A Linear Time and Space Local Point Cloud Geometry Encoder via Vectorized Kernel Mixture (VecKM)

Dehao Yuan [1] **Cornelia Fermüller** [1] **Tahseen Rabbani** [1] **Furong Huang** [1] **Yiannis Aloimonos** [1]

Abstract

We propose VecKM, a local point cloud geometry encoder that is descriptive and efficient to compute. VecKM leverages a unique approach by vectorizing a kernel mixture to represent the local point cloud. Such representation's descriptiveness is supported by two theorems that validate its ability to reconstruct and preserve the similarity of the local shape. Unlike existing encoders downsampling the local point cloud, VecKM constructs the local geometry encoding using all neighboring points, producing a more descriptive encoding. Moreover, VecKM is efficient to compute and scalable to large point cloud inputs: VecKM reduces the memory cost from $(n^2 + nKd)$ to $(nd + np)$; and reduces the major runtime cost from computing nK MLPs to n MLPs, where n is the size of the point cloud, K is the neighborhood size, d is the encoding dimension, and p is a marginal factor. The efficiency is due to VecKM's unique factorizable property that eliminates the need of explicitly grouping points into neighbors. In the normal estimation task, VecKM demonstrates not only 100x faster inference speed but also highest accuracy and strongest robustness. In classification and segmentation tasks, integrating VecKM as a preprocessing module achieves consistently better performance than the PointNet, PointNet++, and point transformer baselines, and runs consistently faster by up to 10 times.

1. Introduction

The ubiquity and low cost of 3D sensors have drawn increased interest in the usage of three-dimensional point clouds for tasks such as autonomous driving (Sen et al., 2023), robotics (Zampogiannis et al., 2019), and remote sensing (Lu et al., 2020; Chen et al., 2022).

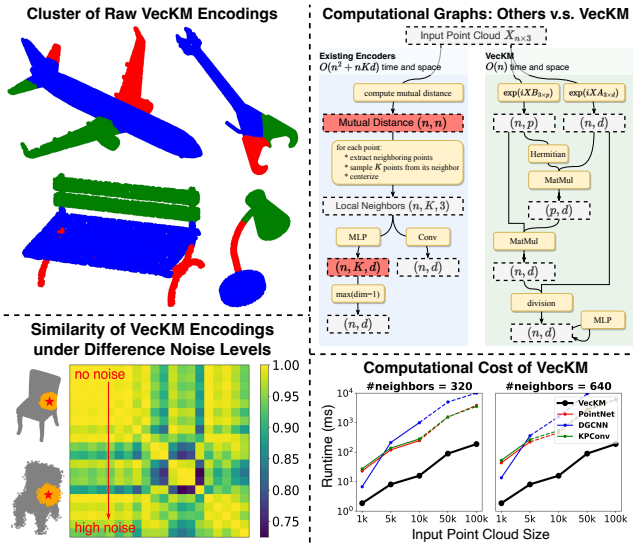

Figure 1. Our VecKM encoding is descriptive, robust to noise, and efficient in runtime and memory cost. **Upper Left**: Raw VecKM encodings, *without any training*, already capture rich geometric features such as orientations and shapes. **Lower Left**: Under varying levels of noise, VecKM encodings remain highly consistent. **Upper Right**: Existing encoders face memory costs of $(n^2 + nKd)$, while VecKM costs only $(nd + np)$ memory. Existing encoders compute nK MLPs, whereas VecKM only computes n MLPs. **Lower Right**: VecKM is 10x$\sim$100x faster than existing encoders in wall-clock time and scalable to large point cloud inputs.

In point cloud analysis, encoding local geometry is a fundamental step. In both low-level tasks such as feature matching and normal estimation, and high-level tasks such as classification, segmentation, and detection, encoding local geometry is usually required before passing the point cloud into any deep network. Much effort has been placed into the design of local geometry encoders, which can be loosely divided into two categories: hand-crafted features and learnable encoders. Hand-crafted features (Han et al., 2023) are manually defined features based on domain expertise, and learnable encoders require computationally expensive processing through trainable structures such as multi-layer perceptrons (MLP) (Qi et al., 2017a; Ma et al., 2022) or convolutions (Li et al., 2018; Thomas et al., 2019).

[1]Department of Computer Science, University of Maryland, College Park, USA. Correspondence to: Dehao Yuan <dhyuan@umd.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

These local geometry encoders follow a similar pipeline. They first group the input point cloud into neighborhoods and then process each neighborhood individually. As illustrated in Figure 1 (upper right), the pipeline involves computing the mutual distance between points. Then for each point, a number of K points are sampled from its neighborhood, and MLP or convolution are used to transform the sampled neighborhood. In this pipeline, grouping the point cloud into neighborhoods requires n^2 time and space, and the MLP-based architectures, in particular, requires a sequence of MLPs to transform nK vectors and reaches an intermediate stage of (n, K, d). The pipeline results in bottlenecks in both computation and memory. Consequently, they usually resort to downsampling the local point clouds (i.e. reducing K), which can lead to inadequate representation of the local point cloud.

In this work, we address the computation and memory bottlenecks faced by the existing encoders, reducing the memory cost from $(n^2 + nKd)$ to $(nd + np)$ and only computing n MLPs. Besides, our representation is constructed from all the neighboring points without downsampling, and hence is more descriptive. Our approach is inspired by Frady et al. (2022); Yuan et al. (2023), which converts continuous functions into fixed-length vectors. Building on this concept, we introduce VecKM, which conceptualizes local point clouds as kernel mixtures (a form of continuous function) and vectorizes them. Under this formulation, we prove the local geometry encoding is reconstructive and isometric to the local point cloud, which guarantees the descriptiveness of the representation. One essential advantage of VecKM is its factorizable property, which eliminates the need of explicitly grouping the neighborhoods and reuses many computations.

The VecKM encodings can subsequently be passed to deep cloud point models, such as PointNet++ (Qi et al., 2017b) and transformers (Vaswani et al., 2017). VecKM's light representation and ease of computation significantly speed up the inference, while still achieving on-par or improved performance than other networks in classification and segmentation tasks. Our contributions are summarized below:

- We present VecKM, a local geometry encoder that is descriptive and efficient. VecKM costs only $nd+np$ memory and computes only n MLPs. This is achieved through a novel approach of vectorizing kernel mixtures, coupled with its unique factorizability. VecKM is the only existing local geometry encoder that costs linear time and space.
- Unlike existing encoders downsampling the local point cloud, VecKM constructs the local geometry encoding using all neighboring points, and hence is more descriptive.
- We evaluate our VecKM on multiple point cloud tasks. In normal estimation, VecKM is > 100x faster and achieves $> 16\%$ lower error than other widely-used learnable encoders and demonstrates the strongest robustness against different types of data corruption. In classification and segmentation tasks, integrating VecKM as a preprocessing module achieves consistently better performance than the PointNet, PointNet++, and point transformer baselines, and runs consistently faster by up to 10 times.

2. Related Work

2.1. Local Geometry Encoder

The initial processing of raw point cloud data, an unordered collection of points, typically begins with the extraction of local geometric features. This step is essential before any further processing can occur. Existing methods for encoding local geometry can generally be categorized into two groups: hand-crafted features and learning-based encoders. All the encoders require grouping point clouds into neighborhoods.

Hand-Crafted Features are manually defined features that describe the local geometry. Domain expertise of the point cloud dataset and the task of interest are usually needed for constructing those features. We refer the reader to Han et al. (2023) for a comprehensive survey of hand-crafted features.

Histogram-based features represent a significant category within hand-crafted features, which transform the local point cloud into specific coordinate systems such as Cartesian (Prakhya et al., 2017), polar (Ge, 2016), star-shaped (Steder et al., 2010). Then the coordinate system is quantized, and the local point cloud is binned accordingly. The resulting feature is formed by concatenating the histograms. While these features induce minimal information loss, the resolution of the point cloud affects the quality of such features.

Statistics-based features form another category within hand-crafted features, which construct statistical descriptors from geometric parameters. Examples include eigenvalues (Vandapel et al., 2004), covariance (Fehr et al., 2012), normal orientation distribution (Triebel et al., 2006), angles between normals (Rusu et al., 2008), local umbrella shapes (Ran et al., 2022). These types of features can include rich geometric features and easily achieve rotation invariance. But they tend to be lossy and are often not robust to noise and variations in density.

Learning-Based Encoders transform the raw local point clouds into fixed-length vectors through trainable networks. These encoders can be broadly categorized into MLP-based encoders and convolution-based encoders.

MLP-based encoders use MLPs to transform the local point cloud and use max-pooling to cast the point cloud into a fixed-length vector. These operations can be performed repeatedly to retrieve deep point cloud features (Qi et al., 2017b). Examples of MLP-based encoders include PointNet (Qi et al., 2017a), CurveNet (Xiang et al., 2021), PointMLP (Ma et al., 2022). MLP-based encoders are usually faster

to compute than convolution-based ones. But they require computing an intermediate step of (n, K, d) to perform the max-pooling operation. So they induce high memory cost when the input size n and the neighborhood size K is large.

Convolution-based encoders use point or edge convolution to transform the local point cloud into a fixed-length vector. Examples include KPConv (Thomas et al., 2019), PointCNN (Li et al., 2018), PointConv (Wu et al., 2019), SpiderCNN (Xu et al., 2018). Convolution-based encoders are more expensive to compute, but they do not encounter the memory bottleneck faced by MLP-based encoders.

VecKM is both a hand-crafted feature and a learning-based encoder. It not only captures the geometric features, but also faithfully encodes the point distribution. This duality allows VecKM to leverage the strengths of both approaches.

2.2. Point Cloud Architectures

We describe two major families of architectures for processing point clouds: PointNet++ and transformers. VecKM, as to be shown later, is compatible with both architectures.

PointNet++ (Qi et al., 2017b) utilizes hierarchical neural layers to capture fine geometric details at multiple scales. Within each layer, PointNets are utilized to transform the features. Many works have been done to improve the architecture. Examples include using different learning-based local geometry encoders, as introduced in Section 2.1, and improving the neighborhood grouping strategies (Xiang et al., 2021; Yan et al., 2020). PointNet++ and its derivatives tend to be faster than but less accurate than transformers.

Transformers. Given their success on a wide variety of vision tasks, along with their tolerance to permutations, many transformer-based models have been proposed for 3D point cloud processing. Models such as PCT (Guo et al., 2021), 3CROSSNet (Han et al., 2022), and Point-BERT (Yu et al., 2022) apply transformer blocks to individual points to extract global information. Other models such as Point Transformer (PT) (Zhao et al., 2021), Pointformer (Pan et al., 2021), and the Stratified Transformer (Lai et al., 2022) process local patches to extract local feature information. However, transformer-based models can suffer from computational and memory bottlenecks (Han et al., 2023) as the attention map increases in size.

3. Methodology

3.1. Problem Definition and Main Theorems

Problem Definition. Let the input point cloud be $X = \{\mathbf{x}_k\}_{k=1}^n$. Denote the centerized neighbor of the point $\mathbf{x}_k$ as $\mathfrak{N}(\mathbf{x}_k) := \{\mathbf{x}_j - \mathbf{x}_k : ||\mathbf{x}_j - \mathbf{x}_k|| < r\}$. The output is the set of dense local geometric features $G = \{\mathbf{g}_k\}_{k=1}^n$, where $\mathbf{g}_k = E\big(\mathfrak{N}(\mathbf{x}_k)\big) \in \mathbb{C}^d$. We look for an encoder E that maps the local point cloud into a fixed-length vector, which "captures the underlying shape" sampled by the point cloud.

To better formalize the heuristic expression of "capturing the underlying shape", we think of the local shape around the point x_k as a distribution function $f_k : \mathbb{R}^3 \to \mathbb{R}^+$, where $f_k(\mathbf{x})$ gives the probability density that a point $\mathbf{x}$ is on the local shape. We then think of the centerized local point cloud $\mathfrak{N}(\mathbf{x}_k)$ as random samples from the distribution function f_k. We expect the local point cloud encoding $E\big(\mathfrak{N}(\mathbf{x}_k)\big) \in \mathbb{C}^d$ to represent the distribution function f_k. For a good representation, we consider two natural properties: 1. the distribution function can be reconstructed from the encoding; 2. the correlation of the distribution functions is preserved by the similarity of the encodings.

Pointwise Local Geometry Encoding. Under the problem definition, we present the formula for encoding the local geometry around a single point. Unless specified otherwise, all input points $\mathbf{x}_j$ are assumed to be three-dimensional.

Theorem 1 (Pointwise Local Geometry Encoding)**.** *Denote the neighbors of the point $\mathbf{x}_0$ as $\mathfrak{N}(\mathbf{x}_0) := \{\mathbf{x}_k - \mathbf{x}_0\}_{k=1}^n$. The local geometry encoding of $\mathbf{x}_0$ is computed as*

$$E_{\mathbf{A}}\big(\mathfrak{N}(\mathbf{x}_0)\big) = \frac{1}{n}\sum_{k=1}^{n} \exp\big(i(\mathbf{x}_k - \mathbf{x}_0)\mathbf{A}_{3\times d}\big) \qquad (1)$$

where i is the imaginary unit and $\mathbf{A} \in \mathbb{R}^{3\times d}$ is a fixed random matrix where each element follows the normal distribution $\mathcal{N}(0, \alpha^2)$. As to be shown in Section 3.2, $E_{\mathbf{A}}\big(\mathfrak{N}(\mathbf{x}_0)\big)$ is fundamentally vectorizing a kernel mixture about $\mathfrak{N}(\mathbf{x}_0)$, so we name the encoding VecKM. Next, we present two propositions that claim VecKM encoding produces a good representation of the local shape:

Proposition 1 (Reconstruction)**.** *WLOG, let f be the distribution function characterizing the local shape of $\mathbf{0}$, $X = \{\mathbf{x}_k\}_{k=1}^n$ be the random samples drawn from the distribution function f, and $\mathbf{g}_n = \frac{1}{n}\sum_{k=1}^n \exp\big(i\mathbf{x}_k\mathbf{A}\big)$ be the VecKM encoding given by Eqn. (1). $\mathbf{A} \in \mathbb{R}^{3\times d}$ is a fixed matrix whose entries are drawn from $\mathcal{N}(0, \alpha^2)$. Then at all points $\mathbf{x}$ where $f(\mathbf{x})$ is continuous, as $n \to \infty$ and $\alpha^2 \to 0$,*

$$\langle \mathbf{g}_n, \exp(i\mathbf{x}\mathbf{A})\rangle \to f(\mathbf{x})$$

where $\langle\cdot,\cdot\rangle$ denotes the inner product between two complex vectors. The proposition states that under a suitable selection of the parameter α^2, the distribution function f can be approximately reconstructed from the VecKM encoding $\mathbf{g}_n$.

Proposition 2 (Similarity Preservation)**.** *Let f_1, f_2 be two distribution functions characterizing two local shapes and X_1, X_2 be the random samples from the two distribution functions. $\mathbf{g}_1$, $\mathbf{g}_2$ are the VecKM encodings given by Eqn. (1) with X_1, X_2 as inputs. $\mathbf{A}$ is a fixed matrix whose entries are drawn from $\mathcal{N}(0, \alpha^2)$. Then the function similarity is preserved by the VecKM encodings: as $n \to \infty$ and $\alpha^2 \to 0$,*

$$\langle \mathbf{g}_1, \mathbf{g}_2\rangle \to \langle f_1, f_2\rangle = \int_{\mathbb{R}^3} f(x)g(x)dx$$

where $\langle\cdot,\cdot\rangle$ denotes the inner product between two complex vectors. The proposition states that under a suitable selection of the parameter α^2, the correlation of functions (i.e. shapes) is approximately preserved by the VecKM encoding.

In brief, Theorem 1 presents the formula for encoding the local geometry around a single point. Proposition 1, 2 assert that VecKM well represents the underlying local geometry. In Section 3.2, we will explain the mechanism behind Theorem 1 and prove Proposition 1, 2 in detail.

Dense Local Geometry Encoder With Eqn. (1), we can already compute the local geometry encoding for each point individually by grouping their neighborhoods. However, VecKM has a unique factorizable property that enables us to reuse computations and eliminate the intermediate step:

Theorem 2 (Dense Local Geometry Encoding)**.** *Denoting the input point cloud as a matrix $\mathbf{X}_{n\times 3} = [\mathbf{x}_1;\mathbf{x}_2;\cdots;\mathbf{x}_n]$, the dense local geometry encoding $\mathbf{G}_{n\times d}$ is computed by*

$$\begin{aligned}\mathcal{A}_{n\times d} &= \exp(i\mathbf{X}_{n\times 3}\mathbf{A}_{3\times d})\\ \mathcal{B}_{n\times p} &= \exp(i\mathbf{X}_{n\times 3}\mathbf{B}_{3\times p})\\ \mathbf{G}_{n\times d} &= normalize\big((\mathcal{B}\times\mathcal{B}^H\times\mathcal{A})\,./\,\mathcal{A}\big)\end{aligned} \tag{2}$$

where $\mathbf{A}$ and $\mathbf{B}$ are two random fixed matrix whose entries are drawn from $\mathcal{N}(0,\alpha^2)$ and $\mathcal{N}(0,\beta^2)$. $\times$ denotes the matrix multiplication, and $./$ denotes the elementwise division. As to be explained in Section 3.3, computing the dense local geometry encoding using Eqn. (2) has almost the same effect as computing the pointwise local geometry encoding using Eqn. (1). However, Eqn. (2) only takes $\Theta(npd)$ time and $(np+nd)$ space to compute, where p, to be shown, is a marginal factor. The computation graph is visualized in Figure 1 (upper right).

Structure of Proof. In Section 3.2, we explain the mechanism behind Theorem 1 and prove our assertion that VecKM produces a good representation of the local geometry. In Section 3.3, we explain why Eqn. (2) has almost the same effect as Eqn. (1) and the mechanism behind Theorem 2. In Section 3.4, we introduce how to incorporate VecKM encodings into deep point cloud architectures.

3.2. Pointwise Local Geometry Encoder

In this section, we introduce why VecKM (Eqn. 1) produces a good representation of the local geometry. The key idea, as illustrated in Figure 2, is that (i) VecKM vectorizes a Gaussian kernel mixture associated with the local point cloud, where (ii) the associated kernel mixture can approximate the local shape distribution function. Therefore, VecKM effectively represents the local shape. We will separately validate assertion (i) and (ii).

(i) VecKM vectorizes a kernel mixture. We first present a lemma stating that VecKM embodies a Gaussian kernel $\mathcal{G}$:

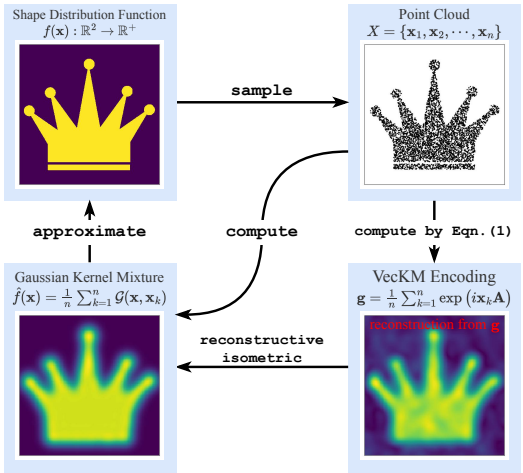

Figure 2. Theoretical outline of VecKM illustrated by 2d shapes. A point cloud, sampled from a shape distribution function, is associated with a Gaussian kernel mixture and a corresponding VecKM encoding, where the VecKM encoding is proved to be reconstructive and isometric to the Gaussian kernel mixture. Since the Gaussian kernel mixture can approximate the shape function, the VecKM encoding yields a good representation of the shape.

Lemma 1 (VecKM embodies a Gaussian kernel)**.** *Let $\mathbf{x},\mathbf{y}\in\mathbb{R}^3$, $\mathbf{A}\in\mathbb{R}^{3\times d}$. All elements in $\mathbf{A}$ are drawn from normal distribution $\mathcal{N}(0,\alpha^2)$. Then as $d\to\infty$,*

$$\frac{1}{d}\langle e^{i\mathbf{x}\mathbf{A}}, e^{i\mathbf{y}\mathbf{A}}\rangle \to \mathcal{G}_\alpha(\mathbf{x},\mathbf{y}) := \exp\Big(-\frac{\alpha^2||\mathbf{x}-\mathbf{y}||^2}{2}\Big)$$

Lemma 1 is a corollary from the Bochner's theorem (Bochner, 2005; Rahimi & Recht, 2007). We provide a detailed proof in Appendix A. Importantly, the Gaussian kernel $\mathcal{G}$ is approximated by the inner product of finite-length vectors $e^{i\mathbf{x}\mathbf{A}}$ and $e^{i\mathbf{y}\mathbf{A}}$. This approximation is important in vectorizing the kernel mixture and ensures the reconstructive and isometric properties in Proposition 1, 2, as detailed in the subsequent two lemmas. Unless otherwise specified, all entries in $\mathbf{A}$ are drawn from $\mathcal{N}(0,\alpha^2)$ and $\mathcal{G}$ means $\mathcal{G}_\alpha$. The proofs are borrowed from Frady et al. (2022).

Lemma 2 (Reconstruction)**.** *Let $\mathbf{g}=\frac{1}{n}\sum_{k=1}^{n}\exp(i\mathbf{x}_k\mathbf{A})$ be the VecKM encoding, where all entries in $\mathbf{A}\in\mathbb{R}^{3\times d}$ are drawn from $\mathcal{N}(0,\alpha^2)$. $\hat{f}(\mathbf{x}) = \frac{1}{n}\sum_{k=1}^{n}\mathcal{G}_\alpha(\mathbf{x},\mathbf{x}_k)$ be the associated Gaussian kernel mixture. Then $\langle\exp(i\mathbf{x}\mathbf{A}),\mathbf{g}\rangle\to\hat{f}(\mathbf{x})$ as $d\to\infty$.*

The lemma is derived from the linearity of the inner product:

$$\begin{aligned}\langle\exp(i\mathbf{x}\mathbf{A}),\mathbf{g}\rangle &= \frac{1}{n}\sum_{k=1}^{n}\langle\exp(i\mathbf{x}\mathbf{A}),\exp(i\mathbf{x}_k\mathbf{A})\rangle\\ &\to \frac{1}{n}\sum_{k=1}^{n}\mathcal{G}(\mathbf{x},\mathbf{x}_k) = \hat{f}(\mathbf{x})\end{aligned}$$

The lemma states that the Gaussian kernel mixture can be approximately reconstructed from the VecKM encoding $\mathbf{g}$,

which theoretically shows that VecKM is equivalent to the Gaussian kernel mixture when d is large.

Lemma 3 (Similarity Preservation)**.** *Let $\mathbf{g}_1$, $\mathbf{g}_2$ be two VecKM encodings and f_1, f_2 be their associated Gaussian kernel mixtures. Then $\langle \mathbf{g}_1, \mathbf{g}_2 \rangle \to \langle f_1, f_2 \rangle$ as $d \to \infty$.*

The lemma states that the VecKM encoding preserves the similarity/correlation between kernel mixtures, which further verifies that the encoding is not only equivalent but also isometric to the Gaussian kernel mixture. The lemma is derived from the linearity of integration:

$$\begin{aligned}\langle f_1, f_2 \rangle &= \int_{\mathbf{x}\in\mathbb{R}^3} \Big(\frac{1}{n}\sum_{p=1}^{n} \mathcal{G}(\mathbf{x}, \mathbf{x}_p)\Big)\Big(\frac{1}{m}\sum_{q=1}^{m} \mathcal{G}(\mathbf{x}, \mathbf{x}'_q)\Big) d\mathbf{x} \\ &= \frac{1}{mn}\sum_{p,q} \int_{\mathbf{x}\in\mathbb{R}^3} \mathcal{G}(\mathbf{x}, \mathbf{x}_p)\mathcal{G}(\mathbf{x}, \mathbf{x}'_q) d\mathbf{x} \\ &= \frac{1}{mn}\sum_{p,q} \mathcal{G}(\mathbf{x}_p, \mathbf{x}'_q) \leftarrow \langle \mathbf{g}_1, \mathbf{g}_2 \rangle\end{aligned}$$

Lemma 1-3 complete the argument that the VecKM encoding is equivalent and isometric to the kernel mixture when d is large. In practice, the selection of d is independent of the size of the point cloud. d as small as 256 yields good encoding in many scenarios, for example, in our experiments.

(ii) The Gaussian kernel mixture associated with the point cloud approximates the shape function. This is derived from the one-class support vector machine (SVM). The input to the one-class SVM is a collection of points and a user-defined kernel function, where the Gaussian (a.k.a. radial basis function) kernel is a common choice. The output of the one-class SVM is a kernel mixture which estimates the distribution of the input point set. Schölkopf et al. (1999) proves that with an appropriately chosen parameter α^2 (defined in Lemma 1), a Gaussian kernel mixture can approximate the distribution function. This validates the assertion that the kernel mixtures associated with VecKM can approximate the shape distribution function. Coupled with Lemma 2, 3, we prove Proposition 1, 2, which reveal that VecKM effectively represents the local geometry.

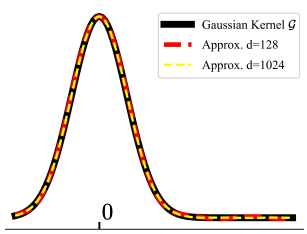

Figure 3. Visualization of Gaussian kernel $\mathcal{G}$ and its approximation with Lemma 1.

3.3. Dense Local Geometry Encoder

In the previous section, we explained why Eqn. 1 well represents the underlying local shape. In this section, we introduce the unique factorizable property that enables efficient computation of the dense local geometry encoding.

The geometry encoding in Eqn. (1) can be factorized into:

$$\begin{aligned}E_{\mathbf{A}}\big(\mathfrak{N}(\mathbf{x}_0)\big) &= \frac{1}{n}\sum_{k=1}^{n} \exp\big(i(\mathbf{x}_k - \mathbf{x}_0)\mathbf{A}_{3\times d}\big) \\ &= \frac{1}{n}\Big[\sum_{k=1}^{n} \exp(i\mathbf{x}_k\mathbf{A})\Big] \,./\, \exp(i\mathbf{x}_0\mathbf{A})\end{aligned}$$

Under this observation, we can write the dense local geometry encoding in terms of matrix computation:

$$\begin{aligned}\mathcal{A}_{n\times d} &= \exp(i\mathbf{X}_{n\times 3}\mathbf{A}_{3\times d}) \\ \mathbf{G}_{n\times d} &= [\mathbf{J}_{n\times n}\mathcal{A}_{n\times d}] \,./\, \mathcal{A}_{n\times d}\end{aligned} \tag{3}$$

$\mathbf{J}_{n\times n}$ is the adjacency matrix of the point cloud $\mathbf{X}_{n\times 3}$, where $\mathbf{J}[j,k] = 1$ if $||\mathbf{x}_j - \mathbf{x}_k|| < r$ and 0 otherwise. Under this formulation, we still require n^2 time and space to compute the adjacency matrix $\mathbf{J}$ and $(n^2 d)$ FLOPs to compute $\mathbf{G}$. But one important idea can be applied to speed up the computation: Instead of adopting a sharp threshold r to define the adjacency relation, we employ an exponential decay function to establish this relationship:

$$\hat{\mathbf{J}}[j,k] = \exp(-\beta^2||\mathbf{x}_j - \mathbf{x}_k||^2/2)$$

where $\hat{\mathbf{J}}[j,k]$ decays from 1 to 0 as $||\mathbf{x}_j - \mathbf{x}_k||$ increases and the parameter β controls the speed of decaying. As comparison, $\mathbf{J}[j,k]$ drops sharply from 1 to 0 when $||\mathbf{x}_j - \mathbf{x}_k||$ reaches $> r$. The parameter β in $\hat{\mathbf{J}}$ has the same functionality as the parameter r in $\mathbf{J}$, which is controlling the receptive field of the local neighbors. Arguably, $\mathbf{J}$ and $\hat{\mathbf{J}}$ behave similarly and it is natural to substitute $\mathbf{J}$ with $\hat{\mathbf{J}}$ in Eqn. (3). The motivation of this substitution is that $\hat{\mathbf{J}}$ can be factorized into a matrix multiplication:

$$\begin{aligned}\mathcal{B}_{n\times p} &= \exp(i\mathbf{X}_{n\times 3}\mathbf{B}_{3\times p}) \\ \hat{\mathbf{J}}_{n\times n} &\leftarrow \mathcal{B} \times \mathcal{B}^H \quad \text{as } p \to \infty\end{aligned}$$

where all entries in $\mathbf{B} \in \mathbb{R}^{3\times p}$ follow $\mathcal{N}(0, \beta^2)$. Such approximation is, again, guaranteed by Lemma 1. With such approximation, Eqn. (3) can be rewritten as

$$\begin{aligned}\mathbf{G}_{n\times d} &= [\hat{\mathbf{J}}_{n\times n}\mathcal{A}_{n\times d}] \,./\, \mathcal{A}_{n\times d} \\ &\approx [\mathcal{B}_{n\times p} \times (\mathcal{B}^H \times \mathcal{A})_{p\times d}] \,./\, \mathcal{A}_{n\times d}\end{aligned}$$

By computing $\mathcal{B}^H \times \mathcal{A}$ first, the computation cost is reduced to $\Theta(npd)$. A large point cloud size usually requires a larger p to reduce the noise, but the value p is much smaller than n. For a point cloud with size 100k, $p = 4096$ is sufficient. A large p improves the quality of the encoding, but does not increase the size of the encoding, and hence does not increase the cost of subsequent processings. Such approximation-and-factorization trick is inspired from Peng et al. (2021), which accelerates the attention computation in transformers. This concludes the proof of Theorem 2.

Effect of α and β. We perform a qualitative analysis of the effect of the parameters α and β in Theorem 2. In short, α controls the level of details and β controls the receptive field of the local neighbor. As illustrated in Figure 8, when α is larger, more high-frequency details are preserved in the encoding, and meanwhile the local geometry encodings tends to be dissimilar to each other. A larger α is usually preferred in tasks that require refined local geometry, such as normal estimation. A smaller α is usually preferred in high-level tasks, such as classification and segmentation. For β selection, we provide a table in Appendix D, which shows a matching between the neighborhood radius and the corresponding β value. More quantitative analysis will be presented in Section 5.

Effect of d and p. The parameters d and p control the quality of the encoding. Higher values lead to better quality of encoding. Figure 9 provides the qualitative analysis.

Uniqueness of VecKM. VecKM cannot be established without two important properties: 1. VecKM embodies a kernel function (Lemma 1); 2. VecKM is factorizable. Importantly, the family of exponential functions is the only family of functions that has the factorizability property with respect to multiplication and division: $f(x-y) = f(x)/f(y)$. But if we use the real exponential functions, the computation is not numerically stable, and meanwhile, the inner product between the constructed vectors will not induce a kernel, i.e. Lemma 1 will not hold. Therefore, VecKM is the only choice to enable both properties, i.e. both being factorizable and inducing a kernel function. *Therefore, we conjecture that VecKM may be the only possible linear local geometry encoder.* Fortunately, we are blessed with the advantages offered by complex vectors, which provide the necessary descriptiveness and efficiency for VecKM.

3.4. VecKM in Point Cloud Deep Learning

VecKM can seamlessly be integrated into widely-used deep point cloud architectures, including PointNet (Qi et al., 2017a), PointNet++ (Qi et al., 2017b), and transformers (Guo et al., 2021; Zhao et al., 2021). Typically, these architectures compute the dense local geometry in the first layer, often utilizing mini-PointNet or sequences of KPConvs (Thomas et al., 2019). To use VecKM in those architectures, we simply replace the dense local geometry modules with our VecKM encodings.

Note that VecKM produces complex vector outputs. To effectively utilize this in subsequent layers, we employ a series of complex linear layers and complex ReLU layers (Trabelsi et al., 2017) to process the encodings. Finally, we cast the complex vectors into real vectors by calculating the squared norm of the complex vectors, thereby making the output compatible with standard architecture requirements. Figure 4 presents several examples of integrating VecKM into deep point cloud architectures, which are capable of solving many tasks involving point cloud inputs. Appendix B gives the elegant implementation of VecKM in PyTorch.

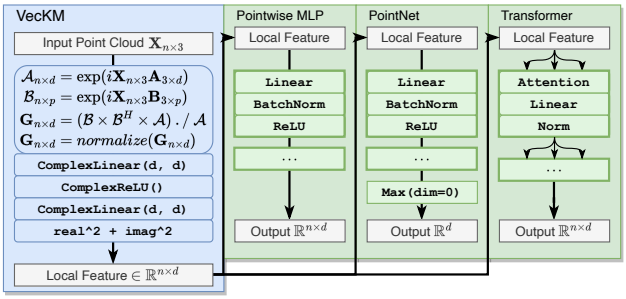

Figure 4. VecKM can be seamlessly integrated into deep point cloud architectures, improving both accuracy and efficiency.

4. Experiments

We present extensive experiments to evaluate our VecKM encoding. In Section 4.1, we present quantitative and qualitative analyses on the effectiveness, efficiency, robustness, and scalability of the proposed VecKM encoding by solving the low-level task of normal estimation. In Section 4.2-4.4, we demonstrate the effectiveness and efficiency when incorporating VecKM into deep point cloud architectures to solve high-level tasks. Depending on the input point cloud size, we use different implementations (either Eqn. (2) or Eqn. (3)) of VecKM to yield the better efficiency.

4.1. Normal Estimation on PCPNet Dataset

We compare our VecKM against other local geometry encoders in four dimensions: accuracy, computational cost, memory cost, and robustness to noise. We select local point cloud normal estimation as our evaluation task because of its inherent challenges. This task requires the geometry encoders to adequately understand the local geometry. Moreover, it presents significant challenges in terms of memory and time complexity, given the large number of points in the input and the large number of neighboring points that need to be considered. As to be shown, **VecKM outperforms other encoders in all four dimensions by large margins.**

Dataset and Metrics. We use PCPNet (Guerrero et al., 2018) as the evaluation dataset. PCPNet includes 8 shapes in the training set and 19 shapes in the test set. Each shape is sampled with 100,000 points and their ground-truth normals are derived from the original meshes. PCPNet provides two types of data corruption for testing: (1) point perturbations: adding Gaussian noise to the point coordinates. (2) point density variation: resampling the point cloud under two scenarios, where *gradient* simulates the effects of varying distances from a sensor and *strips* simulates the occlusion effect. We use the root mean squared angle error (RMSE) in degrees as the evaluation metrics.

Table 1. Normal estimation RMSE on the PCPNet dataset.

	Perturbations				Density Variation		Average
	None	Low	Med	High	Gradient	Stripe	
KPConv, #kp=16	22.68	23.09	25.21	29.05	34.40	25.61	26.67
KPConv, #kp=32	22.74	22.21	24.08	28.25	32.24	24.94	25.74
KPConv, #kp=64	22.09	22.12	23.90	28.45	28.60	24.05	24.86
DGCNN, #nbr=32	24.08	24.04	25.19	28.24	27.12	27.55	26.03
DGCNN, #nbr=64	23.21	25.34	25.66	26.01	28.86	28.20	26.21
DGCNN, #nbr=128	18.46	18.71	20.38	25.62	23.01	21.29	21.24
PointNet, #nbr=300	14.98	16.30	20.19	26.83	23.68	19.00	20.17
PointNet, #nbr=500	16.10	16.54	21.38	26.93	26.06	18.89	20.99
PointNet, #nbr=700	15.59	16.25	20.99	26.21	24.66	17.87	20.27
VecKM (Ours)	**13.59**	**13.99**	**18.04**	**22.21**	**18.98**	**17.20**	**17.34**

Compared Encoders. We compare our VecKM against several widely-used local geometry encoders: PointNet (Qi et al., 2017a), KPConv (Thomas et al., 2019) and DGCNN (Wang et al., 2019). **PointNet**. The input point cloud is first grouped into the shape of $(n, K, 3)$ and transformed into the shape of (n, K, d) by multi-layer perceptrons. Finally, a maxpooling operation shapes the data into (n, d). K is the number of neighboring points, which we attempt different values. **KPConv**. KPConv convolutes the local neighbors through a set of kernel points and transforms the convoluted features through a fully-connected layer. KPConv has a tunable parameter: the number of kernel points, which we attempt different values. **DGCNN** models the neighboring points as dynamic graphs and performs edge convolution to aggregate the local feature. We adopt the architecture in the original paper, which consists of five layers of edge convolution. DGCNN has a tunable parameter: the number of neighbors being convoluted, which we attempt different values. **VecKM** (Ours): We adopt a multi-scale of $\alpha = 60$ and $\beta = [10, 20]$. Since the size of the point cloud is large, we implement VecKM by Eqn. (2). We set d as 256 and p as 4096. We ensure the number of neighboring points considered by each encoder to be within 500∼1000, which is sufficient to estimate the local normals. After encoding the local geometry, three layers of neural network are applied to predict the normals.

Training Details. Each model is trained with a batch size of 200 for a total of 200 epochs. We use the Adam optimizer, setting the learning rate at 10^{-3}. For data augmentation, Gaussian noise is added to the input point cloud. The input point cloud and their normals are randomly rotated.

As shown in Table 1, **VecKM achieves $> 16\%$ lower errors than all the compared encoders and performs the best under all data corruption settings.** This reveals that VecKM effectively captures the local geometry and is more robust to input perturbation and density variation. The effectiveness of VecKM can be attributed to its reconstructive and isometric properties, and its noise robustness is derived from the robustness inherent in the kernel mixture. Figure 5 visualizes the explanation, which shows that even under corruptions, VecKM can still reconstruct local shapes and the

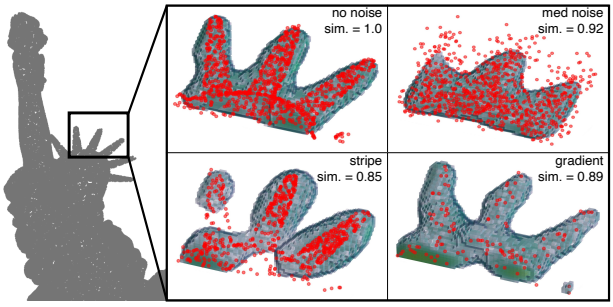

Figure 5. VecKM's robustness to data corruptions. VecKM can reconstruct the local shape under corrupted inputs. The VecKM encodings remain highly similar under data corruptions.

VecKM encodings are consistent. In the case of the *stripe* corruption setting, while the reconstruction may appear less accurate, the downstream neural network compensates for this discrepancy. This is evidenced by the relatively stable RMSE of the *stripe* setting in Table 1, indicating that the overall impact on performance is not substantial.

As shown in Figure 6, **VecKM is $>$ 100x faster than all the compared encoders and is scalable to large point cloud inputs.** Even when the input size is as large as 100k, VecKM only takes 150 ms to run. For memory cost, PointNet and DGCNN easily incur memory outrage when the neighbor size K is large because they require an intermediate step of (n, K, d) to compute the encoding. KPConv can be memory efficient through careful parallel programming, but existing implementations are not scalable to the settings we experiment with. VecKM, however, thanks to its unique factorizable property, only costs less than 8GB memory even with pure PyTorch implementation.

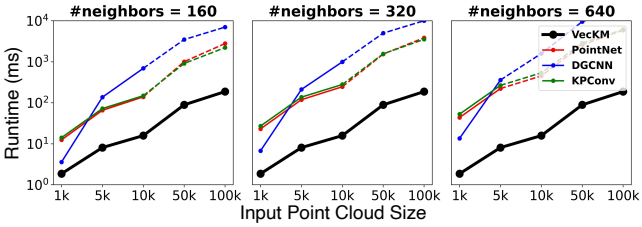

Figure 6. Runtime of local geometry encoders under different input point cloud size and neighbor size. All models are tested on an RTXA-5000 with 24 GB memory. Dash lines mean the memory is not sufficient to process all the points in one batch and has to process the points batches by batches.

4.2. Classification on ModelNet40 Dataset

We evaluate our VecKM on 3D object classification using the ModelNet40 dataset (Wu et al., 2015). We compare classification accuracy and inference time with the baselines.

Training Details. We use the same training setting for all the methods. We use the official split with 9,843 objects for training and 2,468 for testing. Each point cloud is uniformly

sampled to 1,024 points. During training, random translation in $[-0.2, 0.2]$, and random scaling in $[0.67, 1.50]$ are applied. We set the batch size to 32 and train the models for 250 epochs. We use the Adam optimizer, setting the initial learning rate as 0.001, with a cosine annealing scheduler. All models are trained and tested on an RTXA-5000.

Baselines. For our experiments, we select three widely-used point cloud architectures: PointNet (Qi et al., 2017a), PointNet++ (Qi et al., 2017b) and the Point Cloud Transformer (PCT) (Guo et al., 2021). We integrate VecKM encoding into these architectures as outlined in Sec. 3.4, which involves adding or replacing the original local geometry encoding modules with VecKM with $\alpha = 30$, $\beta = 6$ and $d = 256$. Since the size of the point cloud is small, we implement VecKM by Eqn. (3). We also compare VecKM-based architectures with the another light-weight network PointMLP (Ma et al., 2022).

Specifically, for PointNet, since it does not have a local geometry encoding module, we *add* the VecKM module before the PointNet, which means the PointNet receives the geometry encoding as input instead of the raw point coordinates. Since we *add* (denoted by $\rightarrow$) VecKM as an additional module, the runtime is going to be longer. For PointNet++, we *replace* (denoted by $\leftrightharpoons$) the first set abstraction layer with our VecKM encoding and leave the rest unchanged. For PCT, we *replace* the initial input embedding module with the VecKM while retaining the transformer modules. For PointNet++ and PCT, since we *replace* the dense local geometry encoding module with the more efficient VecKM, the runtime is expected to decrease.

As demonstrated in Table 2, **architectures based on VecKM consistently outperform their baseline counterparts in accuracy while also benefiting from significantly reduced runtime.** When VecKM is integrated with PointNet++ and PCT, not only is performance enhanced, but the speed of operation is also faster compared to the baselines. When comparing VecKM $\rightarrow$ PN against PointNet, there is a notable improvement in accuracy by 2.1% and 2.6%, with only a minimal increase in runtime. This is significant since the VecKM $\rightarrow$ PN architecture exhibits superior performance compared to both PointNet++ and PCT, and meanwhile operating 7.18x and 9.5x faster, respectively. Compared with PointMLP, VecKM-based architectures are even more efficient, achieving on-par accuracies.

4.3. Part Segmentation on ShapeNet Dataset

We evaluate our VecKM on 3D object part segmentation. Our experiment utilizes the ShapeNet (Chang et al., 2015) dataset. Similar to the classification experiment, we compare the IoU and inference time with PointNet, PointNet++, PCT, and PointMLP. The baselines and their VecKM counter-parts are obtained like the classification experiment

Table 2. Classification performance on the ModelNet40 dataset. VecKM $\rightarrow$ PN means *adding* VecKM as a preprocessing module to PointNet, so the runtime is expected to be longer than the PointNet baseline. VecKM $\leftrightharpoons$ PN++/PCT means *replacing* the original dense local geometry encoding in the original architectures with VecKM. Since VecKM is more efficient, the runtime is reduced.

	Instance Accuracy	Avg. Class Accuracy	Inference Time (ms) (1 batch)	# parameters
PointMLP	93.2%	90.1%	325.85	13.2M
PointNet	90.8%	87.1%	3.04	1.61M
VecKM $\rightarrow$ PN	92.9%	89.7%	14.32	9.06M
Difference	$\uparrow$ 2.1%	$\uparrow$ 2.6%	not comparable	+7.61M
PointNet++	92.7%	89.4%	117.13	1.48M
VecKM $\leftrightharpoons$ PN++	93.0%	89.7%	65.78	3.94M
Difference	$\uparrow$ 0.3%	$\uparrow$ 0.3%	78% faster	+2.46M
PCT	92.9%	89.8%	149.72	2.88M
VecKM $\leftrightharpoons$ PCT	93.1%	90.6%	21.44	5.07M
Difference	$\uparrow$ 0.2%	$\uparrow$ 0.8%	5.98x faster	+2.19M

Table 3. Part segmentation performance on the ShapeNet dataset. Similar to the classification, $\rightarrow$ means adding VecKM as a preprocessing module, so the runtime is expected to be longer. $\leftrightharpoons$ means replacing the dense local geometry encoding module with VecKM. Since VecKM is more efficient, the runtime is reduced.

	Instance mIoU	Avg. Class mIoU	Inference Time (ms) (1 batch)	# parameters
PointMLP	85.1%	82.1%	240.39	16.76M
PointNet	83.1%	77.6%	15.1	8.34M
VecKM $\rightarrow$ PN	84.9%	81.8%	40.8	1.29M
Difference	$\uparrow$ 1.8%	$\uparrow$ 4.2%	not comparable	+7.05M
PointNet++	85.0%	81.9%	130.8	1.41M
VecKM $\leftrightharpoons$ PN++	85.3%	82.0%	65.9	1.50M
Difference	$\uparrow$ 0.3%	$\uparrow$ 0.1%	98% faster	+0.09M
PCT	85.7%	82.6%	145.2	1.63M
VecKM $\leftrightharpoons$ PCT	85.8%	82.6%	46.6	1.71M
Difference	$\uparrow$ 0.1%	0.0%	2.11x faster	+0.08M

in Section 4.2. The parameters of VecKM are selected as $\alpha = 30, \beta = 9$ and $d = 256$. Since the size of the point cloud is small, we implement VecKM by Eqn. (3).

Training Details. We use the same training setting for all the methods. We use the official split with 14,006 3D models for training and 2,874 for testing. Each point cloud is uniformly sampled to 2,048 points. During training, random translation in $[-0.2, 0.2]$, and random scaling in $[0.67, 1.50]$ are applied. We set the batch size to 16 and train the model for 250 epochs. We use the Adam optimizer, setting the initial learning rate as 0.001, with a cosine annealing scheduler. All models are trained and tested on an RTXA-5000.

As demonstrated in Table 3, similar to the classfication experiment, architectures based on VecKM consistently outperform their baseline counter-parts in accuracy while also benefiting from significantly reduced runtime.

4.4. Semantic Segmentation on S3DIS Dataset

We evaluate our VecKM on 3D semantic segmentation. We use the S3DIS dataset (Armeni et al., 2017), which is an in-

Table 4. Semantic segmentation performance on the S3DIS dataset. Similar to the classification experiment, $\rightarrow$ means adding VecKM as a preprocessing module. Since PointNet++ downsamples the point cloud at the first layer while VecKM $\rightarrow$ PN++ does not, their inference time is not comparable. $\leftrightharpoons$ means replacing the original dense local geometry encoding module with VecKM. Since VecKM is more efficient, the runtime is reduced.

	Instance mIoU	Avg. Class mIoU	Overall Accuracy	Inference Time (ms) (per scene)	#parameters
PointNet++	64.05	71.52	87.92	96	0.968M
VecKM $\rightarrow$ PN++	67.48	73.53	89.33	391	1.11M
Difference	$\uparrow$ 3.43	$\uparrow$ 2.01	$\uparrow$ 1.41	not comparable	+0.142M
Point Transformer	69.29	75.66	90.36	559	7.77M
VecKM $\leftrightharpoons$ PT	69.53	75.84	90.39	447	7.93M
Difference	$\uparrow$ 0.24	$\uparrow$ 0.18	$\uparrow$ 0.03	20% faster	+0.16M

door scene dataset. It contains 6 areas and 271 rooms. Each point in this dataset is classified into one of 13 categories. Each scene contains around 10,000$\sim$100,000 points. We use the same training setting as Zhao et al. (2021).

Baselines. We select PointNet++ and Point Transformer (Zhao et al., 2021) as the baselines. For PointNet++, in its first layer, PointNet++ first downsamples the point cloud by $1/4$ and for each sampled point, 32 neighboring points are sampled and transformed by a PointNet. The VecKM $\rightarrow$ PN++ counter-part is obtained by *adding* the dense local geometry encoder before the first layer. Consequently, the PointNet in the first layer will transform the local geometry encoding instead of the raw 3d coordinates. Because of the downsampling operation in PointNet++, its inference time is much shorter. Therefore, PN++ and VecKM $\rightarrow$ PN++ are not comparable in terms of inference time. For Point Transformer, its first layer is a dense local geometry encoder with PointNet. We *replace* the dense local geometry encoder with our VecKM encoding to obtain the PT $\leftrightharpoons$ VecKM architecture. In both architectures, since the size of the point cloud is large, we implement VecKM by Eqn. (2). We set $\alpha = 30, \beta = 9, d = 256, p = 2048$, and we use a sequence of two complex linear layers to transform the local geometry encoding from $\mathbb{C}^{256}$ to $\mathbb{C}^{64}$.

As shown in Table 4, **VecKM improves PointNet++ baseline significantly**. This is because the downsampling of the point cloud induces information loss in the PointNet++ baseline, while the dense VecKM encoding effectively bridges the gap. On the other hand, **VecKM improves the inference speed of point transformer**, which is expected given the efficiency of VecKM especially on large point cloud input. Regarding why VecKM $\leftrightharpoons$ PT does not yield better accuracy, it is possibly because the heavy-weight point transformer architecture already adequately reasons on the geometry. Unlike PointNet++, the local geometry encoding is not a bottleneck for point transformer. Since the subsequent processing costs the majority of the running time, the acceleration is not as significant as the previous experiments.

Table 5. Ablation study on the selection of the parameters α and β in Theorem 2, in the context of ModelNet40 classification experiment. Numbers greater than 92.5% are bolded.

	$\alpha = 20$	$\alpha = 25$	$\alpha = 30$	$\alpha = 35$
$\beta = 4$	91.73%	91.94%	91.73%	91.77%
$\beta = 6$	**92.59%**	92.14%	**92.87%**	**92.50%**
$\beta = 9$	92.18%	**92.71%**	**92.95%**	**92.50%**
$\beta = 12$	92.10%	**92.54%**	**92.59%**	92.38%

5. Ablation Studies

In Section 3.3, we qualitatively analyze the effect of the parameters α and β in Theorem 2. In this section, we quantitatively analyze the effect of the parameters in the context of the ModelNet40 classification experiment, with the VecKM $\rightarrow$ PN architecture. For α selection, when the input point cloud is normalized within a unit ball, setting α in the range of $(20, 35)$ yields good performance. As shown in Table 5, appropriate selections of α and β are important to yield a good performance on the downstream tasks.

We study how many fully-connected layers are needed for transforming the VecKM encoding, in the context of normal estimation tasks. As shown in Figure 7, two layers are sufficient for stably satisfactory performance, highlighting the inherent descriptiveness of VecKM encoding.

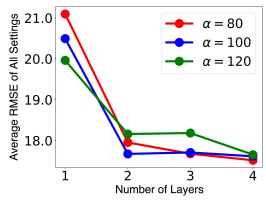

Figure 7. Average RMSE of normal estimation trained with different numbers of layers.

Notice that the selection of the α parameter varies across different tasks. In classification, where refined local geometry is less critical, a smaller α is used to abstract away finer details. For normal estimation tasks, where accurate local shape representation is crucial, a larger α is employed to retain essential details. These findings demonstrate VecKM's adaptability in meeting the diverse requirements of various tasks, adjusting to the specific level of detail needed.

6. Conclusion

VecKM, our novel local point cloud encoder, stands out for its efficiency and noise robustness. VecKM vectorizes a kernel mixture associated with the local point cloud, providing a solid theoretical foundation for its descriptiveness and robustness. Thanks to its special formulation, VecKM is the only existing local geometry encoder that costs linear time and space. Through extensive experiments, VecKM has demonstrated significant improvements in speed and accuracy across a variety of point cloud processing tasks. VecKM has many potential applications due to its notable features. Its efficiency facilitates faster inference, ideal for time-critical tasks like event data processing.

Impact Statements

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgement

The support of NSF under awards OISE 2020624 and BCS 2318255, and ARL under the Army Cooperative Agreement W911NF2120076 is greatly acknowledged. We extend our gratitude to Pixabay for providing free access to the icon used in drafting Figure 2 and 8.

References

Armeni, I., Sax, S., Zamir, A. R., and Savarese, S. Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105*, 2017.

Bochner, S. *Harmonic analysis and the theory of probability*. Courier Corporation, 2005.

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 preprint arXiv:1512.03012*, 2015.

Chen, Y., Yuan, D., Chen, W., Hu, M., Fung, J. C., Sun, H., and Lu, X. Estimation and variation analysis of secondary inorganic aerosols across the greater bay area in 2005 and 2015. *Chemosphere*, 292:133393, 2022.

Fehr, D., Cherian, A., Sivalingam, R., Nickolay, S., Morellas, V., and Papanikolopoulos, N. Compact covariance descriptors in 3d point clouds for object recognition. In *2012 IEEE international conference on robotics and automation*, pp. 1793–1798. IEEE, 2012.

Frady, E. P., Kleyko, D., Kymn, C. J., Olshausen, B. A., and Sommer, F. T. Computing on functions using randomized vector representations (in brief). In *Proceedings of the 2022 Annual Neuro-Inspired Computational Elements Conference*, pp. 115–122, 2022.

Ge, X. Non-rigid registration of 3d point clouds under isometric deformation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 121:192–202, 2016.

Guerrero, P., Kleiman, Y., Ovsjanikov, M., and Mitra, N. J. Pcpnet learning local shape properties from raw point clouds. In *Computer graphics forum*, volume 37, pp. 75–85. Wiley Online Library, 2018.

Guo, M.-H., Cai, J.-X., Liu, Z.-N., Mu, T.-J., Martin, R. R., and Hu, S.-M. Pct: Point cloud transformer. *Computational Visual Media*, 7:187–199, 2021.

Han, X.-F., He, Z.-Y., Chen, J., and Xiao, G.-Q. 3crossnet: Cross-level cross-scale cross-attention network for point cloud representation. *IEEE Robotics and Automation Letters*, 7(2):3718–3725, 2022.

Han, X.-F., Feng, Z.-A., Sun, S.-J., and Xiao, G.-Q. 3d point cloud descriptors: state-of-the-art. *Artificial Intelligence Review*, pp. 1–51, 2023.

Lai, X., Liu, J., Jiang, L., Wang, L., Zhao, H., Liu, S., Qi, X., and Jia, J. Stratified transformer for 3d point cloud segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8500–8509, 2022.

Li, Y., Bu, R., Sun, M., Wu, W., Di, X., and Chen, B. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 2018.

Lu, X., Yuan, D., Chen, Y., Fung, J. C., Li, W., and Lau, A. K. Estimations of long-term nss-so42–and no3–wet depositions over east asia by use of ensemble machine-learning method. *Environmental Science & Technology*, 54(18):11118–11126, 2020.

Ma, X., Qin, C., You, H., Ran, H., and Fu, Y. Rethinking network design and local geometry in point cloud: A simple residual mlp framework. *arXiv preprint arXiv:2202.07123*, 2022.

Pan, X., Xia, Z., Song, S., Li, L. E., and Huang, G. 3d object detection with pointformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7463–7472, 2021.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., and Kong, L. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

Prakhya, S. M., Lin, J., Chandrasekhar, V., Lin, W., and Liu, B. 3dhopd: A fast low-dimensional 3-d descriptor. *IEEE Robotics and Automation Letters*, 2(3):1472–1479, 2017.

Qi, C. R., Su, H., Mo, K., and 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*, pp. 652–660, 2017a.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017b.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

Ran, H., Liu, J., and Wang, C. Surface representation for point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18942–18952, 2022.

Rusu, R. B., Marton, Z. C., Blodow, N., and Beetz, M. Persistent point feature histograms for 3d point clouds. In *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany*, pp. 119–128, 2008.

Schölkopf, B., Williamson, R. C., Smola, A., Shawe-Taylor, J., and Platt, J. Support vector method for novelty detection. *Advances in neural information processing systems*, 12, 1999.

Sen, A., Pan, G., Mitrokhin, A., and Islam, A. Scenecalib: Automatic targetless calibration of cameras and lidars in autonomous driving. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7771–7777. IEEE, 2023.

Steder, B., Rusu, R. B., Konolige, K., and Burgard, W. Narf: 3d range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 44, pp. 2. Citeseer, 2010.

Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., and Guibas, L. J. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6411–6420, 2019.

Trabelsi, C., Bilaniuk, O., Serdyuk, D., Subramanian, S., Santos, J. F., Mehri, S., Rostamzadeh, N., Bengio, Y., and Pal, C. J. Deep complex networks. *CoRR*, abs/1705.09792, 2017. URL http://arxiv.org/abs/1705.09792.

Triebel, R., Kersting, K., and Burgard, W. Robust 3d scan point classification using associative markov networks. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 2603–2608. IEEE, 2006.

Vandapel, N., Huber, D. F., Kapuria, A., and Hebert, M. Natural terrain classification using 3-d ladar data. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 5, pp. 5117–5122. IEEE, 2004.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38 (5):1–12, 2019.

Wu, W., Qi, Z., and Fuxin, L. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 9621–9630, 2019.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and 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.

Xiang, T., Zhang, C., Song, Y., Yu, J., and Cai, W. Walk in the cloud: Learning curves for point clouds shape analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 915–924, 2021.

Xu, Y., Fan, T., Xu, M., Zeng, L., and Qiao, Y. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 87–102, 2018.

Yan, X., Zheng, C., Li, Z., Wang, S., and Cui, S. Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5589–5598, 2020.

Yu, X., Tang, L., Rao, Y., Huang, T., Zhou, J., and Lu, J. Point-bert: Pre-training 3d point cloud transformers with masked point modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19313–19322, 2022.

Yuan, D., Huang, F., Fermüller, C., and Aloimonos, Y. Decodable and sample invariant continuous object encoder. *arXiv preprint arXiv:2311.00187*, 2023.

Zampogiannis, K., Fermüller, C., and Aloimonos, Y. Topology-aware non-rigid point cloud registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(3):1056–1069, 2019.

Zhao, H., Jiang, L., Jia, J., Torr, P. H., and Koltun, V. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 16259–16268, 2021.

A. Proof of Lemma 1

Lemma 1 (VecKM embodies a Gaussian kernel)**.** *Let* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^3$*,* $\mathbf{A} \in \mathbb{R}^{3\times d}$*. All elements in* $\mathbf{A}$ *are drawn from normal distribution* $\mathcal{N}(0, \alpha^2)$*. Then as* $d \to \infty$*,*

$$\frac{1}{d}\langle e^{i\mathbf{x}\mathbf{A}}, e^{i\mathbf{y}\mathbf{A}}\rangle \to \mathcal{G}_\alpha(\mathbf{x}, \mathbf{y}) := \exp\big(-\frac{\alpha^2||\mathbf{x}-\mathbf{y}||^2}{2}\big)$$

Proof. Let $\mathbf{a} \in \mathbb{R}^3$ where $\mathbf{a} \sim \mathcal{N}(\mathbf{0}, \alpha^2\mathbf{I}_{3\times 3})$ be one column of the matrix $\mathbf{A}$, we claim that $\mathbb{E}\big[\Re\big(e^{i\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})}\big)\big] = \mathcal{G}_\alpha(\mathbf{x}, \mathbf{y})$:

$$\begin{aligned}
\mathbb{E}\Big[\Re\big(e^{i\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})}\big)\Big] &= \mathbb{E}\Big[\cos\big(\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})\big)\Big] \\
&= \mathbb{E}\Big[\sum_{k=0}^{\infty}(-1)^k\frac{\big(\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})\big)^{2k}}{(2k)!}\Big] \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\mathbb{E}\Big[\big(\sum_{j=1}^{3}(\mathbf{x}_j-\mathbf{y}_j)\mathbf{a}_j\big)^{2k}\Big] \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\mathbb{E}\Big[\big(\alpha||\mathbf{x}-\mathbf{y}||Z\big)^{2k}\Big] \quad \text{where } Z \in \mathcal{N}(0,1) \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\cdot\alpha^{2k}||\mathbf{x}-\mathbf{y}||^{2k}\cdot\frac{(2k)!}{k!2^k} \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{k!2^k}\cdot\alpha^{2k}||\mathbf{x}-\mathbf{y}||^{2k} \\
&= \exp\big(-\frac{\alpha^2||\mathbf{x}-\mathbf{y}||^2}{2}\big)
\end{aligned}$$

On the other hand, $\mathbb{E}\big[\Im\big(e^{i\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})}\big)\big] = 0$ because normal distribution is a symmetric distribution around 0:

$$\begin{aligned}
\mathbb{E}\Big[\Im\big(e^{i\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})}\big)\Big] &= \mathbb{E}\Big[\sin\big(\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})\big)\Big] \\
&= \mathbb{E}\Big[\sum_{k=0}^{\infty}(-1)^k\frac{\big(\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})\big)^{2k+1}}{(2k+1)!}\Big] \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)!}\mathbb{E}\Big[\big(\sum_{j=1}^{3}(\mathbf{x}_j-\mathbf{y}_j)\mathbf{a}_j\big)^{2k+1}\Big] \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)!}\mathbb{E}\Big[\big(\alpha||\mathbf{x}-\mathbf{y}||Z\big)^{2k+1}\Big] \quad \text{where } Z \in \mathcal{N}(0,1) \\
&= 0 \quad \text{because } \mathbb{E}(Z^{2k+1}) = 0
\end{aligned}$$

Therefore, when we randomize d rows of such $\mathbf{a}$ vector, the inner product $\frac{1}{d}\langle e^{i\mathbf{x}\mathbf{A}}, e^{i\mathbf{y}\mathbf{A}}\rangle = \frac{1}{d}\sum_{k=1}^{d} e^{i\mathbf{a}_k\cdot(\mathbf{x}-\mathbf{y})}$ will converge to $\mathcal{G}_\alpha(\mathbf{x}, \mathbf{y})$ thanks to the Law of Large Number and the Central Limit Theorem. □

B. PyTorch Implementation of VecKM

```python
import torch
import torch.nn as nn
import numpy as np
from scipy.stats import norm

def strict_standard_normal(d):
    # this function generate very similar outcomes as torch.randn(d)
    # but the numbers are strictly standard normal, no randomness.
    y = np.linspace(0, 1, d+2)
    x = norm.ppf(y)[1:-1]
    np.random.shuffle(x)
    x = torch.tensor(x).float()
    return x

class VecKM(nn.Module):
    def __init__(self, d=256, alpha=6, beta=1.8, p=4096):
        super().__init__()
        self.alpha, self.beta, self.d, self.p = alpha, beta, d, p
        self.sqrt_d = d ** 0.5

        self.A = torch.stack(
            [strict_standard_normal(d) for _ in range(3)],
            dim=0
        ) * alpha
        self.A = nn.Parameter(self.A, False)            # Real(3, d)

        self.B = torch.stack(
            [strict_standard_normal(p) for _ in range(3)],
            dim=0
        ) * beta
        self.B = nn.Parameter(self.B, False)            # Real(3, d)

    def forward(self, pts):
        """ Compute the dense local geometry encodings of the given point cloud.
        Args:
            pts: (bs, n, 3) or (n, 3) tensor, the input point cloud.

        Returns:
            G: (bs, n, d) or (n, d) tensor. the dense local geometry encodings.
        """
        pA = pts @ self.A                               # Real(..., n, d)
        pB = pts @ self.B                               # Real(..., n, p)
        eA = torch.concatenate(
            (torch.cos(pA), torch.sin(pA)), dim=-1)     # Real(..., n, 2d)
        eB = torch.concatenate(
            (torch.cos(pB), torch.sin(pB)), dim=-1)     # Real(..., n, 2p)
        G = torch.matmul(
            eB,                                         # Real(..., n, 2p)
            eB.transpose(-1,-2) @ eA                    # Real(..., 2p, 2d)
        )                                               # Real(..., n, 2d)
        G = torch.complex(
            G[..., :self.d], G[..., self.d:]
        ) / torch.complex(
            eA[..., :self.d], eA[..., self.d:]
        )                                               # Complex(..., n, d)
        G = G / torch.norm(G, dim=-1, keepdim=True) * self.sqrt_d
        return G

vkm = VecKM()
pts = torch.rand((10,1000,3))
print(vkm(pts).shape) # it will be Complex(10,1000,256)
pts = torch.rand((1000,3))
print(vkm(pts).shape) # it will be Complex(1000, 256)
```

```
from complexPyTorch.complexLayers import ComplexLinear, ComplexReLU
# You may want to use apply two-layer feature transform to the encoding.
feat_trans = nn.Sequential(
    ComplexLinear(256, 128),
    ComplexReLU(),
    ComplexLinear(128, 128)
)
G = feat_trans(vkm(pts))
G = G.real**2 + G.imag**2 # it will be Real(10, 1000, 128) or Real(1000, 128).
```

C. Effect of Parameters α, β, d, p

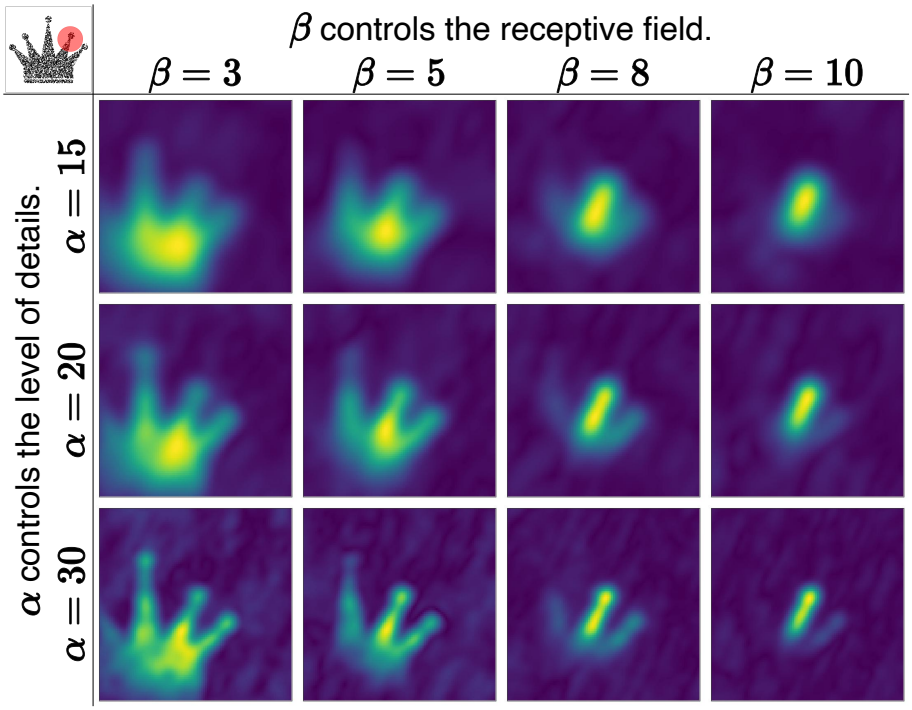

Figure 8. Effect of the parameters α and β in Theorem 2.

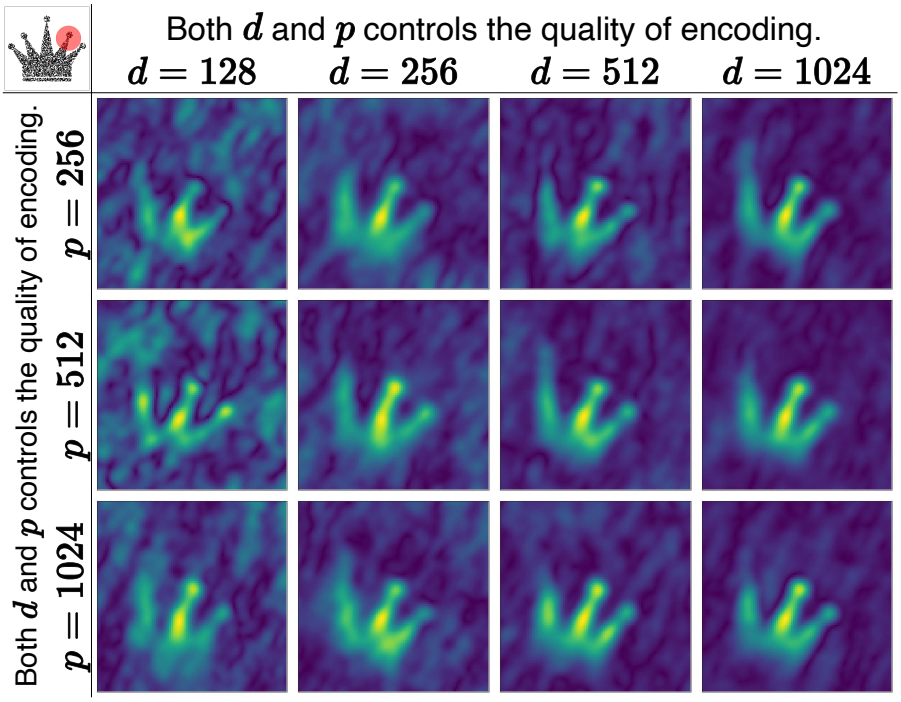

Figure 9. Effect of the parameters d and p in Theorem 2.

D. Guidance for Selecting the Parameter β

The statistics is obtained by $r = \min_r\{r : e^{-r^2/2} < 0.1\}$. The beta and the radius have a relation of $\beta_1 r_1 = \beta_2 r_2$.

Table 6. Relation between the parameter β and the neighborhood radius.

beta	1	2	3	4	5	6	7	8	9	10
radius	1.800	0.900	0.600	0.450	0.360	0.300	0.257	0.225	0.200	0.180
beta	11	12	13	14	15	16	17	18	19	20
radius	0.163	0.150	0.138	0.129	0.120	0.113	0.106	0.100	0.095	0.090
beta	21	22	23	24	25	26	27	28	29	30
radius	0.086	0.082	0.078	0.075	0.072	0.069	0.067	0.065	0.062	0.060