Nearest Neighbor Sampling of Point Sets
using Random Rays

Liangchen Liu$^{1†}$, Louis Ly$^{2†}$, Colin B. Macdonald$^3†$
and Richard Tsai$^{1,2†}$

$^1$Department of Mathematics, The University of Texas at
Austin, 2515 Speedway, Austin, 78712, TX, USA.

$^2$Oden Institute for Computational Engineering and Sciences,
The University of Texas at Austin, 201 E 24th St, Austin, 78712,
TX, USA.

$^3$Department of Mathematics, University of British Columbia,
1984 Mathematics Rd, Vancouver, V6T 1Z2, BC, Canada.

*Corresponding author(s). E-mail(s): lcliu@utexas.edu;
Contributing authors: louisly@utexas.edu; cbm@math.ubc.ca;
ytsai@math.utexas.edu;
†These authors contributed equally to this work.

Abstract

We propose a new framework for the sampling, compression, and
analysis of distributions of point sets and other geometric objects
embedded in Euclidean spaces. Nearest neighbors of points on a
set of randomly selected rays are recorded into a tensor, called
the RaySense signature. From the signature, statistical information
about the data set, as well as certain geometrical information,
can be extracted, independent of the ray set. We present a few
examples illustrating applications of the proposed sampling strategy.

Keywords: point clouds, sampling, classification, registration, deep learning,
Voronoi cells

MSC Classification: 68T09, 65D19, 68T07, 65D40
1 Introduction

We propose a novel method for sampling and analyzing point clouds, or other geometric objects, embedded in high dimensions. We call our approach “RaySense” because it samples by firing randomly-chosen rays through the ambient space occupied by the object. A ray senses the structure of the object by recording some functions of the nearest neighbors in the object to points on the ray. We can then work with this sampled data—the RaySense signature—instead of the original object, which can be a point cloud, triangulated surface, volumetric representation, etc. An important feature of this approach is that a point in the object may be sampled multiple times by the same or different rays.

RaySense assumes that the object is embedded in a Euclidean space, \( \mathbb{R}^d \) using some suitable representation. For example, \( d = 3 \) for CAD models of a physical object [1, 2], an implicit surface, or a collection of shapes, discretized and represented as black-and-white images. \( \Gamma \) may contain point sets from geometrical objects containing parts of different Hausdorff dimensions; e.g., solids balls inter-connected by line segments. In general, RaySense will work on data already transformed into a suitable feature space. We assume the objects to be compared are calibrated, e.g., centered, rotated, and scaled consistently.

When the object is a point cloud, the RaySense samples are easy to find via discrete nearest neighbor searches. There are computationally efficient algorithms for performing nearest-in-Euclidean-distance neighbor searches, for example, tree-based algorithms [3], grid-based algorithms [4], and building an indexing structure [5]. For very high dimensions, one may consider randomized nearest neighbor search algorithms such as [6, 7] or certain dimension reduction approaches. We show that certain statistical information from the sampled data are dependent only on the ray distribution, not specific ray sets.

When the object is a smooth submanifold in \( \mathbb{R}^d \), one can easily extract local geometrical information (such as curvature) from the nearest points of the rays. When \( \Gamma \) is a finite point set on a smooth manifold, curvature information can also be derived from multiple nearest neighbors of each point on the rays. The RaySense signature can also be used to compute line integrals of functions defined over the point cloud.

RaySense has potential applications for registration, classification, segmentation, and compression of data. In this article, we present examples using RaySense for point cloud registration and classification. The size of the RaySense signature can be predetermined for a collection of data sets, while the cardinality of each member data set may vary. Therefore, the use of RaySense signatures offers flexibility in designing algorithms for comparing data sets of different sizes.

We also demonstrate that RaySense, combined with a simple convolution neural network, RayNN, performs at least on par with carefully tuned state-of-the-art convolutional neural networks on 3D point cloud classification problems. Intuitive explanations for RaySense’s success include (a) repeated sampling of salient feature points; and (b) some locality and high-order information related to (suitably defined notions) of curvatures.
1.1 Related work

In the field of Integral Geometry, one uses the probability of intersection of affine subspaces of different dimensions with the target data manifold to deduce information about the manifold. The interaction information obtained from the “sensing” affine subspaces is binary: yes or no. One thus has a counting problem: how frequently will affine subspaces intersect with the data manifold. From these probabilities, one may extract geometrical information about the manifold, based on a duality between the dimensions of the “sensing” subspaces and the Hausdorff dimensions of the data set; see e.g., [8]. Nevertheless, such approaches may be inefficient in practical computations.

Our idea of using rays is perhaps most-closely related to the X-ray transform, which coincides with the Radon Transform in 2D [9]. In an X-ray
Nearest Neighbor Sampling of Point Sets using Random Rays

Fig. 3: Each digit averaged over the entire data set (top) versus those sampled by RaySense (bottom).

transform, one integrates a given real-valued function defined in $\mathbb{R}^d$ along lines; in a Radon transform, one integrates the given function restricted on hyperplanes of $\mathbb{R}^d$.

We advocate using random lines (rays) and recording information about the data along each ray (instead of accumulating binary “yes/no” information or a scalar over each rays). In this paper, we collect points in the data set nearest to the points on the rays along with the values of some function on those points. With such data, we can compute approximate line integrals and relate our method to the X-ray transform.

From the perspective of the computer vision community, RaySense can be considered as a shape descriptor, mapping from 3D point sets to a more informative feature space where point sets can be easily compared. Generally, descriptors try to capture statistics related to local and global features. See [10] for a survey. More recently, researchers have combined shape descriptors with machine learning [11–14]. RaySense applies more generally to data in arbitrary dimensions.

Some methods use machine learning directly on the point set to directly learn features for specific tasks, such as classification [15–23]. PointNet [18] pioneered deep learning on point clouds by applying independent operations on each point and aggregating features via a symmetric function. Based on that, other architectures [19, 24] exploit neighboring information to extract local descriptors. SO-Net [25] uses self-organizing maps to hierarchically group nodes while applying fully-connected layers for feature extraction. PCNN [15] defines an extension and pulling operator similar to the closest point method [26, 27] to facilitate the implementation of regular convolution, while DGCNN [22] and PointCNN [17] generalize the convolution operator to point sets. Rather than using machine learning on the point set, we use the RaySense signature as input. In Sec. 4.5, we present a deep learning model for 3D point cloud classification using this idea. The model is very efficient for classification, and our experiments suggest that by storing $\kappa$ nearest neighbors to points on the rays, the resulting classifiers are robust against outliers.

By recording the closest point coordinates, RaySense can be thought as a sampling scheme. However, the RaySense sampling biases more toward the “salient” points in the underlying geometry, as will be discussed in Sec. 3.1. And by further retaining only the most frequently sampled points, RaySense resembles keypoint detectors [28] or compression algorithms. When the rays are sufficiently long, RaySense will sample points on the portions of the boundary
that have relatively large positive curvature; \cite{29} uses rays and such points to approximate the convex hull. Convex polytopes are used \cite{30} to summarize multivariate point set data.

The ray-casting and ray-tracing communities \cite{31–33} use kd-tree–based algorithms for very efficient computation of nearest-neighbor queries, curvature, and other quantities, for very large sets of rays. These techniques would be useful for improving the efficiency of RaySense implementations.

\section{Methods}

Let $L$ denote a distribution of lines, parameterized by $\theta \in S^{d-1}$ and $b \in \mathbb{R}^d$, where $r \equiv L_{\theta,b} : \mathbb{R} \mapsto \mathbb{R}^d$ denotes a line in $L$ parameterized by its length

$$r(s) = b + s\theta.$$ 

We choose $m$ lines from the distribution $L$. We then work with a finite segment of each line (for example, $0 \leq s \leq 1$ or $-1 \leq s \leq 1$), and we refer to that line segment $r(s)$ as a ray. Appendix A shows some different distributions of lines, and details for choosing segments from them.

Once we have our rays, we sample from the point cloud. Let $\Gamma \subset \mathbb{R}^d$ denote the point set to be analyzed, and $P_\Gamma$ be the nearest neighbor projection to $\Gamma$, i.e.,

$$P_\Gamma x \in \arg \inf_{y \in \Gamma} \|x - y\|_2.$$ \hspace{1cm} (1)

In cases of non-uniqueness, we choose arbitrarily.

Let $f : \Gamma \times \mathbb{R}^d \mapsto X$ be the feature map that maps points from $\Gamma$ and points on a ray into some “feature space” $X$, and assume that $X$ is embedded in $\mathbb{R}^c$. Each random ray $r$ defines a piecewise continuous path in $X$ by the values of $f(P_\Gamma r(s), r(s))$. A simple example of such a function is $f(P_\Gamma r(s), r(s)) = P_\Gamma r(s)$, where our feature space of closest points is $\mathbb{R}^d$ itself ($c = d$). Others are discussed below. In general, each ray records some information belonging to the target feature space. By collecting all the recorded information from the $m$ rays together, we obtain the RaySense signature:

$$S_m(\Gamma, f; L),$$ \hspace{1cm} (2)

consisting of $m$ random samples of these paths.

We will work with a discrete version of the signature defined in the following way. We consider $n_r$ uniformly-spaced points along each ray, with corresponding spacing $\delta r$. With $r_{i,j}$ denoting the $j$-th point on the $i$-th ray, we define the discretized RaySense signature in the following tensor:

$$[S_{m,\delta r}(\Gamma, f; L)]_{i,j} := f(P_\Gamma r_{i,j}, r_{i,j})$$ \hspace{1cm} (3)

For simplicity, we will denote this discrete RaySense signature as “$S(\Gamma, f)$” or simply “$S(\Gamma)$” when $f$ is not the focus of discussion.
In this paper, we consider the RaySense signature to include the coordinates of the closest point to each ray sample point

\[ f(\mathcal{P}_r i,j, r_i,j) = [\mathcal{P}_r i,j]; \quad (4) \]

we will sometimes augment with the vector to the closest point

\[ f(\mathcal{P}_r i,j, r_i,j) = [\mathcal{P}_r i,j, \mathcal{P}_r i,j - r_i,j], \quad (5) \]

in particular in the neural network experiments in Sec. 4.5. The signature can also be extended to include features from multiple (say \(\kappa\)) nearest-neighbors, as well as the distance from the ray sample point to the closest point. We will see that incorporating additional neighbors into the signature increases robustness to outliers.

Since the signature for any point set is a fixed-size tensor storing useful feature information, one might compare two point sets (of potentially different cardinalities) via comparing a RaySense signature of each.

### 2.1 Comparing RaySense signatures

A natural idea is to choose a suitable metric to define the distances between the RaySense signature tensors.

**The Frobenius norm of the signature tensor** is suitable if the signatures contain the distance and the closest point coordinates. For data sampled from smooth objects such signature information along each ray is piecewise continuous. Thus, if the signatures are generated using the same set of sampling rays, one may compare the RaySense signatures of different data sets using the Frobenius norm.

**Wasserstein distances** are more appropriate for comparison of histograms of the RaySense data, especially when the signatures are generated by different sets of random rays. The normalized histograms can be regarded as probability distributions. In particular, notice (Fig. 6) that RaySense histograms tend to have “spikes” that correspond to the salient points in the data set; \(\ell^2\) distances are not adequate for comparing distributions with such features.

Here we briefly describe the Wasserstein-1 distance, or Earth mover’s distance, that we used in this paper. Let \((X, \mu_1)\) and \(\tilde{X}, \mu_2\) be two probability spaces and \(F\) and \(G\) be the cumulative distribution functions of \(\mu\) and \(\nu\), respectively. The Wasserstein-1 distance is defined as

\[ W_1(\mu_1, \mu_2) := \int_{\mathbb{R}} |F(t) - G(t)| \, dt. \]

**Neural networks:** One can consider using a properly designed and trained neural network. In Sec. 4.5, we present a neural network model, RayNN, for comparing point clouds in three dimensions based on RaySense signatures.
3 Properties of RaySense signature

3.1 Sampling of salient points

Roughly speaking, “salient” points in a data set correspond to the points near the singularity in the geometry of the data set. They tend to be sampled more frequently by RaySense.

For discrete point sets, the likelihood that a ray senses a particular point is related to the Voronoi cell of the point. In Fig. 1, we show examples of two rays sensing various 3D point clouds, along with the corresponding features of the signature tensor. Fig. 2 shows how often each point is “sensed” by different rays. Points that are sampled by multiple rays are larger. Fig. 5 shows that salient points are more likely to be sensed by a ray due to their larger Voronoi cells. We shall return with a more in-depth discussion in Sec. 3.4.

We consider the MNIST dataset [34], treating each image as a point in $d = 784$ dimensions. Here $\Gamma$ is the point set consisting of all images of the same digit. Fig. 3 shows the average digits over the whole dataset, versus the average of those sampled by RaySense. In the context of MNIST, salient points are digits that are drawn using less typical strokes (according to the data). These are the data points that may be harder to classify, since they appear less frequently in the data. RaySense may be used to determine the most useful data points to label, as in active learning [35]. RaySense also provides a special notion of importance sampling based on the notion of saliency described above.

3.2 Curvature information

If the object is a smooth manifold in $\mathbb{R}^d$ (e.g., the sphere in Fig. 1) then each ray induces a piecewise smooth parameterized curve $\gamma(t) \in \mathbb{R}^d$. The curvature of $\gamma(t)$ is $\kappa = \sqrt{\|\gamma'\|^2\|\gamma''\|^2 - (\gamma' \cdot \gamma'')^2/\|\gamma'\|^3}$, and can be approximated by finite differences of $\gamma(t)$ at consecutive values of $t$ along the ray. For example, a typical experiment with $\delta r = 0.05$ when $\Gamma$ is a unit sphere gave values such as 0.99913, 0.99825 and 0.99748, compared to the exact value of one.

Thus even with only a few rays, we obtain local samples of higher-order geometric information. Note the calculations in this example can be performed as combinations of $1 \times 3$ convolutions along the ray; thus we can expect RayNN in Sec. 4.5 to have access to curvature information.

Another notion of curvature for finite convex point clouds at a point $x$ is related to the spherical volume of the set of all unit normals of supporting hyperplanes at $x$ [36]. When the rays are sufficiently long, RaySense will sample points on the portions of the boundary that have relatively large positive curvature because the corresponding Voronoi cells are unbounded and relatively larger than other cells on the boundary.

In [29], one uses points with large curvature to approximate the convex hull of the underlying point set, where the approximation to the spherical volume is obtained by rays fired from the origin. This suggests RaySense can also be applied as a convex hull approximation scheme. By studying how these volumes
tend to infinity, its likely that RaySense can be used to infer dimensionality of
the support of data distribution; a topic for future study.

3.3 “Coverage” of an object by RaySense

Our next experiments look at how well a set of rays “cover” data sets in higher
dimensional Euclidean spaces. One way to measure this is to compute the
maximum distance from every point in an object \( \Gamma \) to points that contribute
to the signature \( S(\Gamma) \). The smaller this value, the closer we are to sampling the
entire object. We experiment with different point clouds of various dimensions
in Fig. 4. Notably, the coverage does not strongly depend on \( m \), the number of
rays in a random ray set, when the object is inherently lower dimension and
merely “rotated” into the higher dimensional space (first column of Fig. 4). If
the object is more complicated, we may need more rays to attain the same
coverage as the dimension increases (Fig. 4 top-right). Nonetheless, we often
obtain coverage that is roughly dimension-independent (Fig. 4 bottom-right).

![Fig. 4: Coverage of point clouds in various dimensions by RaySense using \( m \) rays with 32 samples per ray. Top: 5000 points sampled from curves. Bottom: 25000 points sampled from hemispheres. Low-dimensional examples embedded by random rotations into \( \mathbb{R}^d \). Noise of size \( 10^{-4} \) added and results averaged over 40 realizations.](image)

3.4 Histograms of RaySense samples

If we collect the histogram of the points sampled from a set of randomly
selected rays, we can show that the histogram has a well-defined limit as the
number of rays tends to infinity. Let \( U \in \mathbb{R}^d \) be a solid dimension-\( d \) ball of
suitable radius, and \( \Gamma \subsetneq U \) is a finite point set containing \( N \) distinct points.
Let $V_j$ denote the Voronoi cell for the $j$-th point, $x_j$ in $\Gamma$, as seen in Fig. 5. Let $\ell_j(r)$ denote the length of the segment of a ray, $r$, that lies in $V_j \cap U$. If $r$ does not intersect $V_j$, $\ell_j(r) := 0$. Thus, $\ell_j$ is a random variable, and we denote it’s expectation by $\mathbb{E}[^{\ell_j}]$; in other words,

$$
\mathbb{E}[\ell_j] := \int \ell_j(\omega) \, d\mu_L(\omega),
$$

where $\mu_L$ is the probability measure corresponding to the distribution $L$. See Fig. 5 for an illustration.

**Fig. 5:** A simple 2D point set (gray). Two rays (black) sense nearest neighbors of the point set (blue). Singular points, such as the tip of the tail, have larger Voronoi cells (dashed lines) and are more likely to be sampled. Closest point pairs are shown in green and red.

A hybrid Monte-Carlo approach can approximate $\mathbb{E}[\ell_j]$. Draw $m$ rays, $r_1, r_2, \ldots, r_m$, from the distribution $L$. On each ray, collect the closest points in $\Gamma$ from equidistant points that lie within $U$. Enumerate this set of points by $r_{i,j}$. Let $\delta r_i < \delta r$ denote the spacing between two adjacent points on $r_i$. The closest point of $r_{i,j}$ is $x_k$ if $r_{i,j} \in V_k$. If $r_{i,j}$ lies on the boundary of different Voronoi cells, we pick one randomly. Let

$$
H_k(S_{m,\delta r}(\Gamma)) := \frac{1}{m} \sum_{i=1}^{m} \sum_{r_{i,j} \in V_k} \delta r_i.
$$

Here, $S_{m,\delta r}(\Gamma)$ denotes the signature tensor. This $H_k$ is precisely the number of times $x_k$ is sampled by the RaySense approach, normalized by $\delta r_i/m$. Therefore, we arrive at the following Theorem:

**Theorem 1** Convergence of RaySensed data histograms:

$$
\lim_{m \to \infty} \lim_{\delta r \to 0} H_k(S_{m,\delta r}(\Gamma)) = \mathbb{E}[\ell_k].
$$
Monte-Carlo approximations of integrals converge with a rate independent of the dimension. Consequently, for sufficiently many randomly selected rays, the histogram is essentially independent of the rays that are actually used.

Similar arguments show that the sampling of any function of the data set will be independent of the actual ray set, since the histograms are identical in the limit. More precisely, suppose \( g : x \in \Gamma \mapsto \mathbb{R} \) is some function, then

\[
\lim_{m \to \infty} \frac{1}{m} \sum_{i=1}^{m} \sum_{r_{i,j} \in V_k} g(x_k) \delta r_i = \mathbb{E}[g(x_k)\ell_k].
\]

In Fig. 6, we show the histograms of the coordinates of the RaySensed points of \( \Gamma \).

The integral \( \mathbb{E}[\ell_j] \) (or \( \mathbb{E}[g(x_k)\ell_j] \) for continuous \( g \)) depends smoothly on \( \Gamma \), and is therefore stable against perturbation to the coordinates of the points in \( \Gamma \). However, the effect of introducing new members to \( \Gamma \), such as outliers, will be non-negligible. One possible way to overcome this is to use multiple nearest neighbors for points on the rays. Such information will be different for the outliers.

![Fig. 6: Histogram of coordinates from two point sets. Columns 1 and 2 correspond to 2 different sets of rays, each containing 50 rays and 50 samples per ray. These histograms are similar for the same object and different for different objects. Column 3 corresponds to the entire point cloud; these differ from the RaySense histograms.](image)

### 3.5 Approximate line integrals

In the previous sections, sampling a point multiple times is related to the size of the associated Voronoi cell; that is the “saliency” factor for RaySense. On the other hand, RaySense maintains fidelity to the point cloud in various ways. This section shows an example: when the rays stay within the support of the underlying density of the point cloud, the RaySense signature can be post-processed to compute line integrals of functions defined on the points in the point cloud.
Suppose we have a family of point clouds $\Gamma_N$ representing an object in $\mathbb{R}^d$. As $N$ increases, the point cloud becomes denser, and $\mathcal{P}_r r(s) \approx r(s)$ as the Voronoi cells shrink around each point. If we have function $g : \mathbb{R}^d \to \mathbb{R}$ evaluated on the point cloud, then $g(\mathcal{P}_r r(s)) \approx g(r(s))$ and we expect that integrals of $g$ along lines can be approximated by integrals of the RaySense signature (and quadrature of the discrete RaySense signature). We make this precise below, and discuss implications for spectral calculations.

**Assumptions and notations** for this section:

A1 The point set, $\Gamma \equiv \Gamma_N$, is a realization of a collection of $N$ i.i.d. random vectors $\{X_i\}_{i=1}^N$.

A2 The probability space induced by the random vector $X_i$ is $(\mathbb{R}^d, \mathcal{F}, \mu)$, where $\mathcal{F}$ is the Borel $\sigma$-algebra on $\mathbb{R}^d$ and $\mu$ is a probability measure.

A3 The induced probability measure $\mu$ is also known as the distribution of $X_i$, with compactly supported Lipschitz density $\rho$.

A4 The RaySense signature is of the closest points feature space (4).

A5 In the case that $r(s)$, for some $s$, has more than one nearest neighbor, we will randomly assign one, say $x_k$, and say that $r(s)$ intersect with the Voronoi cell, $V_k$.

A6 We will assume that the ray, $r(s)$, $0 \leq s \leq 1$, is given entirely in the interior of $\text{supp}(\rho)$, and it intersects $M$ distinct Voronoi cells, and use $k_i, i = 1, 2, \ldots, M$, to refer to the corresponding points $x_{k_i} \in \Gamma_N$.

A7 We assume $\text{supp}(\rho)$ can be covered by a finite union of hypercubes $\{\Omega_j\}_j$ in $\mathbb{R}^d$, each of non-zero probability measure, i.e., $\text{supp}(\rho) \subset \bigcup_j \Omega_j$, with $P_{\Omega_j} = \int_{x \in \Omega_j} \rho(x)dx > 0$, and $\{\Omega_j\}_j$ overlap with each other only on sets of measure zero.

Note that by A2-A3, we may regard $\rho$ as representing the density of a solid body in $\mathbb{R}^d$. In other words, $\Gamma_N$ does not consist of samples from a lower dimensional set. (RaySense can sample much more general sets, but the line integral analysis of this section may not hold.)

Define the RaySense integral along a ray $r(s)$, $0 \leq s \leq 1$, as follows:

$$
\int_0^1 g(\mathcal{P}_{\Gamma_N} r(s))ds = \int_0^1 g(x_{k(s)})ds, \quad x_{k(s)} := \mathcal{P}_{\Gamma_N} r(s) \in \Gamma_N,
$$

where $k(s) \in \{1, 2, \ldots, N\}$ indexes the nearest neighbor of $r(s)$ in $\Gamma_N$. Note that in practice, (6) would be approximated with quadrature, a weighted sum of discrete values of $g$.

The following Theorem shows that the RaySense integral can be thought as an approximation to the true line integral along $r(s)$ provided there is some control on the size of the Voronoi cells.

**Theorem 2** Suppose that $g \in C(\mathbb{R}^d; \mathbb{R})$ is $J$-Lipschitz, ray $r(s) \in \text{supp}(\rho)$ for $0 \leq s \leq 1$, and $\Gamma_N$ is a set of $N$ i.i.d. random samples drawn from $\mu$, with $(x_{k(s)}, g(x_{k(s)}))$ denoting the corresponding RaySense data, then the difference
between the RaySense integral of $g$ and the line integral of $g$ has the following bound:

$$\left| \int_0^1 g(r(s))\,ds - \int_0^1 g(x_{k(s)})\,ds \right| \leq \frac{J}{2} \sum_{i=1}^M \text{diam} (V_{k_i}) \text{diam}(V_{k_i} \cap \text{supp}(\rho)), \quad (7)$$

where $1 \leq k_i \leq N$ such that $\{x_{k_i}\}_{i=1}^M \subset \Gamma_N$ are points in $\Gamma_N$ sensed by the ray and $V_{k_i}$ are their corresponding Voronoi cells.

**Proof** For a fixed number of sampling points $N$, hence a fixed $M$, the approximation error is given by

$$\left| \int_0^1 g(r(s))\,ds - \int_0^1 g(x_{k(s)})\,ds \right| \leq \int_0^1 \chi(\{r(s) \in V_{k_i}\}) \left| g(r(s)) - g(x_{k_i}) \right|\,ds$$

$$\leq J \sum_{i=1}^M \int_0^1 \chi(\{r(s) \in V_{k_i}\}) \|r(s) - x_{k_i}\|\,ds$$

$$\leq J \sum_{i=1}^M \int_0^1 \text{diam}(V_{k_i} \cap \text{supp}(\rho)) \|r(s) - x_{k_i}\|\,ds$$

$$\leq \frac{J}{2} \sum_{i=1}^M \text{diam} (V_{k_i}) \text{diam}(V_{k_i} \cap \text{supp}(\rho)),$$

where $\chi$ denotes an indicator function. \hfill \square

In scattered data interpolation, the nearest neighbor interpolation would have an error of $O(h)$ where $h = \max_k \text{diam}(V_k)$. Integration of the interpolated data would result in an error of $O(h^2)$, consistent with Theorem 2.

Intuitively, for a fixed density $\rho$ with compact support, as the cardinality $N$ of the point cloud $\Gamma_N$ increases, the Voronoi cells in the interior shrink. Making this precise (in a series of lemmas, with proofs in Appendix C), we obtain error estimates for the line integral in terms of $N$.

Our first lemma tells us how large the point cloud must be in order to have at least one sample in any region achieving a certain probability measure.

**Lemma 3** Suppose that $\text{supp}(\rho)$ satisfies assumption A7. Let $\Gamma_N$ be a set of $N$ i.i.d. random samples drawn from $\mu$, and $p_0 \in (0, 1)$. If the number of sample points $N > \nu(P)$ where $\nu : \mathbb{R} \to \mathbb{R}$ is defined by

$$\nu(P) := \sqrt{\ln \left( \frac{2}{1-p_0} \right) \left( \ln \left( \frac{2}{1-p_0} \right) + 2P \right) + P + \ln \left( \frac{2}{1-p_0} \right) \frac{P^2}{2} }, \quad (8)$$

then, with probability at least $p_0$, at least one of the samples lies in every $\Omega_j$ with $P_{\Omega_j} \geq P$. Additionally, we note bounds for $\nu(P)$:

$$\frac{2 \ln \left( \frac{2}{1-p_0} \right) + P}{P^2} < \nu(P) < \frac{2 \ln \left( \frac{2}{1-p_0} \right) + 3P}{P^2}. \quad (9)$$
We notice that $\nu(P) \sim O\left(\frac{1}{P^2}\right)$ as $P \to 0$ indicating $\nu(P)$ is inversely proportional to $P^2$ asymptotically. This matches with the intuition that more points are needed to ensure sampling in regions with smaller probability measure.

The next two lemmas reveal that the volume of the Voronoi cell for a sample point amongst the others in the point cloud $\Gamma_N$ decreases to zero with high probability as the number of sampled points tends to $\infty$.

**Lemma 4** Suppose $\rho$ is $L$-Lipschitz and $\text{supp}(\rho)$ satisfies assumption A7. Given $\Gamma_N$ a set of $N$ i.i.d. random samples drawn from $\mu$, for $N$ large enough, the size of the Voronoi cell for a sample point $x_k \in \Gamma_N$ in the interior of $\text{supp}(\rho)$ is inversely proportional to its local density $\rho(x_k)$, and with probability at least $p_0$ its diameter has the following upper bound:

$$\text{diam}(V(x_k)) \leq 3\sqrt{d} \left( \frac{21 + 7(9 + 8N \ln \left( \frac{2}{1 - p_0} \right))^{1/2}}{6 \rho(x_k) N} \right)^{\frac{1}{2}}.$$

When the underlying distribution $\mu$ is uniform, $\rho(x_k)$ is the same everywhere inside $\text{supp}(\rho)$, therefore the Voronoi diameter for every $x_k$ in the interior should shrink uniformly. However, a better bound can be obtained for this case, shown in the following lemma.

**Lemma 5** If the distribution $\mu$ is a uniform distribution, then given $\Gamma_N$ with $N$ large enough, with probability at least $p_0$, the diameter of the Voronoi cell of any sample point in the interior has the bound

$$\text{diam}(V) \leq 3\sqrt{d} \left( \left| \frac{1}{c(N)N} \ln \left( \frac{N}{1 - p_0} \right) \right| \right)^{\frac{1}{2}},$$

with some $c(N)$, $c(N) \to 1$ as $N \to \infty$.

We apply these lemmas to Theorem 2 to state our main result on the convergence of the RaySense integral to the line integral.

**Theorem 6** Suppose $g \in C(\mathbb{R}^d; \mathbb{R})$ is $J$-Lipschitz, $\rho$ is $L$-Lipschitz and satisfies assumption A7, and ray $r(s) \in \text{supp}(\rho)$ for $0 \leq s \leq 1$, Then given $\Gamma_N$ a set of $N$ i.i.d. random samples drawn from $\mu$, with $(x_{k(s)}, g(x_{k(s)}))$ being the corresponding RaySense data, with probability at least $p_0$,

$$\left| \int_0^1 g(r(s)) ds - \int_0^1 g(x_{k(s)}) ds \right| \leq \frac{9dJ}{2} \sum_{i=1}^M \left( \frac{21 + 7(9 + 8N \ln \left( \frac{2}{1 - p_0} \right))^{1/2}}{6 \rho(x_k_i) N} \right)^{\frac{2}{3}}.$$

When the distribution is uniform:

$$\left| \int_0^1 g(r(s)) ds - \int_0^1 g(x_{k(s)}) ds \right| \leq \frac{9dJ}{2} \sum_{i=1}^M \left( \left| \frac{1}{c(N)N} \ln \left( \frac{N}{1 - p_0} \right) \right| \right)^{\frac{2}{3}}.$$
Proof The ray $r(s)$ always senses finitely many points from $\Gamma_N$; let that number be $M$ and denote the sensed points by $\{x_{k_i}\}_{i=1}^M$. From Theorem 2:

$$\left| \int_0^1 g(r(s))ds - \int_0^1 g(x_{k}(s))ds \right| \leq \frac{d}{2} \sum_{i=1}^{M} \text{diam}(V_{k_i}) \text{diam}(V_{k_i} \cap \text{supp}(p)),$$

where $\text{diam}(V_{k_i})$ is the diameter of the Voronoi cell $V_{k_i}$ of point $x_{k_i}$. The remainder of the argument uses a few lemmas to obtain a bound on $\text{diam}(V_{k_i})$ in terms of $N$.

We first analyze the case in which $x_{k_i}$ are far away from the boundary, such that $V_{k_i} \cap \text{supp}(p) = V_{k_i}$ so $\text{diam}(V_{k_i} \cap \text{supp}(p)) = \text{diam}(V_{k_i})$. Then for some sufficiently large $N$ (to be discussed below), Lemma 4 uses a covering of hypercubes to bound—with probability $p_0$—the diameter of the Voronoi cell of $x_{k_i}$ in terms of $N$ and the density $\rho$, whence

$$\left| \int_0^1 g(r(s))ds - \int_0^1 g(x_{k}(s))ds \right| \leq \frac{9dJ}{2} \sum_{i=1}^{M} \left( \left\lfloor \frac{21 + 7(9 + 8N \ln \left( \frac{2}{1-p_0} \right))^{1/2}}{6\rho(x_{k_i})N} \right\rfloor \right)^\frac{2}{3}.$$

How large does $N$ need to be? From the proof of Lemma 4, $\rho(y)$ must, for every hypercube center $y$, satisfy:

$$\rho(y) \geq 3\sqrt{d} L l_N(y), \quad \text{with} \quad l_N(y) = \left( \frac{3 + (9 + 8N \ln \left( \frac{2}{1-p_0} \right))^{1/2}}{\rho(y)N} \right)^\frac{1}{3}.$$

As noted in the proof of Lemma 4, choosing such an $N$ is possible because $\ell_N(y)$ is a decreasing function of $N$.

A similar argument holds for a uniform distribution using Lemma 5: with $c(N) \to 1$ as $N \to \infty$,

$$\left| \int_0^1 g(r(s))ds - \int_0^1 g(x_{k}(s))ds \right| \leq \frac{9dJ}{2} \sum_{i=1}^{M} \left( \left\lfloor \frac{1}{c(N)N} \ln \left( \frac{N}{1-p_0} \right) \right\rfloor \right)^\frac{2}{3}.$$

The inequality holds for any $x_{k_i}$ in the interior, as long as $\text{dist}(x_{k_i}, \partial \text{supp}(p)) > \frac{3}{2} \sqrt{d} l_N(y_{k_i})$, where $y_{k_i}$ is the center of the corresponding hypercube for $x_{k_i}$.

Finally, since $r(s)$ is given in the interior, for $N$ large enough no $x_{k_i}$ will be on the boundary of $\text{supp}(p)$ almost surely, hence we complete the proof. \hfill \Box

Theorem 6 tells us that if $M \sim o(N^{1/d})$, or $M \sim o(N/\ln(N))^{2/d}$ under a uniform distribution, the RaySense integral will converge to the line integral.

Intuitively, for sufficiently large $N$, the RaySensed sampled points along the ray will stay extremely close to $r(s)$ such that, with a high probability, the segment in one of the Voronoi cell $\chi(\{r(s) \in V_{k_i}\})$ is roughly the same scale as $\text{diam}(V_{k_i})$. Therefore, asymptotically we expect an upper bound for $M$ to be roughly in the form of:

$$M \lesssim \frac{1}{\alpha(N) \min_{k_i} \text{diam}(V_{k_i})} \implies M \lesssim \mathcal{O}(\left(\frac{1}{\sqrt{N}}\right)^{\frac{2}{3}}),$$

where $0 < \alpha(N) < 1$, and $\alpha(N) \to 1$ as $N \to \infty$. As a result,

$$\left| \int_0^1 g(r(s))ds - \int_0^1 g(x_{k}(s))ds \right| \lesssim \mathcal{O}(\left(\frac{1}{\sqrt{N}}\right)^{\frac{2}{3}}).$$
Similarly, when the distribution is uniform, $M \approx O \left( \left( \frac{1}{\ln(N)} \right)^{\frac{2}{d}} \right)$ and

$$\left| \int_0^1 g(r(s)) \, ds - \int_0^1 g(x_{k(s)}) \, ds \right| \approx O \left( \left( \frac{1}{N} \ln(N) \right)^{\frac{2}{d}} \right).$$

Therefore, we expect that with a high probability, the convergence rate in general will be bounded by $O \left( \left( \frac{1}{\sqrt{N}} \right)^{\frac{1}{d}} \right)$, improving to $O \left( \left( \frac{1}{N} \ln(N) \right)^{\frac{1}{d}} \right)$ if $\rho$ is uniform.

We first confirm these rates with numerical experiments. We return to explore applications to integral transforms in Sec. 4.3. Fig. 7 shows convergence studies of the RaySense integrals for the uniform density case from Theorem 6. We approximate the RaySense integrals by the Trapezoidal Rule:

$$\int_0^1 g(r(s)) \, ds \approx \sum_{n=1}^{n_r} w_n g(P_{\Gamma_r} r(s_n)) \delta r,$$

with weights $w = \langle \frac{1}{2}, 1, \ldots, 1, \frac{1}{2} \rangle$, and thus additional quadrature errors, typically $O(\delta r^2)$, are incurred [37]. The 5-d example uses an integrand of $g(x_1, x_2, x_3, x_4, x_5) = \cos(x_1 x_2) - x_4 x_5 \sin(x_3)$ and a line $r(s) = \frac{\vec{v}}{\|\vec{v}\|^2} s + \langle 1,1,1,1,1 \rangle$ with $\vec{v} = \langle 2,3,4,5,6 \rangle$. In 4-d, 3-d and 2-d, we use $g(x_1, x_2, x_3, x_4, 1)$, $g(x, y, z, 1, 1)$ and $g(x, y, 0, 0, 0)$ respectively, and drop unneeded components from the line. The exact line integrals were computed with the Octave Symbolic package [38] which uses SymPy [39] and mpmath [40]. Each experiment is averaged over 50 runs.

In Fig. 7, we see that for each fixed number of sample points $n_r$ along the ray, the error decreases at the rate discussed above. From the factor of two in the distance between the results of each fixed $n_r$, infer a first-order decrease in $n_r$, and taken together an overall faster rate of convergence if both $N$ and $n_r$ are increased. In all cases in Fig. 7, the experimental convergence rate is bounded by the predicted rate of $O \left( \left( \frac{1}{N} \ln(N) \right)^{\frac{2}{d}} \right)$ and when $n_r$ is large, we appear to achieve a faster rate of $O \left( \left( \frac{1}{N} \ln(N) \right)^{\frac{1}{d}} \right)$. This suggests a tighter analysis may be possible in the future.

4 Applications

4.1 Comparison of histograms of RaySense samples

We experiment by comparing $\Gamma_i$ drawn from 16384 objects of 16 categories from the ShapeNet dataset [1]. Let $\beta^i$ be the label for object $\Gamma_i$. We compute the histogram $h_x^i, h_y^i, h_z^i$ of the $x, y, z$ coordinates, respectively, for points sampled by 50 rays with $n_r = 10$ samples per ray. We compare the histograms against those corresponding to other objects in the dataset, using

$$D_{i,j} = d(h_x^i, h_x^j) + d(h_y^i, h_y^j) + d(h_z^i, h_z^j),$$
where \( d(\cdot, \cdot) \) is either the \( \ell_2 \) or Wasserstein-1 distance. We sum \( D \) according to the respective labels

\[
M_{a,b} \propto \sum_{i; \beta^i = a} \sum_{j; \beta^j = b} D_{i,j}, \quad a, b = 1, \ldots, 16,
\]

and normalize by the number of occurrences for each \( a, b \) pair. Fig. 8 shows the matrix of pairwise distances \( M \) between the 16 object categories.

Ideally, intra-object distances would be small, while inter-object distances would be large. As expected, Wasserstein-1 is a better metric for comparing histograms. Still, not all objects are correctly classified. When comparing histograms in not sufficient, one may consider using higher-order statistical information or neural networks to learn more complex mappings between the data and label (see Sec. 4.5).

**Fig. 7**: Convergence studies for line integrals approximating from the RaySense signature on point clouds sampled from a uniform density, in dimensions \( d = 2, 3, 4, 5 \). Horizontal dashed lines indicate error inherent to the trapezoidal rule quadrature schemes. Diagonal dashed lines indicate different convergence rates.

**Fig. 8**: Comparison of histogram of the \( x, y, z \) coordinates of points sampled by RaySense, using \( \ell^2 \) and the Wasserstein distance \( W_1 \). Rows and columns correspond to object labels. Red \( \times \) indicate location of the argmin along each row.
4.2 Salient points in the MNIST data

From previous discussion and simulation in Sec. 1, we know RaySense has the ability to detect salient points or boundary points. Here we provide visualization of RaySense salient points on the MNIST dataset.

By vectorizing the MNIST image, each image is a vector in $\mathbb{R}^{784}$ with pixel value from 0 to 255. We generate the random ray set in this ambient space using the method R1 in Appendix A, where each ray has the fixed-length 1, with centers uniformly shifted in the half cube $\left[-\frac{1}{2}, \frac{1}{2}\right]^{784}$. Each ray set has $m = 256$ random rays, with $n_r = 64$ equi-spaced points on each ray. To ensure a good coverage over the data manifold, we rescale the MNIST image by entry-wise dividing so that each data point is constrained in an $\ell^{\infty}$ ball of a certain radius as introduced below; we also shift the dataset to have mean 0.

As mentioned, the RaySense salient points are those in $\Gamma$ sampled most frequently by points from the rays. We record the sampling frequency for each MNIST image, and in Fig. 9 we plot the top-10 images with highest frequency for each class. From the figure, we see that the salient points often correspond to digits with untypical strokes and writing styles, similar to the conclusion obtained from Fig. 3. Fig. 9 further shows that different normalizations of the data (by using scaling values 2550, 5100 and 25500) also affects the sampling, suggesting some kind of multi-resolution analysis may be possible.

4.3 RaySense and integral transforms

A line $r$ in $\mathbb{R}^d$ in the direction of $\theta \in S^{d-1}$ has parameterization $r(s) = b + s\theta$, $s \in (-\infty, \infty)$, with $b \in \mathbb{R}^d$ a reference point on the line. Without loss of generality, let $b$ be in $\theta^\perp$, which is a hyperplane orthogonal to $\theta$ passing through the origin. The X-ray transform for a non-negative continuous function $g$ with compact support [9] is defined on the set of all lines in $\mathbb{R}^d$ by

$$X[g](b, \theta) := \int_{-\infty}^{\infty} g(b + s\theta)ds.$$ (10)

The spectrum of $g$ can be obtained via the Fourier slice theorem [41]:

$$\mathcal{F}[Xg](\theta, \xi) = \mathcal{F}[g](\xi), \quad \xi \in \theta^\perp.$$

When we restrict $\xi$ to be only on a line $\theta^\perp$, we are effectively collecting information on a 2-dimensional slice of $g$ parallel to $\theta^\perp$.

However, when the function $g$ only has a sampling representation, e.g., a point cloud, it is non-trivial to compute such integrals. In Section 3.5, we showed that if $r$ is a member of the sampling ray set, one can compute an approximation of (10) from the RaySense signature obtained from $\{x_j, g(x_j)\}_{j=1}^N$, where $\{x_j\}$ are i.i.d. samples from a known probability density $\rho$. Thus, RaySense provides a convenient alternative in obtaining (or in a sense defining) the Fourier slices of the discrete data set $\{x_j, g(x_j)\}_{j=1}^N$. 

18 Nearest Neighbor Sampling of Point Sets using Random Rays

![Image](image.png)

(a) normalized to 0.1 cube  
(b) normalized to 0.05 cube  
(c) normalized to 0.01 cube  
(d) random subsample

Fig. 9: MNIST digit images with highest RaySense sampling frequencies for each class. Three different normalizations are shown in (a), (b) and (c). Compared to a random subsample (d), we see a wider variety of hand-writing styles in the RaySense output.

The same idea works in higher dimensions which means that one can approximate the X-ray transform from a RaySense signature using suitable ray sets, or, in the random case, we expect RaySense integrals can be regarded as randomized approximations of X-ray transforms.

In Fig. 10 we show an example of using RaySense sampling with prescribed (rather than random) rays to approximate the Radon transform. In this experiment, a point cloud $\Gamma$ (Fig. 10(a) top) with 15010 points, is sampled from density $\rho = \frac{1}{2} - 3xe^{-9x^2-9y^2}$ (Fig. 10(a) bottom)—note denser (darker) region on left and sparser (lighter) region on right. $\Gamma$ has data shown in Fig. 10(b) evaluated from the piecewise constant function $g$, shown by the solid colours (for visualization only; $g$ is only known at the discrete points in $\Gamma$). Blue lines show the locations of the RaySense signature for one particular angle (illustrated with 21 rays but the computation uses 100). We note increasingly jagged lines to the right where the point cloud is sparser. Fig. 10(c) shows that approximate Radon transform computed over 180 degrees in steps of one degree by integrating the RaySense signature using trapezoidal rule at $n_r = 64$ points per ray. Fig. 10(d) shows the filtered back projection computed by the
Octave Image package [42]. Note a more jagged reconstruction on the right where the point cloud is sparsest. If we instead used random rays, we could generate samples at scattered points in the sinogram (Fig. 10(c)) which could then be used for an approximate inverse transform.

Fig. 10: Approximate Radon transform computed with RaySense from point cloud data (a)–(c) and filtered back projection reconstruction (d).

4.4 Point cloud registration

In this section, we explore the application of RaySense to the point cloud registration problem. Given two point sets $\Gamma$ and $\tilde{\Gamma}$ in 3-d consisting of distinct points, registration aims to find the 3-d rotation matrix $U$ and translation vector $b$ to minimize the Euclidean norm of points in correspondence. When the correspondence is known, this is the orthogonal Procrustes problem and the solution can be obtained explicitly via the singular value decomposition. When the correspondence is unknown, one can formulate an optimization problem and solve with various carefully-designed algorithms. Here we choose to use the Iterative Closest Point (ICP) [43] due to its simplicity, which minimizes point-wise Euclidean distance iteratively from the optimization problem

$$\min_{U \in SO(3), b \in \mathbb{R}^3} \sum_{x_j \in \tilde{\Gamma}} \min_{y \in \Gamma} \|U(x_j + b) - y\|_2^2.$$ 

We set up the problem using the Stanford Dragon [44] as a point cloud $\Gamma$ as with 100 000 points. We artificially generate the target point cloud to register by rotating by $\pi/3$ in one direction. We compare the performance of ICP in three scenarios: 1) the original dense point clouds, 2) a uniformly random subsampling (in index) of the point clouds, 3) RaySense samples (without repetition of sampled points) of each point cloud. Specifically, we use $m = 512$ rays, each with $n_r = 64$ sample points, to subsample the original point cloud in RaySense, which usually generates a set of around 800 unique points. We then sample the second point cloud with a different set of rays. For fair comparison, we also subsample 800 points in the case of uniformly random subsampling. We use the root mean square error (RMSE) as our metric, and we also record the convergence time, where the convergence criteria is a threshold of the relative
RMSE. We summarize the performance results in Tab. 1, and we provide some visualization to compare the three different settings in Fig. 11

Table 1: Sample point cloud registration result. Performances are evaluated by registration accuracy (measured by root mean squared error (RMSE)) and computation times. The statistics reported are averaged over 5 runs. “RMSE” is evaluated over the subsampled points while “RMSE(full)” is evaluated over the original point cloud.

| number of points | RMSE       | RMSE(full) | Convergence time (s) |
|------------------|------------|------------|----------------------|
| Vanilla ICP      | 100 000    | 4.544e-06  | 4.544e-06            | 6.319     |
| ICP + random     | 800        | 4.509e-02  | 3.053e-03            | 0.0192    |
| ICP + RaySense   | 804.6      | 2.601e-02  | 1.077e-03            | 0.0116    |

Fig. 11: Registration simulation on a rotated Stanford dragon. Top row: initial pose and sparse samples; Bottom row: registration results. Note that the visualizations (e) and (f) are obtained by applying the transformation computed from the sparse samples in (b) and (c) to the original dense point clouds.

From Tab. 1, it is clear that both the sampling schemes accelerate the registration process drastically by considering only a portion of the original dense point cloud. It also suggests that RaySense sample has slight advantage over the uniform random sample, in both accuracy and convergence time.
However, generating the RaySense samples on the fly needs around 0.65s on average, while generating a random subsample requires only 0.01s.

Fig. 11c again shows that RaySense is sampling salient features. Thus a possible improvement is to use this repetition information from RaySense sampling to perform a weighted registration, for example with the (autodetected) salient features receiving higher weights. This is left for future investigation.

4.5 Point cloud classification using neural networks

We use the RaySense signature to classify objects from the ModelNet dataset [2], using a neural network, which we call RayNN.

We use a postfix notation to indicate more precisely what is included in the RaySense signature, see Sec. 2. We use \( f \) with different number of neighbors, denoted by RayNN-X, where X is related to the input features. For our implementation, while we might use different numbers of nearest neighbors, we always include the closest point coordinates and the vector to closest points in our feature space (\( c = 6 \) fixed). We denote our models by RayNN-cpn where \( n \) denotes the number of nearest neighbors.

Table 2: ModelNet classification results. Here we report our best accuracy results over all experiments. For reference, the test scores for RayNN-cp5 (\( m = 32 \)) has mean around 90.31% and standard deviation around 0.25% over 600 tests.

| Method                  | ModelNet10 | ModelNet40 |
|-------------------------|------------|------------|
| PointNet [18]           | --         | 89.2       |
| PointNet++ [19]         | --         | 90.7       |
| ECC [20]                | 90.8       | 87.4       |
| kd-net [16]             | 93.3       | 90.6       |
| PointCNN [17]           | --         | 92.5       |
| PCNN [15]               | 94.9       | 92.3       |
| DGCNN [22]              | --         | **92.9**   |
| RayNN-cp1 (\( m = 16 \))| 94.05      | 90.84      |
| RayNN-cp5 (\( m = 32 \))| **95.04**  | 90.96      |

We compare with some well-known methods for 3D point cloud classification tasks. In addition to the results reported by [18], we also compare against PointNet.pytorch, a PyTorch re-implementation [45] of PointNet. In all our experiments, we report overall accuracy. Table 2 shows RayNN is competitive. To investigate the robustness of our network, we perform several more experiments.

Robustness to sample size  We repeat the experiments in [18, 22] whereby, after training, data is randomly removed prior to testing on the remaining points. The results in Fig. 12 show that RayNN performs very well with significant missing data.
Fig. 12: Testing DGCNN [22], PointNet [18] and RayNN on ModelNet40 with missing data.

Using fewer rays We experiment with training using a full set of $m = 32$ rays but test using smaller number $\hat{m}$ of rays. Table 3 shows that RayNN can achieve a reasonable score even if only $\hat{m} = 4$ rays are used for inference.

Robustness to outliers This experiment simulates situations where noise severely perturbs the original data during testing. We compare the performance of RayNN-cp1, RayNN-cp5 and PointNet.pytorch in Table 4. The comparison reveals RaySense’s capability in handling unexpected outliers, especially when additional nearest neighbors are used. Note the experiment here is different from that in [18] where the outliers are fixed and included in the training set.

Table 3: Accuracy when testing with a reduced ray set. RayNN-cp1 was trained using $m = 32$ rays. Results averaged over 5 runs.

| $\hat{m}$ | 32  | 16  | 8   | 4   |
|----------|-----|-----|-----|-----|
| $\lambda = 1$ | 88.50% | 86.13% | 74.64% | 43.28% |
| $\lambda = 8$ | 89.77% | 88.94% | 82.97% | 55.24% |

Comparison of model complexity

Table 5 shows that our network has an advantage in model size and feed-forward time even against the simple and efficient PointNet. In both training and testing, there is some overhead in data preprocessing to build a kd-tree, generate rays, and perform the nearest-neighbor queries to form the RaySense signature. For point clouds of around $N = 1024$, these costs are not too onerous in practice as shown in table 5.

The convolution layers have $48c + 840016$ parameters, where $c$ is the dimension of input feature space. The fully-connected layers have $64K + 278528$ parameters, where $K$ is the number of output classes. In total, our network has $1.1 \times 10^6 + 48c + 64K \approx 1.1M$ parameters. In comparison, PointNet [18] contains 3.5M parameters.
Table 4: Outliers sampled uniformly from the unit sphere are introduced during testing. The networks are trained without any outliers. Results averaged over 5 runs.

|                      | no outliers | 5 outliers | 10 outliers |
|----------------------|-------------|------------|-------------|
| RayNN-cp1            | 93.26%      | 79.76%     | 53.94%      |
| RayNN-cp5            | 93.85%      | 92.66%     | 90.90%      |
| PointNet.pytorch     | 91.08%      | 48.57%     | 25.55%      |

Table 5: Top: storage and timings for RayNN-cp1 and PointNet.pytorch on ModelNet40 using one Nvidia 1080-Ti GPU and batch size 32. The preprocessing and forward time are both measured per batch. Bottom: data from [22] is included only for reference; no proper basis for direct comparison.

|                      | Model size | Forward time | Preprocessing time | Time per epoch |
|----------------------|------------|--------------|--------------------|----------------|
| PointNet.pytorch     | 14 MB      | 12 ms        | 3.6 ms             | 14 s           |
| RayNN-cp1            | 4.5 MB     | 2 ms         | 7.5 ms             | 22 s           |
| RayNN-cp5            | 40 MB      | 16.6 ms      | -                  | -              |
| PointNet [18]        | 94 MB      | 117 ms       | -                  | -              |
| PCNN [15]            | 21 MB      | 27.2 ms      | -                  | -              |
| DGCNN [22]           | 40 MB      | 16.6 ms      | -                  | -              |

5 Summary

RaySense is a sampling technique based on projecting random rays onto a data set. The projection is done by finding nearest neighbors in the data for points on the rays. These nearest neighbors form the basic “RaySense signature”, which can be used for data processing tasks.

RaySense does not produce merely narrowly interpreted subsamples of given data sets. Salient features of the data set, such as corners or edges, are sampled with higher probability. Points near salient features may be recorded in the signature tensor multiple times.

From the RaySense signature, one can further extract snapshots of integral or local (differential) information about the data set. Relevant operations are defined on the rays randomly sampled from a chosen distribution. Since rays are one dimensional objects, the formal complexity of RaySense does not increase exponentially with the dimensions of the embedding space. We have shown theoretically that the statistics of a sampled point cloud depends only on the distribution of the rays, but not on a particular ray set. We also showed that with the appropriate post-processing of the RaySense signature tensor obtained from a given point cloud, one may compute approximations of line integrals. Thus and by way of the Fourier Slice Theorem, we argue that RaySense provides spectral information of the point cloud being sampled.

We showed that RaySense signatures could be used to register and classify point clouds of different cardinality. For classification of point clouds in three
dimensions, we presented a neural network classifier called “RayNN” which takes the RaySense signatures as input. Nearest-neighbor information can be sensitive to outliers. For finite point sets, we advocated augmentation of the signature tensor by including multiple nearest neighbors enhances RaySense’s capability to capture persistent features in the data set, thereby improving robustness. We compared its performance to several other prominent models. RayNN is lightweight, flexible, efficient, and different from conventional models; for the same data set, one can test multiple times with different ray sets.

Acknowledgments We acknowledge Albert Chern for interesting discussions. Part of this research was performed while the second author was visiting the Institute for Pure and Applied Mathematics (IPAM), which is supported by the National Science Foundation (Grant No. DMS-1440415). This work was partially supported by a grant from the Simons Foundation, NSF Grants DMS-1720171 and DMS-2110895, and a Discovery Grant from Natural Sciences and Engineering Research Council of Canada. The authors thank Texas Advanced Computing Center (TACC) and UBC Math Dept Cloud Computing for providing computing resources.

References

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

[2] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d shapenets: A deep representation for volumetric shapes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1912–1920 (2015)

[3] Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM (1975). https://doi.org/10.1145/361002.361007

[4] Tsai, Y.-H.R.: Rapid and accurate computation of the distance function using grids. J. Comput. Phys. (2002)

[5] Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with GPUs. IEEE Transactions on Big Data 7(3), 535–547 (2019)

[6] Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pp. 604–613 (1998)
[7] Jones, P.W., Osipov, A., Rokhlin, V.: Randomized approximate nearest neighbors algorithm. Proc. Natl. Acad. Sci. (2011). https://doi.org/10.1073/pnas.1107769108

[8] Hadwiger, H.: Vorlesungen ¨Uber Inhalt, Oberﬂ¨ache und Isoperimetrie vol. 93. Springer-Verlag, Berlin (2013)

[9] Natterer, F.: The Mathematics of Computerized Tomography. Society for Industrial and Applied Mathematics, USA (2001). https://doi.org/10.1137/1.9780898719284

[10] Kazmi, I.K., You, L., Zhang, J.J.: A survey of 2D and 3D shape descriptors. In: 2013 10th International Conference Computer Graphics, Imaging and Visualization, pp. 1–10 (2013). IEEE

[11] Fang, Y., Xie, J., Dai, G., Wang, M., Zhu, F., Xu, T., Wong, E.: 3D deep shape descriptor. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2319–2328 (2015)

[12] Rostami, R., Bashiri, F.S., Rostami, B., Yu, Z.: A survey on data-driven 3D shape descriptors. Comput. Graph. Forum 38(1), 356–393 (2019)

[13] Sinha, A., Bai, J., Ramani, K.: Deep learning 3d shape surfaces using geometry images. In: European Conference on Computer Vision (2016). Springer

[14] Xie, J., Dai, G., Zhu, F., Wong, E.K., Fang, Y.: Deepshape: Deep-learned shape descriptor for 3d shape retrieval. IEEE transactions on pattern analysis and machine intelligence 39(7), 1335–1345 (2016)

[15] Atzmon, M., Maron, H., Lipman, Y.: Point convolutional neural networks by extension operators. arXiv:1803.10091 (2018)

[16] Klokov, R., Lempitsky, V.: Escape from cells: Deep KD-networks for the recognition of 3D point cloud models. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 863–872 (2017)

[17] Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B.: Pointcnn: Convolution on x-transformed points. In: Advances in Neural Information Processing Systems, pp. 820–830 (2018)

[18] Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2017)

[19] Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: Advances in Neural
Information Processing Systems, pp. 5099–5108 (2017)

[20] Simonovsky, M., Komodakis, N.: Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3693–3702 (2017)

[21] Wang, P.-S., Liu, Y., Guo, Y.-X., Sun, C.-Y., Tong, X.: O-cnn: Octree-based convolutional neural networks for 3d shape analysis. ACM Transactions on Graphics (TOG) 36(4), 72 (2017)

[22] Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph cnn for learning on point clouds. ACM Transactions on Graphics (TOG) 38(5), 146 (2019)

[23] Zhou, Y., Tuzel, O.: Voxelnet: End-to-end learning for point cloud based 3d object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4490–4499 (2018)

[24] Shen, Y., Feng, C., Yang, Y., Tian, D.: Mining point cloud local structures by kernel correlation and graph pooling. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)

[25] Li, Chen, Lee, H.: SO-Net: self-organizing network for point cloud analysis. In: CVPR (2018)

[26] Ruuth, S.J., Merriman, B.: A simple embedding method for solving partial differential equations on surfaces. Journal of Computational Physics 227(3), 1943–1961 (2008)

[27] Macdonald, C.B., Merriman, B., Ruuth, S.J.: Simple computation of reaction–diffusion processes on point clouds. Proc. Natl. Acad. Sci. 110(23), 9209–9214 (2013)

[28] Krig, S.: Interest point detector and feature descriptor survey. In: Computer Vision Metrics, pp. 187–246. Springer, Cham, Switzerland (2016)

[29] Graham, R., Oberman, A.M.: Approximate convex hulls: sketching the convex hull using curvature. arXiv:1703.01350 (2017)

[30] Osting, B., Wang, D., Xu, Y., Zosso, D.: Consistency of archetypal analysis. SIAM Journal on Mathematics of Data Science 3(1), 1–30 (2021). https://doi.org/10.1137/20M1331792

[31] Hadwiger, M., Sigg, C., Scharsach, H., Bühler, K., Gross, M.: Real-time ray-casting and advanced shading of discrete isosurfaces. Computer Graphics Forum (2005). https://doi.org/10.1111/j.1467-8659.2005.00855.x
[32] Kruger, J., Westermann, R.: Acceleration techniques for gpu-based volume rendering. In: IEEE Visualization VIS 2003. (2003). https://doi.org/10.1109/VISUAL.2003.1250384

[33] Peddie, J.: Ray Tracing: A Tool for All. Springer, Cham, Switzerland (2019). https://doi.org/10.1007/978-3-030-17490-3

[34] LeCun, Y.: The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/ (1998)

[35] Settles, B.: Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences (2009)

[36] Pak, I.: Lectures on discrete and polyhedral geometry. Manuscript https://www.math.ucla.edu/~pak/book (2010)

[37] Trefethen, L.N., Weideman, J.A.C.: The exponentially convergent trapezoidal rule. SIAM Rev. 56(3), 385–458 (2014)

[38] Macdonald, C.B., Miller, M., Vong, A., et al.: The Octave Symbolic package (version 3.0.1). https://gnu-octave.github.io/packages/symbolic (2022)

[39] Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A.: SymPy: symbolic computing in Python. PeerJ Computer Science 3 (2017). https://doi.org/10.7717/peerj-cs.103

[40] Johansson, F., et al.: Mpmath: a Python Library for Arbitrary-precision Floating-point Arithmetic (version 1.2.1). (2021). https://mpmath.org/

[41] Solmon, D.C.: The X-ray transform. Journal of Mathematical Analysis and Applications 56(1) (1976). https://doi.org/10.1016/0022-247X(76)90008-1

[42] Draug, C., Gimpel, H., Kalma, A., et al.: The Octave Image package (version 2.14.0). https://gnu-octave.github.io/packages/image (2022)

[43] Besl, P.J., McKay, N.D.: Method for registration of 3-D shapes. In: Sensor Fusion IV: Control Paradigms and Data Structures, vol. 1611 (1992)

[44] Curless, B., Levoy, M.: A volumetric method for building complex models from range images computer graphics. In: SIGGRAPH 1996 Proceedings (1996). http://www-graphics.stanford.edu/data/3Dscanrep
[45] Fei Xia and others: PointNet.pytorch Git repository. https://github.com/fxia22/pointnet.pytorch. https://github.com/fxia22/pointnet.pytorch

[46] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NIPS Autodiff Workshop (2017)

[47] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167 (2015)

[48] Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp. 807–814 (2010)

[49] Sedaghat, N., Zolfaghari, M., Amiri, E., Brox, T.: Orientation-boosted voxel nets for 3d object recognition. arXiv:1604.03351 (2016)

[50] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv:1412.6980 (2014)

[51] Dvoretzky, A., Kiefer, J., Wolfowitz, J.: Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator. The Annals of Mathematical Statistics, 642–669 (1956)

[52] Massart, P.: The tight constant in the Dvoretzky-Kiefer-Wolfowitz inequality. The annals of Probability, 1269–1283 (1990)

[53] Doerr, B.: Probabilistic tools for the analysis of randomized optimization heuristics. In: Theory of Evolutionary Computation, pp. 1–87. Springer, Cham, Switzerland (2020)

[54] Corless, R.M., Gonnet, G.H., Hare, D.E., Jeffrey, D.J., Knuth, D.E.: On the LambertW function. Advances in Computational mathematics 5(1), 329–359 (1996)

[55] De Bruijn, N.G.: Asymptotic Methods in Analysis vol. 4. Courier Corporation, USA (1981)
A Examples of ray distributions

We assume all points are properly calibrated by a common preprocessing step. This could also be learned. In fact, one can use RaySense to train such a preprocessor to register the dataset, for example, using Sec. 4.4 or similar. However, for simplicity, in our experiments, we generally normalize each point set to be in the unit $\ell^2$ ball, with center of mass at the origin, unless otherwise indicated.

We present two ways to generate random rays. There is no right way to generate rays, although it is conceivable that one may find optimal ray distributions for specific applications.

**Method R1** One simple approach is generating rays of fixed-length $L$, whose direction $\vec{v}$ is uniformly sampled from the unit sphere. We add a shift $\vec{b}$ sampled uniformly from $[-\frac{1}{2}, \frac{1}{2}]^d$ to avoid a bias for the origin. The $n_r$ sample points are distributed evenly along the ray:

$$\vec{r}_i = \vec{b} + L \left( \frac{i}{n_r - 1} - \frac{1}{2} \right) \vec{v}, \quad i = 0, \ldots, n_r - 1$$

The spacing between adjacent points on each ray is denoted by $\delta r$, which is $L/(n_r - 1)$. We use $L = 2$.

**Method R2** Another natural way to generate random rays is by random endpoints selection: choose two random points $\vec{p}, \vec{q}$ on a sphere and connect them to form a ray. Then we evenly sample $n_r$ points between $\vec{p}, \vec{q}$ on the ray. To avoid overly short rays where information would be redundant, we use a minimum ray-length threshold $\tau$ to discard rays. Note that the distance between $n_r$ sample points are different on different rays:

$$\vec{r}_i = \vec{p} + \frac{i}{n_r - 1} (\vec{q} - \vec{p}), \quad i = 0, \ldots, n_r - 1.$$  

The spacing of points on each ray varies, depending on the length of the ray.

Fig. 13 shows the density of rays from the ray generation methods. In this paper, we use Method R1; a fixed $\delta r$ seems to help maintain spatial consistency along the rays, which increases RayNN’s classification accuracy in Sec. 4.5.
B Implementation details of RayNN

Our implementation uses PyTorch [46].

**Architecture** RayNN takes the $m \times k \times c$ RaySense signature tensor $S(\Gamma)$ as input, and outputs a $K$-vector of probabilities, where $K$ is the number of object classes.

The first few layers of the network are blocks of 1D convolution followed by max-pooling to encode the signature into a single vector per ray. Convolution and max-pooling are applied along the ray. After this downsizing, we implement a max operation across rays. Fig. 14 includes some details. The output of the max pooling layer is fed into fully-connected layers with output sizes 256, 64, and $K$ to produce the desired vector of probabilities $\vec{p}_i \in \mathbb{R}^K$. Batchnorm [47] along with ReLU [48] are used for every fully-connected and convolution layer.

Note that our network uses convolution along rays to capture local information while the fully-connected layers aggregate global information. Between the two, the max operation across rays ensures invariance to the ordering of the rays. It also allows for an arbitrary number of rays to be used during inference. These invariance properties are similar to PointNet’s input-order invariance [18].

![Fig. 14: The RayNN architecture for $m$ rays and $n_r$ samples per ray. The input is $c$ feature matrices from $S(\Gamma)$. With $n_r = 16$, each matrix is downsized to an $m$-vector by 4 layers of 1-D convolution and max-pooling. The max operator is then applied to each of the 1024 $m$-vectors. The length-1024 feature vector is fed into a multi-layer perceptron (mlp) which outputs a vector of probabilities, one for each of the $K$ classes in the classification task. Note the number of intermediate layers (blue) can be increased based on $n_r$ and $c$.](image)

**Data** We apply RayNN on the standard ModelNet10 and ModelNet40 benchmarks [2] for 3D object classification. ModelNet40 consists of 12311 orientation-aligned [49] meshed 3D CAD models, divided into 9843 training and 2468 test objects. ModelNet10 contains 3991 training and 908 test objects. Following the experiment setup in [18], we sample $N = 1024$ points from each
of these models and rescale them to be bounded by the unit sphere to form point sets. Our results do not appear to be sensitive to \( N \).

**Training** During training, we use dropout with ratio 0.5 on the penultimate fully-connected layer. We also augment our training dataset on-the-fly by adding \( \mathcal{N}(0,0.0004) \) noise to the coordinates. For the optimizer, we use Adam [50] with momentum 0.9 and batch size 16. The learning rate starts at 0.002 and is halved every 100 epochs.

**Inference** Our algorithm uses random rays, so it is natural to consider strategies to reduce the variance in the prediction. We consider one simple approach during inference by making an ensemble of predictions from \( \lambda \) different ray sets. The ensemble prediction is based on the average over the \( \lambda \) different probability vectors \( \vec{p}_i \in \mathbb{R}^K \), i.e.,

\[
\text{Prediction}(\lambda) = \frac{1}{\lambda} \sum_{i=1}^{\lambda} \vec{p}_i.
\]

The assigned label then corresponds to the entry with the largest probability. We denote the number of rays used during training by \( m \), while the number of rays used for inference is \( \hat{m} \). Unless otherwise specified, we use \( \lambda = 8 \), \( m = 32 \) rays, and \( \hat{m} = m \).

### C Details of the proof of Theorem 6

This appendix contains the proofs of Lemmas 3, 4, and 5.

**Proof of Lemma 3.** The probability measure of \( \Omega_j \) is

\[
P_{\Omega_j} = \int_{x \in \Omega_j} \rho(x) dx > 0,
\]

which represents the probability of sampling \( \Omega_j \) when drawing i.i.d. random samples from \( \mu \). For a fixed set of such hypercubes, any \( x \in \text{supp}(\rho) \) will fall in one of the \( \Omega_j \)'s. Then one can define a mapping \( h : \text{supp}(\rho) \subset \mathbb{R}^d \rightarrow \mathbb{R} \) by:

\[
s = h(x) = j - 1 \quad \text{where} \ x \in \Omega_j, \ j = 1, 2, \ldots, M
\]

By applying the mapping to \( X \), we obtain a new discrete random variable \( S \) with the discrete probability distribution \( \mu_M \) on \( \mathbb{R} \) and the corresponding density \( \rho_M \). The random variable \( S \) lives in a discrete space: \( S \in \{0, 1, \ldots, M - 1\} \) and \( \rho_M \) is given as a sum of delta spikes as

\[
\rho_M(s) = \sum_{j=1}^{M} P_{\Omega_j} \delta_j(s)
\]

As a result, sampling from the distribution \( \mu_M \) is equivalent to sampling the hypercubes according to the distribution \( \mu \) in \( \mathbb{R}^d \), but one cares only about the sample.

---

\(^1\)RaySense does not require point clouds for inputs: we could apply RaySense directly to surface meshes, implicit surfaces, or even—given an fast nearest neighbor calculator—the CAD models directly.
being in a specific hypercube $\Omega_j$, not the precise location of the sample. Let $F_M(s)$ denote the cumulative density function related to the density function $\rho_M(s)$.

Now, given a set of $N$ independent samples of $X: \{x_i\}_{i=1}^N \subset \mathbb{R}^d$, we have a corresponding set of $N$ independent sample points of $S: \{s_i\}_{i=1}^N$ such that $x_i \in \Omega_{s_i+1}$. From there, we can regard the histogram of $\{s_i\}_{i=1}^N$ as an empirical density of the true density $\rho_M$. Denote the empirical density by $\tilde{\rho}_M^N$ which is given by

$$\tilde{\rho}_M^N = \frac{1}{N} \sum_{i=1}^{N} \delta_{s_i}$$

one can therefore also obtain an empirical cumulative density function $\tilde{F}_M^N(s)$ using the indicator function $\chi$:

$$\tilde{F}_M^N(s) = \frac{1}{N} \sum_{i=1}^{N} \chi_{\{s_i \leq s\}}$$

By Dvoretzky–Kiefer–Wolfowitz inequality [51, 52] we have

\[ \text{Prob}\left( \sup_{s \in \mathbb{R}} |F_M(s) - \tilde{F}_M^N(s)| > \varepsilon \right) \leq 2e^{-2N\varepsilon^2} \quad \text{for all } \varepsilon > 0 \]

Therefore, for a desired fixed probability $p_0$, the above indicates the approximating error given by the empirical $\tilde{F}_M^N(s)$ is at most

$$\sup_{s \in \mathbb{R}} |F_M(s) - \tilde{F}_M^N(s)| \leq \varepsilon_N = \left( -\frac{1}{2N} \ln \left( \frac{1-p_0}{2} \right) \right)^{\frac{1}{2}}$$

with probability at least $p_0$. Then note that the true probability measure $\mu_{\Omega_j}$ of $\Omega_j$ being sampled by random drawings from $\mu$ is equivalent to the true probability of $j-1$ being drawn from $\mu_M$, i.e.

$$\mu_{\Omega_j} = \mu_M(j-1) := \rho_M(j-1)$$

therefore $\mu_{\Omega_j} = \mu_M(j-1)$ can be computed from $F_M(x)$ by:

$$\mu_{\Omega_j} = F_M(j-1) - F_M(j-2)$$

$$= F_M(j-1) - \tilde{F}_M^N(j-1) + \tilde{F}_M^N(j-1) - \tilde{F}_M^N(j-2) + \tilde{F}_M^N(j-2) - F_M(j-2)$$

Taking absolute value and using the triangle inequality, with the fixed $p_0$

$$\mu_{\Omega_j} \leq 2\varepsilon_N + \tilde{F}_M^N(j-1)$$

where $\tilde{F}_M^N(j-1)$ denotes the empirical probability at $j-1$. Apply the same argument to $\tilde{F}_M^N(j-1)$ one has

$$|\mu_{\Omega_j} - \tilde{F}_M^N(j-1)| \leq 2\varepsilon_N \quad \text{for all } j = 1, 2, \ldots, M$$

For a set of $N$ sample points, $\tilde{F}_M^N(j-1)$ is computed by $\frac{N_j}{N}$, where $N_j$ is the number of times $j-1$ got sampled by $\{s_i\}_{i=1}^N$, or equivalently $\Omega_j$ got sampled by $\{x_i\}_{i=1}^N$, which indicates that in practice, with probability at least $p_0$, the number of sampling points $N_j$ in $\Omega_j$ satisfies the following bound:

$$\mu_{\Omega_j} - 2\varepsilon_N \leq \frac{1}{N} N_j \leq \mu_{\Omega_j} + 2\varepsilon_N \quad \Rightarrow \quad \mu_{\Omega_j} - 2\varepsilon_N N \leq N_j \leq \mu_{\Omega_j} N + 2\varepsilon_N N$$

By taking $N$ large enough such that $\mu_{\Omega_j} N - 2\varepsilon_N N = 1 \Rightarrow N_j \geq 1$:

$$\Rightarrow N = \frac{\sqrt{\ln \left( \frac{1-p_0}{2} \right) \left( \ln \left( \frac{1-p_0}{2} \right) - 2\mu_{\Omega_j} \right)} + \mu_{\Omega_j} - \ln \left( \frac{1-p_0}{2} \right)}{P_{\Omega_j}^2}$$
The above quantity is clearly a function with respect to the probability measure
\( P_{\Omega i} \), and any \( \Omega i \) with \( P_{\Omega j} \geq P_{\Omega i} \) would have \( N_i \geq N_j \geq 1 \). Using \( \nu \) to denote such a function and \( 0 < P \leq 1 \) as the threshold measure completes the first part of the proof:

\[
\nu(P) = \frac{\sqrt{\ln\left(\frac{2}{1-P_0}\right)\ln\left(\frac{2}{1-P_0} + 2P\right)} + P + \ln\left(\frac{2}{1-P_0}\right)}{P^2}.
\]

To establish the bounds on the expression, we note

\[
\nu(P) > \frac{\sqrt{\ln\left(\frac{2}{1-P_0}\right)\ln\left(\frac{2}{1-P_0} + 2P\right)} + P + \ln\left(\frac{2}{1-P_0}\right)}{P^2} = \frac{2\ln\left(\frac{2}{1-P_0}\right) + P}{P^2},
\]

\[
\nu(P) < \frac{\sqrt{\ln\left(\frac{2}{1-P_0}\right) + 2P}^2 + P + \ln\left(\frac{2}{1-P_0}\right)}{P^2} = \frac{2\ln\left(\frac{2}{1-P_0}\right) + 3P}{P^2}.
\]

Proof of Lemma 4. Consider a local hypercube centered at \( y \), \( \Omega y := \{x + y \in \mathbb{R}^d : ||x||_\infty = l/2\} \) of length \( l \) to be determined. We shall just say “cube.” The probability of cube \( \Omega y \) being sampled is given by \( P_{\Omega y} = \int_{\Omega y} \rho(x) dx \). Now for the set of standard basis vector \( \{e_i\}_{i=1}^d \), let \( v_d \) denotes the sum of all the basis: \( v_d := \sum_{i=1}^d e_i \). Without loss of generality, the probability of a diagonal cube, defined by \( \Omega y_d := \{x + y + v_d \in \mathbb{R}^d : ||x||_\infty = l/2\} \), being sampled (unconditional to \( \Omega y \) being sampled) has the following bound by Lipschitz continuity of \( \rho \):

\[
|P_{\Omega y_d} - P_{\Omega y}| \leq \int_{\Omega y} |\rho(x + lv_d) - \rho(x)| dx \leq L\sqrt{d}l|\Omega y| \implies P_{\Omega y_d} \geq P_{\Omega y} - L\sqrt{d}l^{d+1}.
\]

Furthermore, \( P_{\Omega y} \) has the following lower bound also by Lipschitz continuity of \( \rho \). For any \( x \in \Omega y \), we have:

\[
|\rho(x) - \rho(y)| \leq L\sqrt{d}l^2 \implies \rho(x) \geq \rho(y) - L\sqrt{d}l \implies P_{\Omega y} \geq (\rho(y) - L\sqrt{d}l)|\Omega y| \quad (11)
\]

Combining with the previous bound for \( P_{\Omega y_d} \), we further have:

\[
P_{\Omega y_d} \geq (\rho(y) - L\sqrt{d}l)|\Omega y| - L\sqrt{d}l^{d+1} = \rho(y)|\Omega y| - \frac{3\sqrt{d}}{2}Ll^{d+1}.
\]

By setting \( \rho(y) > \frac{3\sqrt{d}}{2}Ll \) we can ensure \( P_{\Omega y_d} > 0 \), but this extreme lower bound is based on on Lipschitz continuity. To obtain a more useful bound, we will show below that by picking \( l := l_N \) judiciously, \( \rho(y) > 3\sqrt{d}Ll_N \), any surrounding cube has non-zero probability to be sampled. Therefore, with \( \rho(y) > 3\sqrt{d}Ll_N \), for any diagonal cube \( \Omega y_d \):

\[
\rho(y)|\Omega y| - \frac{1}{2}\rho(y)|\Omega y| > \frac{3\sqrt{d}}{2}Ll^{d+1} \implies P_{\Omega y_d} > \frac{1}{2}\rho(y)|\Omega y|.
\]

Since the diagonal cube is the furthest to \( y \) among all the surrounding cubes, we have for every surrounding cube of \( \Omega y \), their probability measure is at least \( P_{\Omega y_d} \).

According to Lemma 3, for \( N \) sampling points, with probability at least \( p_0 \), if a region has probability measure \( \geq P_N \), then there is at least one point sampled in that region, where \( P_N \) is the threshold probability depending on \( N \) obtained by solving the equation below:

\[
N = \frac{\sqrt{\ln\left(\frac{2}{1-(P_0)^2}\right)\ln\left(\frac{2}{1-(P_0)^2} + 2P_N\right)}}{(P_N)^2}.
\]
By the bounds for $N$ in (9) of Lemma 3, we know there is some constant $c \in (1, 3)$ s.t.:

$$ N = \frac{2 \ln \left( \frac{2}{1-p_0} \right) + c P_N}{P_N^2} \implies NP_N^2 - c P_N - 2 \ln \left( \frac{2}{1-p_0} \right) = 0 $$

Solving the above quadratic equation and realize that $P_N > 0$, we have

$$ P_N = \frac{c + \left( c^2 + 8 N \ln \left( \frac{2}{1-p_0} \right) \right)^{1/2}}{2 N} $$

Therefore, for a fixed $N$, by requiring

$$ P_{\Omega_{y,d}} > \frac{\rho(y) l_N^d}{2} \geq P_N \implies l_N \geq \left( \frac{c + \left( c^2 + 8 N \ln \left( \frac{2}{1-p_0} \right) \right)^{1/2}}{\rho(y) N} \right)^{\frac{1}{d}} $$

we have with probability $p_0$ that at every surrounding cube of $\Omega_y$ of side length $l_N$, there is at least one point. This lower bound for $l_N$ ensures the surrounding cube has enough probability measure to be sampled. Since $1 < c < 3$, we can just take $l_N$ to be:

$$ l_N := \left( \frac{3 + \left( 9 + 8 N \ln \left( \frac{2}{1-p_0} \right) \right)^{1/2}}{\rho(y) N} \right)^{\frac{1}{d}} > \left( \frac{c + \left( c^2 + 8 N \ln \left( \frac{2}{1-p_0} \right) \right)^{1/2}}{\rho(y) N} \right)^{\frac{1}{d}} $$

From above we see that for a fixed $\rho(y)$, $l_N$ decreases as $N$ increases. Therefore, by choosing $N$ large enough, we can always satisfy the prescribed assumption $\rho(y) \geq 3\sqrt{d} l_N$.

Furthermore, when $N$ is so large such that $\rho(y) \geq 3\sqrt{d} l_N$ is always satisfied, we see that $l_N$ is a decreasing function of $\rho$, meaning that with a higher local density $\rho(y)$, the $l_N$ can be taken smaller while the sampling statement still holds, meaning the local region is more compact.

Finally, since there is a point in every surrounding cube of $\Omega_y$, the diameter of the Voronoi cell of $y$ has the following upper-bound with the desired probability $p_0$:

$$ \text{diam}(V(y)) \leq 3l \sqrt{d} = 3\sqrt{d} \left( \frac{3 + \left( 9 + 8 N \ln \left( \frac{2}{1-p_0} \right) \right)^{1/2}}{\rho(y) N} \right)^{\frac{1}{d}}. $$

Now, for a sample point $x_k$ in the interior of $\text{supp}(\rho)$, given a cover of cubes as in Lemma 3, $x_k$ must belong to one of the cubes with center also denoted by $y$ with a slight abuse of notation. Then note that the diameter of $V(x_k)$ also has the same upper bound as shown above. To go from $\rho(y)$ to $\rho(x_k)$, by Lipschitz continuity:

$$ \rho(y) \geq \rho(x_k) - \frac{L \sqrt{d} l_N}{2} \implies \rho(x_k) \leq \rho(y) + \frac{L \sqrt{d} l_N}{2}. $$

Since we require $\rho(y) \geq 3\sqrt{d} l_N$, we have $\rho(x_k) \leq \rho(y) + \frac{L \sqrt{d} l_N}{2}$. Therefore:

$$ \rho(y) \geq \frac{6}{7} \rho(x_k) \implies \text{diam}(V(x_k)) \leq 3\sqrt{d} \left( \frac{21 + 7 \left( 9 + 8 N \ln \left( \frac{2}{1-p_0} \right) \right)^{1/2}}{6 \rho(x) N} \right)^{\frac{1}{d}}. $$

\[\square\]

Proof of Lemma 5. Without loss of generality, we assume that $|\text{supp}(\rho)| = 1$, then $\rho = 1$ everywhere within its support. We partition $\text{supp}(\rho)$ into $M$ regions such that each region has probability measure $\frac{1}{M}$. This partition can be constructed in the following way: for most of the interior of $\text{supp}(\rho)$, subdivide into hypercubes $\Omega_y$'s of the same size such that $P_{\Omega_y} = \frac{1}{M}$ and $\Omega_y$'s are contained completely inside $\text{supp}(\rho)$.

Then the length of the hypercube, $l$, is determined by $l^d/|\text{supp}(\rho)| = 1/M \implies \boxed{\textit{Some text here}}.$
$l = (1/M)^{1/d}$. For the remaining uncovered regions of $\text{supp}(\rho)$, cover with some small cubes of appropriate sizes and combine them together to obtain a region with measure $\frac{1}{M}$.

Then, following a similar idea from Lemma 4, one has a discrete sampling problem with equal probability for each candidate, which resembles the coupon collector problem. The probability $p(N, d, M)$ that each of the $M$ region contains at least one sample point has a well-known lower bound $[53]$:

$$p(N, d, M) \geq 1 - Me^{-\frac{N}{M}}$$

With the probability $p(N, d, M)$ given above, for an interior hypercube we again have there is at least one sample in each of its surrounding hypercube, since now there is at least one sample in each of the $M$ region. Then the Voronoi diameter for each point is at most $3l\sqrt{d}$. Fixing a desired probability $p_0$, we want to determine the number of regions $M$ to get a control on $l$. We need to have a bound as follows:

$$p \geq 1 - Me^{-\frac{N}{M}} \geq p_0 \implies 0 < Me^{-\frac{N}{M}} \leq 1 - p_0$$

(12)

By rearranging, the above equality holds only when

$$\frac{N}{M} e^{\frac{N}{M}} = \frac{N}{1 - p_0}$$

The above equation is solvable by using the Lambert $W$ function:

$$M = \frac{N}{W(\frac{N}{1 - p_0})}$$

where $W_0$ is the principal branch of the Lambert $W$ function. Note that the Lambert $W$ function satisfies

$$W_0(x)e^{W_0(x)} = x \implies \frac{x}{W_0(x)} = e^{W_0(x)}$$

Plug in the above identity, one has

$$\frac{M}{1 - p_0} = \frac{N}{(1 - p_0)W_0(\frac{N}{1 - p_0})} \implies M = (1 - p_0)e^{W_0(\frac{N}{1 - p_0})}$$

Also note that the function $Me^{-\frac{N}{M}}$ is monotonically increasing for $M > 0$, so for the bound in (12) to hold we require:

$$M \leq (1 - p_0)e^{W_0(\frac{N}{1 - p_0})}$$

By taking the largest possible integer $M$ satisfying the above inequality, we then have

$$l = (1/M)^{1/d} = ((1 - p_0)e^{W_0(\frac{N}{1 - p_0})})^{-1/d}$$

for every hypercube contained in $\text{supp}(\rho)$. Then this yields a uniform bound for the Voronoi diameter of any point that is in an interior hypercube surrounded by other interior hypercubes:

$$\text{diam}(V) \leq 3\sqrt{d}((1 - p_0)e^{W_0(\frac{N}{1 - p_0})})^{-\frac{1}{d}}$$

In terms of the limiting behavior, for large $x$, the Lambda $W$ function is asymptotic to the following [54, 55]:

$$W_0(x) = \ln x - \ln\ln x + o(1) \implies e^{W_0(x)} = c(x)\frac{x}{\ln x}$$

with $c(x) \to 1$ as $x \to \infty$. Therefore, for sufficiently large $N/(1 - p_0)$, we have

$$\text{diam}(V) \leq 3\sqrt{d}\left(\left|(1 - p_0)c\left(\frac{N}{(1 - p_0)\ln\frac{N}{1 - p_0}}\right)\right|^{-\frac{1}{d}}\right) = 3\sqrt{d}\left(\left|\frac{1}{cN}\ln\frac{N}{1 - p_0}\right|\right)^{\frac{1}{2}}$$

$\square$