heritage. In *2018 3rd digital heritage international congress (DigitalHERITAGE) held jointly with 2018 24th international conference on virtual systems & multimedia (VSMM 2018)*, pages 1–8. IEEE, 2018.

[43] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.

[44] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[46] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery.

[47] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11108–11117, 2020.

[48] Yan Huang, Jingliang Peng, C-C Jay Kuo, and M Gopi. Octree-based progressive geometry coding of point clouds. In *PBG@ SIGGRAPH*, pages 103–110, 2006.

[49] Martin Isenburg. Laszip: lossless compression of lidar data. *Photogrammetric engineering and remote sensing*, 79(2):209–217, 2013.

[50] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986.

[51] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[52] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

[53] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[54] EP Klement, Radko Mesiar, and E Pap. Triangular norms. *Tatra Mountains Mathematical Publications*, 13:169–193, 1997.

[55] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[56] Samuli Laine and Tero Karras. Efficient sparse voxel octrees. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 55–63, 2010.

[57] Yann Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3.

[58] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

[59] Chen-Yu Lee, Patrick W Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial intelligence and statistics*, pages 464–472. PMLR, 2016.

[60] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial intelligence and statistics*, pages 562–570. Pmlr, 2015.

[61] Fei-Fei Li, Marco Andreeto, Marc'Aurelio Ranzato, and Pietro Perona. Caltech 101, Apr 2022.

[62] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018.

[63] Max Limper, Stefan Wagner, Christian Stein, Yvonne Jung, and André Stork. Fast delivery of 3d web content: A case study. In *Proceedings of the 18th International Conference on 3D Web Technology*, pages 11–17, 2013.

[64] Peter Lindstrom and Greg Turk. Image-driven simplification. *ACM Transactions on Graphics (ToG)*, 19(3):204–241, 2000.

[65] Sotiris Logothetis, Adelinas Delinasiou, and Efstratios Stylianidis. Building information modelling for cultural heritage: a review. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2:177–183, 2015.

[66] William E Lorensen. Marching through the visible man. In *Proceedings Visualization'95*, pages 368–373. IEEE, 1995.

[67] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

[68] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019.

[69] Rufael Mekuria, Kees Blom, and Pablo Cesar. Design, implementation, and evaluation of a point cloud codec for tele-immersive video. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(4):828–842, 2016.

[70] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019.

[71] Radko Mesiar, Anna Kolesárová, and Magda Komorníková. Aggregation functions on [0, 1]. *Springer Handbook of Computational Intelligence*, pages 61–74, 2015.

[72] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[73] Tomas Möller. A fast triangle-triangle intersection test. *Journal of graphics tools*, 2(2):25–30, 1997.

[74] Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2227–2240, 2014.

[75] Jawad Nagi, Frederick Ducatelle, Gianni A Di Caro, Dan Cireşan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, and Luca Maria Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE international conference on signal and image processing applications (ICSIPA)*, pages 342–347. IEEE, 2011.

[76] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[77] Fred Edwin Nicodemus, Joseph C Richmond, Jack J Hsia, Irving W Ginsberg, Thomas Limperis, et al. *Geometrical considerations and nomenclature for reflectance*, volume 160. US Department of Commerce, National Bureau of Standards Washington, DC, USA, 1977.

[78] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian conference on computer vision, graphics & image processing*, pages 722–729. IEEE, 2008.

[79] Tomoyuki Nishita, Thomas W Sederberg, and Masanori Kakimoto. Ray tracing trimmed rational surface patches. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 337–345, 1990.

[80] Edwin Olson, John Leonard, and Seth Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2262–2269. IEEE, 2006.

[81] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

[82] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation.* MIT Press, 2023.

[83] Massimiliano Pieraccini, Gabriele Guidi, and Carlo Atzeni. 3d digitizing of cultural heritage. *Journal of Cultural Heritage*, 2(1):63–70, 2001.

[84] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[85] Ramona Quattrini, Roberto Pierdicca, and Christian Morbidoni. Knowledge-based data enrichment for hbim: Exploring high-quality models using the semantic-web. *Journal of Cultural Heritage*, 28:129–139, 2017.

[86] Tahir Rabbani, Frank Van Den Heuvel, and George Vosselmann. Segmentation of point clouds using smoothness constraint. *International archives of photogrammetry, remote sensing and spatial information sciences*, 36(5):248–253, 2006.

[87] Gilles Rainer, Wenzel Jakob, Abhijeet Ghosh, and Tim Weyrich. Neural btf compression and interpolation. In *Computer Graphics Forum*, volume 38, pages 235–244. Wiley Online Library, 2019.

[88] Fabio Remondino. Heritage recording and 3d modeling with photogrammetry and 3d scanning. *Remote sensing*, 3(6):1104–1138, 2011.

[89] Fabio Remondino and Stefano Campana. *3D recording and modelling in archaeology and cultural heritage*. British Archaeological Reports Oxford, 2014.

[90] Iosu Rodriguez-Martinez, Tiago da Cruz Asmus, Graçaliz Pereira Dimuro, Francisco Herrera, Zdenko Takáč, and Humberto Bustince. Generalizing max pooling via (a, b)-grouping functions for convolutional neural networks. *Information Fusion*, 99:101893, 2023.

[91] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009.

[92] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[93] Ruwen Schnabel and Reinhard Klein. Octree-based point-cloud compression. *PBG@ SIGGRAPH*, 3(3), 2006.

[94] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007.

[95] William J Schroeder, Jonathan A Zarge, and William E Lorensen. Decimation of triangle meshes. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, 1992.

[96] Roberto Scopigno, Paolo Cignoni, Nico Pietroni, Marco Callieri, and Matteo Dellepiane. Digital fabrication techniques for cultural heritage: a survey. In *Computer graphics forum*, volume 36, pages 6–21. Wiley Online Library, 2017.

[97] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 1, pages 519–528. IEEE, 2006.

[98] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[99] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[100] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. Optimized spatial hashing for collision detection of deformable objects. In *Vmv*, volume 3, pages 47–54, 2003.

[101] Shimon Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153):405–426, 1979.

[102] Eric Veach. *Robust Monte Carlo methods for light transport simulation*. Stanford University, 1998.

[103] George Vosselman and Hans-Gerd Maas. *Airborne and terrestrial laser scanning*. CRC Press (Taylor & Francis), 2010.

[104] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[105] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

[106] Cheng Zhang, Zihan Yu, and Shuang Zhao. Path-space differentiable rendering of participating media. *ACM Transactions on Graphics (TOG)*, 40(4):1–15, 2021.

Index