3D Gaussian Point Encoders

Jim James
Georgia Tech
jimjames@gatech.edu

Benjamin Wilson
Georgia Tech

Simon Lucey
University of Adelaide

James Hays
Georgia Tech

Abstract

*In this work, we introduce the 3D Gaussian Point Encoder, an explicit per-point embedding built on mixtures of learned 3D Gaussians. This explicit geometric representation for 3D recognition tasks is a departure from widely used implicit representations such as PointNet. However, it is difficult to learn 3D Gaussian encoders in end-to-end fashion with standard optimizers. We develop optimization techniques based on natural gradients and distillation from PointNets to find a Gaussian Basis that can reconstruct PointNet activations. The resulting 3D Gaussian Point Encoders are faster and more parameter efficient than traditional PointNets. As in the 3D reconstruction literature where there has been considerable interest in the move from implicit (e.g., NeRF) to explicit (e.g., Gaussian Splatting) representations, we can take advantage of computational geometry heuristics to accelerate 3D Gaussian Point Encoders further. We extend filtering techniques from 3D Gaussian Splatting to construct encoders that run **2.7**× faster as a comparable accuracy PointNet while using **46%** less memory and **88%** fewer FLOPs. Furthermore, we demonstrate the effectiveness of 3D Gaussian Point Encoders as a component in Mamba3D, running **1.27**× faster and achieving a reduction in memory and FLOPs by **42%** and **54%** respectively. 3D Gaussian Point Encoders are lightweight enough to achieve high framerates on CPU-only devices.*

1. Introduction

Point cloud processing plays a crucial role in robotics and autonomous vehicles where LiDAR and related sensors capture three-dimensional spatial data. Since point clouds are unordered sets of points, deep networks designed for point cloud analysis must be permutation invariant to ensure consistent representations regardless of input ordering. Methods such as PointNet achieve this by employing symmetric aggregation functions, preserving the inherent structure of the data while enabling effective learning. This key feature has made PointNet ubiquitous in a variety of 3D tasks, including: classification [21, 26], detection [19, 38, 46], and segmentation.

In PointNet, the majority of computational cost arises from per-point embedding, as it requires computing multiple large MLPs across a high number of points in each input point cloud. In contrast, the classifier stage applies MLPs only to a single global feature, making it relatively lightweight. To address this inefficiency, prior works have explored alternative approaches [44], such as LUTI-MLP [28], which replaces computationally expensive ReLU-MLP operations with lookup tables, and GPointNet [29], which employs single Gaussians. Although these methods greatly reduce FLOPs per sample compared to PointNet, their throughput on low-power platforms, such as CPU inference, remains limited. LUTI-MLP suffers from complex memory access patterns, while GPointNet requires evaluating a large number of Gaussian kernels, both of which hinder performance in resource-constrained environments.

Recently, explicit models based on mixtures of 3D Gaussians have gained traction in the view-synthesis literature due to their ability to efficiently represent volumetric data [5, 8, 14, 17]. Several studies have leveraged the explicit nature of 3D Gaussians to reduce computational costs, employing techniques such as Gaussian pruning [8, 15, 22] and heuristic-based filtering of low-value Gaussian-point pairs [14, 39]. These optimizations significantly accelerate inference compared to per-point coordinate networks.

In this work, we propose a novel **3D Gaussian Point Encoder**, a per-point embedding that integrates PointNet's max-pooling aggregation with performance optimizations from view synthesis using mixtures of 3D Gaussians. By interpreting each dimension of PointNet's embedding function as a volumetric representation, we leverage the capacity of 3D Gaussian mixtures to model volumes, enabling a lightweight approximation of a pre-trained PointNet. Moreover, we demonstrate it is possible to train this encoder end-to-end through Gaussian-specific natural gradient methods. Additionally, we exploit the explicit structure of Gaussians to enhance computational efficiency through filtering Gaussian-Point pairs. To assess the effectiveness of our approach, we conduct shape classification experiments

on ModelNet40 [36] and ScanObjectNN [31] while equipping our encoder with classical and modern classifiers from PointNet and Mamba3D. In summary, our primary contributions are:

(i) We present a novel *explicit* 3D representation as a drop in replacement for the *implicit* PointNet representations which are ubiquitous in 3D scene understanding

(ii) We discover that 3D Gaussian representations present significant optimization challenges when using off-the-shelf optimizers. We find two paths to overcome this roadblock – distillation from PointNet teachers and direct optimization with Natural Gradients

(iii) We show that explicit representations can benefit from geometric acceleration techniques, such as pairwise Gaussian-point filtering, inspired by the 3DGS literature

(iv) We demonstrate that 3D Gaussian representations can achieve similar levels of accuracy to PointNet per-point embeddings, while achieving **2.7**$\times$ higher throughput and **46%** less memory. When integrated into Mamba3D, we achieve **1.27**$\times$ the throughput and **42%** less memory.

2. Related Work

Point Embeddings. PointNet [25] is one of the first models to directly process point clouds, utilizing a per-point MLP with ReLU activations followed by a max-pooling operator. The output of this MLP serves as a spatial encoding for each point, while max-pooling aggregates these per-point embeddings into a single global feature representing the entire point cloud. Several works have extended PointNet to support hierarchical feature learning, including PointNet++ [25] and its modern variants [21, 27].

Several approaches rely on Transformers [32] as a component of their backbone, such as Point Cloud Transformer [11] and Point Transformer [34, 35, 43]. Transformer methods have the advantage of being possible to train from the vast quantity of unlabeled data via self-supervised learning, as done in Point-MAE [23] and Point-BERT [40]. However, Transformers suffer from quadratic time complexity in sequence length, potentially resulting in inefficiency when processing large point sets. To resolve this issue, recent approaches instead utilize Mamba [4, 10], a structured state space model alternative to the Transformer with linear time complexity. PointMamba [20] and PCM [42] aim to produce a vanilla Mamba-based model without a hierarchical encoder architecture. Most recently, Mamba3D [13] achieves near state-of-the-art performance on point classification, expanding upon PointMamba through the use of a bidirectional Mamba variant and local feature aggregation.

Efficient PointNet Variants. A variety of prior works have explored efficient point encoders based on PointNet's per-point embedding with max-pooling framework. LUTI-MLP [28] utilizes a lookup table per dimension of PointNet's embeddings formed followed by trilinear interpolation to form point embeddings. The lookup table is optimized during training time by discretizing and interpolating a pre-trained PointNet MLP, which is then voxelized at test time. This results in faster calculation of point embeddings compared to PointNet's MLP for 3D point clouds. However, as the input dimension increases, the runtime and memory cost grows exponentially due to the increased lookup table size and number of neighbors to interpolate. GPointNet [29] instead represents each dimension of a point embedding via the likelihood of a single anisotropic Gaussian, resulting in an encoder requiring significantly fewer FLOPs per sample compared to PointNet. Other approaches [41, 44] additionally utilize untrained random features for improved robustness. RobustPPE [44] uses random fourier features in place of an MLP, while VecKM [41] leverages randomized kernels to replace sampling and grouping operations typically used for encoding local geometric features.

Preconditioning and Natural Gradients. Preconditioning is a technique in which an optimization problem is transformed to make it more amenable to numerical solvers. Several optimizers internally apply preconditioning to their gradients to stabilize training. These include the diagonal preconditioners in AdaGrad [6] and Adam [18], as well as the block diagonal preconditioners in modern optimizers such as Shampoo [12] and SOAP [33]. One explicit form of preconditioning is Natural Gradients [1], a generalization of steepest descent for arbitrary metric spaces. This in contrast to standard gradient descent, where steps are considered with fixed Euclidean distance. Amari [1] demonstrated that given a metric, Natural Gradient descent can be viewed as preconditioning the gradients by the inverse of the metric space's Riemannian metric tensor.

Mixtures of Gaussians as Approximators. Methods utilizing mixtures of Gaussians, or more generally, radial basis functions [24], have been widely studied. Classical works have used isotropic Gaussians to approximate volumes [45]. Most recently, a variety of works involving Gaussians have been applied to novel view synthesis. 3D Gaussian Splatting (3DGS) [17] represents volumes via a mixture of anisotropic Gaussians, and is able to render novel views signficantly faster than coordinate networks. The use of explicit Gaussians allows the method to exploit sparsity in real-world scenes. However, optimizing the set of Gaussians requires additional techniques compared to coordinate neural network-based approaches. Niemeyer et al. [22] notes that utilizing guidance from a pre-trained

coordinate network can help train a more robust Gaussian representation to work around this issue.

Several works [7, 8, 15, 22] have reduced the computational costs of 3DGS via pruning and filtering. Ye et al. [39] improve runtime by learning a truncation threshold on the Mahalanobis distance for each Gaussian, while Hanson et al. [14] instead propose filtering before the Mahalanobis distance calculation by bounding each Gaussian with a rectangle or bounding via tiles, and then only computing points that fall within each bounding box or tile respectively.

3. Method

We introduce the **3D Gaussian Point Encoder** (3DGPE), which replaces PointNet representations by simple, explicit 3D Gaussian functions for effective 3D shape classification. Surprisingly, we find that distilling point cloud features into a Gaussian-based network yields superior performance compared to directly optimizing Gaussian parameters. Additionally, our 3D Gaussian representation significantly reduces computational overhead by efficiently removing Gaussians and Gaussian-point pairs that do not meaningfully contribute to the final feature representation.

Our 3D Gaussian Point Encoder comprises two key components: the **Gaussian Basis Encoder** and the **Gaussian Basis Mixer**. The Gaussian Basis Encoder encodes a point's representation by computing its proximity to a set of 3D Gaussians, effectively capturing local geometric features. The Gaussian Basis Mixer then integrates these Gaussian-based features, transforming them into a richer and more expressive feature representation. This structured approach enables efficient and flexible encoding of spatial information for downstream tasks. In the following section, we outline their construction.

3.1. Gaussian Basis Encoder

The Gaussian Basis Encoder is a parametric function that maps input points from a 3D point cloud into a structured feature space using a set of learnable Gaussian functions. Given a point cloud $\mathcal{X} = \{x_i\}_{i=1}^{N}$, where each point $x_i \in \mathbb{R}^3$, the encoder represents the input as a mixture of spatial Gaussians. Each Gaussian component g is defined by a mean $\boldsymbol{\mu}_g \in \mathbb{R}^3$, which represents the center of the Gaussian in 3D space; a precision matrix (inverse of covariance matrix) $\boldsymbol{\Sigma}_g^{-1} \in \mathbb{R}^{3\times3}$, modeling spatial extent; and a set of mixture coefficients $\{\alpha_{g,k}\}_{k=1}^{K}$, where K denotes the number of activation volumes.

Precision Matrix Parameterization. To ensure that $\boldsymbol{\Sigma}_g^{-1}$ remains positive semi-definite, we parameterize it using the Cholesky decomposition [5]:

$$\boldsymbol{\Sigma}_g^{-1} = \mathbf{L}_g \mathbf{L}_g^{\top}, \tag{1}$$

where $\mathbf{L}_g$ is a lower triangular matrix. This factorization guarantees valid precision matrices while enabling efficient optimization. We parameterize precision directly to reduce the risk of numerical instability during training.

Feature Encoding. For each input point x, we compute its unweighted Gaussian likelihood under each Gaussian g as follows:

$$\phi_g(x) = \exp\left(-\frac{1}{2}(x-\boldsymbol{\mu}_g)^{\top}\boldsymbol{\Sigma}_g^{-1}(x-\boldsymbol{\mu}_g)\right). \tag{2}$$

This function measures the proximity of x to the Gaussian distribution centered at $\boldsymbol{\mu}_g$, with spatial spread determined by $\boldsymbol{\Sigma}_g$.

3.2. Gaussian Basis Mixer

Following the Gaussian Basis Encoder, we introduce the Gaussian Basis Mixer, a critical component of our architecture that distinguishes it from prior methods such as GPointNet [29]. Unlike previous approaches, the Gaussian Basis Mixer employs shared Gaussians across multiple activation volumes, effectively utilizing these Gaussians as basis functions. This design exploits redundancy, enhancing efficiency and enabling the network to represent complex activation volumes beyond simple ellipsoids.

Mathematically, the Gaussian Basis Mixer applies a linear layer using mixture coefficients to combine Gaussians and form activation volumes:

$$l_k(x) = \sum_{g=1}^{N_G} \alpha_{g,k}\phi_g(x) + b_k, \tag{3}$$

where b_k is a bias term unique to each activation volume. Following this, we maxpool across points to produce a permutation-invariant global feature.

Gaussian sharing significantly reduces latency and memory overhead as the input dimension increases, addressing a key computational bottleneck. The complexity of computing Gaussian likelihoods grows quadratically with input dimension due to the Mahalanobis distance computation. In contrast, the additional cost of uniquely recombining Gaussians for each activation volume scales only linearly with both the total number of Gaussians (N_G) and the number of activation volumes (K). This trade-off enables our architecture to efficiently handle high-dimensional inputs while maintaining expressiveness.

3.3. Shape Classification Architecture

We primarily experiment with utilizing our encoder with two classification architectures: PointNet [25] and Mamba3D [13].

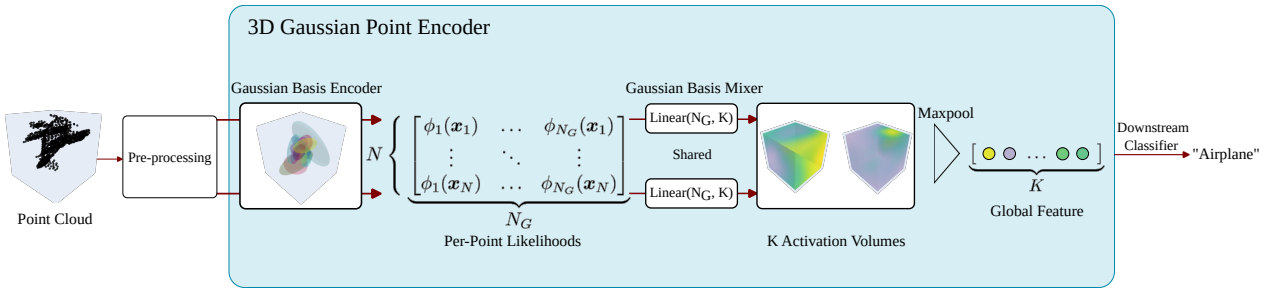


Figure 1. **Base architecture of 3DGPE.** An input point cloud is first pre-processed, such as by a T-Net or through Farthest Point Sampling and KNN. Afterwards, each input point is processed independently through the Gaussian Basis Encoder by first computing a set of Gaussian likelihoods, followed by the Gaussian Basis Mixer, mixing the likelihoods to form a set of embeddings for each activation volume. We max-pool across points to derive a global feature which is then passed to a downstream classifier, such as an MLP.

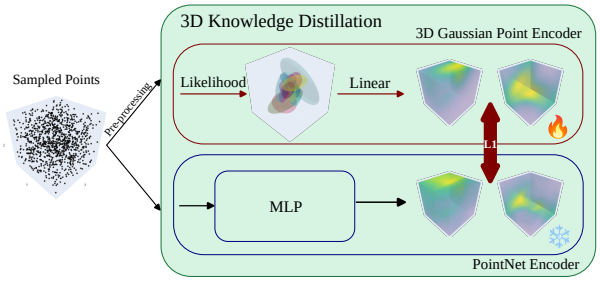


Figure 2. **Implicit to Explicit 3D Knowledge Distillation.** Points are sampled and pre-processed (T-Net or FPS + KNN) before being passed through each encoder. We then measure L_1 loss between the 3D Gaussian Point Encoder and PointNet per-point embeddings. Maroon outlines indicate trainable components, while blue indicates frozen components.

3D Gaussian Point Encoder with PointNet. The 3D Gaussian Point Encoder serves as the per-point embedding network; however, we add a few critical components to mimic a PointNet. The T-Net used in PointNet predicts a rotation matrix to achieve invariance to geometric transformations such as translation, rotation, and scaling. Since it's constructed from a PointNet, we are able to replace it with a 3DGPE network. We add the 3D Gaussian T-Net prior to passing the points through the backbone network. After then generating the global feature from the 3DGPE network, we compute our class logits by passing the global feature through a simple MLP classifier. This is equivalent in architecture to PointNet's classifier.

3D Gaussian Point Encoder with Mamba3D. Here, the 3D Gaussian Point Encoder serves as the patch encoder, generating feature embeddings for point sets formed through farthest point sampling and KNN-based grouping. After the point patches have been passed through the 3D Gaussian Point Encoder, we pass these embeddings through Mamba3D's middle encoder blocks while applying positional encodings. These blocks consist of a per-group normalization and feature aggregation operation, followed by a bi-directional state space model to capture global information about the point patch embeddings. We then compute class logits by applying an MLP classifier to the aggregated point embeddings. See Han et al. [13] for more details.

3.4. Optimization of 3D Gaussian Point Encoder

We observe that end-to-end training of the 3D Gaussian Point Encoder with standard optimizers yields significantly lower performance compared to baseline models in PointNet and Mamba3D, as shown in Tab. 2. We uncover two strategies to bypass this roadblock. The first is preconditioning the gradient via *natural gradients*, and the second is to *distill* the implicit geometry of PointNet features to the explicit geometry of 3DGPE.

3.4.1. Natural Gradients for 3D Gaussians

Standard gradient descent minimizes loss by stepping in the direction of steepest decrease in the loss, assuming a fixed step size in Euclidean distance. Natural gradients [1] generalize this by considering a different metric for step size, often resulting in faster convergence. Amari [1] notes that natural gradient descent can be performed via SGD while preconditioning the gradients by the inverse of the Riemannian metric tensor associated with a given parameter space, like so:

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \gamma \mathbf{G}^{-1} \nabla \mathcal{L}(\mathbf{x}^t), \tag{4}$$

where $\mathbf{x}^t$ is a parameter at iteration t, γ is the learning rate, $\mathbf{G}$ is the Riemannian metric tensor, and $\mathcal{L}$ is the loss function. Gaussians as primitives admit two natural metrics.

Mahalanobis Distance. Gaussian likelihoods are a function of the Mahalanobis distance of a query point to the mean with respect to the precision matrix. Accordingly, we can consider each Gaussian's mean an element in the metric space equipped with the Mahalanobis distance given its

precision matrix. In this case, the Riemannian metric tensor is the precision matrix itself [16]. Thus, the natural gradient update for the g-th Gaussian's mean becomes:

$$\boldsymbol{\mu}_g^{t+1} = \boldsymbol{\mu}_g^t - \gamma \boldsymbol{\Sigma}_g^t \nabla \mathcal{L}\left(\boldsymbol{\mu}_g^t\right). \tag{5}$$

See Sec. 1 of the supplemental material for an example.

Fisher Information Metric. If we view each Gaussian primitive as a probability distribution, we can treat its mean and Cholesky decomposition parameters combined as an element in a parameter space defining probability distributions. One commonly used divergence for comparing such distributions is the KL divergence. The KL divergence can be approximated via a second order Taylor expansion to Fisher Information [30]. In this case, the Riemannian metric tensor is the Fisher Information Matrix $\boldsymbol{F}_g$, whose inverse can be easily computed in closed form and includes the same mean update as the Mahalanobis case [30]. Thus, the natural gradient update for the g-th Gaussian's parameters becomes:

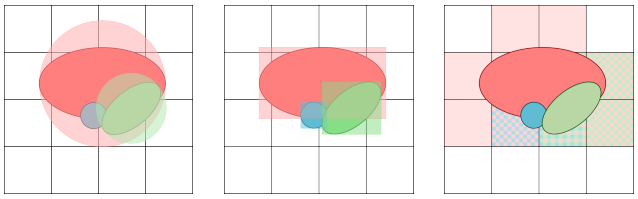
$$\left(\boldsymbol{\mu}_g^{t+1}, \boldsymbol{L}_g^{t+1}\right) = \left(\boldsymbol{\mu}_g^t, \boldsymbol{L}_g^t\right) - \gamma \boldsymbol{F}_g^{-1} \nabla \mathcal{L}\left(\boldsymbol{\mu}_g^t, \boldsymbol{L}_g^t\right). \tag{6}$$

In comparison to preconditioners applied by AdaGrad-inspired optimizers [6, 18], neither of these preconditioning matrices are constrained to be diagonal, allowing them to capture more of the local geometry of each Gaussian. Furthermore, while optimizers like Shampoo [12] and SOAP [33] instead utilize block diagonal preconditioning matrices, they recalculate the preconditioning matrices only for a subset of gradient updates for efficiency.

3.4.2. Implicit to Explicit 3D Knowledge Distillation

Our second approach is to first directly supervise our 3D Gaussian Point Encoder via a pre-trained PointNet-style per-point embedding. For our PointNet classification experiments, we perform this in three stages. Initially, we optimize the first Gaussian Basis Encoder, which serves as a T-Net, by sampling random points from the minimum bounding rectangular prism of the training set (*e.g.*, the unit cube). We then minimize the L_1 loss between the per-point embeddings of PointNet's T-Net and those generated by the Gaussian Basis Encoder, prior to maxpooling.

Next, we enhance the 3D Gaussian Basis T-Net by incorporating a copy of the transform regressor MLP from the pre-trained T-Net. Once the per-point encodings for the T-Net are aligned, we proceed to optimize the main Gaussian Basis Encoder, which replaces the PointNet encoder. This optimization follows a similar process: sampling points from the bounding rectangular prism, computing the transformed points via each encoder's T-Net, and minimizing the L_1 loss between the per-point embeddings produced by each encoder.



(a) Distance Filtering (b) Bounding-box Filtering (c) Voxel Filtering

Figure 3. **Pairwise Gaussian-Point Filtering.** (a) Distance filtering only evaluates Gaussian-Point pairs within a radius of a Gaussian's mean. (b) Bounding-box Filtering evaluates Gaussian-Point pairs when a point falls within the axis-aligned bounding box center on a Gaussian. (c) Voxel Filtering evaluates Gaussian-Point pairs when a point lies in a voxel occupied with sufficiently high likelihood by a given Gaussian.

After this distillation phase, the model is trained end-to-end on the training set, utilizing a copy of the parent model's classifier. To preserve the learned per-point embeddings, we apply a reduced learning rate to both the T-Net encoder and the main encoder parameters, preventing significant deviations in their representations.

For our Mamba3D classification experiments, we only need two stages, as there is only one per-point embedding to distill. We optimize our 3DGPE network, which serves as a patch encoder, by instead sampling point clouds from the training dataset, and applying farthest point sampling and KNN-based grouping to generate in-distribution point patches. Similar to the PointNet case, we aim to minimize L_1 loss of the encoders' per-point embedding, following this up with end-to-end training with a copy of the parent model's middle encoder and classifier.

3.5. Filtering via Explicit 3D Geometry

Inspired by the pruning and filtering techniques in 3DGS [5, 7, 8, 22], we introduce Pairwise Gaussian-Point Filtering at inference-time in our encoder to further improve computational efficiency.

Computing the Mahalanobis distance has quadratic complexity in input dimension, making it relatively expensive. However, our experiments also reveal that a sizable percentage of the calculated Gaussian likelihoods are very small (see Fig. 2 in the supplemental). If the likelihood is sufficiently small, we can potentially filter it out and instead assume it to be zero. This requires a heuristic that is significantly faster to compute than Mahalanobis distance. We experiment with three heuristics:

(i) **Distance Filtering.** We compute the Euclidean distance to each Gaussian's mean, which only requires linear complexity in dimension as opposed to quadratic. We then threshold the distances by $2\lambda_g \log\left(\frac{\alpha_{g,\max}}{t_{\text{distance}}}\right)$, where λ_g is the largest eigenvalue

of the covariance matrix. We only evaluate the likelihood for Gaussian-point pairs below this distance. In essence, this method bounds an anistropic Gaussian with an isotropic Gaussian. This is variant of the method used by 3DGS [17].

(ii) **Bounding-box Filtering.** We compute the minimal axis-aligned bounding box for each Gaussian confidence ellipsoid given a threshold t_{bbox}, which requires bilinear computational cost in input dimension and number of Gaussians. Then, we check to see if a point falls within a bounding box before computing its likelihood. This is closely related to the "SnugBox" technique proposed by Hanson et al. [14] for 3DGS, except extended to arbitrary dimensions rather than 2D.

(iii) **Voxel Filtering.** We coarsely voxelize the input volume by D_{voxel} in each dimension and pre-compute the maximum weighted likelihood of each Gaussian for points falling within each voxel. We cache the list of Gaussians with weighted likelihood above a threshold t_{voxel}, and at runtime we only compute the likelihood of Gaussian-point pairs for each point's voxel's Gaussian list. This is related to the "AccuTile" technique proposed by Hanson et al. [14] for 3DGS, except we pre-compute the weighted likelihoods rather than derive them with an iterative algorithm.

Each of these methods comes with various advantages and disadvantages. Method (i) benefits from re-using the computation of the difference between the points and the means, but is a poor heuristic if the Gaussians are highly anistropic. Method (ii) requires extra computation to determine bounding box occupancy, but more tightly encloses highly anistropic Gaussians than (i). Finally, method (iii) can be implemented with very low computational costs at runtime using by a lookup table for the Gaussian lists and is the most accurate heuristic given a large enough D_{voxel}, but similar to LUTI-MLP [28], has exponential memory requirements in input dimension.

3.6. Implementation Details

We implement our encoder in pure PyTorch, using [37] as a reference PointNet implementation for distillation experiments. Since our 3D Gaussian Point Encoder does not employ a feature transform (only an input transform via T-Net), we modify the PointNet implementation to remove the feature transform. We utilize Mamba3D's official code and checkpoints for our Mamba3D experiments.

When training both our PointNet and Mamba3D variants end to end with natural gradients, we utilize SGD for the Gaussian parameters with a learning rate of 0.005 on the means and 0.005 on the Cholesky factors, while using AdamW for the rest of the network. During distillation, we train all components of our models using AdamW with a learning rate of 1.6×10^{-3} for the means, 5×10^{-4} for the diagonal Cholesky elements, and 1×10^{-4} for the lower triangular Cholesky elements, mixture coefficients, and biases. MLPs used for 3D Gaussian T-Nets and classifiers utilize a learning rate of 1×10^{-4}. The learning rates for the encoder parameters are reduced by a factor of 100 when fine-tuning following the initial distillation.

4. Experiments

4.1. Shape Classification with PointNet and Mamba3D

We benchmark our encoder on shape classification using the ModelNet40 [36] and ScanObjectNN [31] datasets. ModelNet40 consists of 9,843 training and 2,468 testing meshes of axis-aligned CAD models across 40 classes. We utilize the hardest "PB T50 RS" variant of ScanObjectNN, consisting of 11,416 training and 2,882 testing real-world 3D scans across 15 object classes. For both datasets, we reserve 25% of the training samples as validation data for our ablations and hyperparameter selection. Following common practice, we report both class-averaged accuracy (mAcc.) and overall accuracy (OA) as our metrics. Furthermore, we measure FLOPs using FVCore. To gauge performance on varying hardware platforms, we measure GPU and CPU latency using PyTorch's profiler. GPU latency is measured on a single RTX 4070 Mobile GPU with a point cloud size of 2048 points, while CPU latency is measured on a low power ARM CPU (Rockchip RK3588) for methods that do not require custom CUDA extensions.

4.1.1. Shape Classification Baselines

For our PointNet experiments, we primarily compare against a PointNet with both an input transform and feature transform, as well as GPointNet, LUTI-MLP, and a PointNet pruned according to [2]. For our Mamba3D experiments, we instead compare against other Transformer and Mamba based architectures, including VecKM [41] as a patch encoder. Additionally, we include hierarchical architectures in PointNet++ [26], PointMLP [21], and PointNeXT [27], as well as a near state-of-the-art method in DeLA [3] for reference. All methods are evaluated without voting or cross-modal pre-training. We utilize rotation around the vertical axis for ScanObjectNN, and scaling by $\pm$ 20% and translation by Gaussian noise with a standard deviation of 0.01 for ModelNet40. For both PointNet and Mamba3D experiments, we set N_G to 32 and we utilize the Mahalanobis distance natural gradient. Our filtered PointNet model utilizes bounding-box filtering at test time with t_{bbox} of 0.10.

4.1.2. Comparison to PointNet-like Architectures

All PointNet-style classifiers perform comparably on ModelNet40. However, both GPointNet and LUTI-MLP un-

Table 1. **Shape Classification Results.** FLOPs and Latency are computed per sample on ScanObjectNN with 2048 input points. X indicates that the model cannot be run on CPU, N indicates end-to-end with natural gradients, D indicates distilled, F indicates filtered. * indicates weights are not publicly available, so we cannot directly compare memory and latency.

Method	ModelNet40		ScanObjectNN		FLOPs	Params	GPU Latency	CPU Latency	Memory
	mAcc. (%)	OA (%)	mAcc. (%)	OA (%)	(G)	(M)	(ms)	(ms)	(MB)
PointNet-Like Architectures									
PointNet [25]	86.1	90.0	65.2	69.0	0.891	3.47	1.00	110.2	1057
PointNet (no FT)	**86.4**	90.2	65.3	69.3	0.582	1.61	0.62	69.3	1039
GPointNet [29]	84.3	89.2	58.4	61.5	0.052	1.34	14.81	396.7	2747
LUTI-MLP [28]	85.9	88.0	60.9	63.4	**0.032**	**1.03**	3.67	258.9	4839
Pruned PointNet* [2]	-	88.2	-	**71.7**	-	1.36	-	-	-
3DGPE (N)	**86.4**	90.1	65.5	69.0	0.068	1.39	0.44	45.7	**573**
PointNet → 3DGPE (D)	86.1	**90.3**	65.3	69.1	0.068	1.39	0.44	45.7	**573**
PointNet → 3DGPE (D + F)	85.3	89.8	**65.8**	69.2	0.064	1.39	**0.36**	**37.6**	605
Dedicated & Hierarchical Architectures									
PointNet++ [26]	91.8	89.1	76.0	77.8	1.68	1.5	5.9	403.4	1215
PointMLP [21]	91.3	**94.1**	83.9	85.4	31.4	12.6	7.7	X	1801
PointNeXT [27]	90.8	93.2	85.8	87.7	1.6	**1.4**	1.8	X	1257
DeLA [3]	**92.2**	94.0	**89.3**	**90.4**	**1.5**	5.3	**0.9**	X	**1177**
SimpleView [9]	-	93.9	-	80.5	-	-	-	X	-
Transformer and Mamba Architectures									
PCT [11]	-	93.2	-	-	2.3	**2.9**	14.8	X	6677
PCM [42]	**90.7**	93.4	86.6	88.1	45.0	34.2	31.4	X	5533
PointMamba [20]	-	92.4	-	84.9	3.1	12.3	8.3	X	1510
Mamba3D [13]	89.7	93.3	**90.6**	**91.6**	3.9	16.9	10.4	X	1413
VecKM + Mamba3D [13, 41]	90.5	93.4	86.1	88.8	5.1	16.4	9.2	X	1738
3DGPE + Mamba3D (N)	89.9	**93.6**	86.4	88.0	1.8	16.5	8.2	X	**817**
3DGPE + Mamba3D (D)	89.8	93.5	86.6	88.5	1.8	16.5	8.2	X	**817**
3DGPE + Mamba3D (D + F)	89.6	93.3	86.0	88.3	**1.8**	16.5	**7.8**	X	853

derperform PointNet on ScanObjectNN compared to PointNet by approximately 7.8 and 5.9 percentage points respectively. We hypothesize that GPointNet's relatively low performance arises from its inability to model complex activation volumes, potentially making it harder to deal with the large perturbations present in ScanObjectNN. LUTI-MLP's lower performance may be also be a result of its modified T-Net, as it uses a $\tanh$ activation to constrain point clouds to fit in the unit cube, potentially resulting in deformation that interferes with its interpolation. In comparison, our 3D Gaussian Point Encoder performs comparably to PointNet, achieving the 2nd highest accuracy.

Overall, we find that the 3D Gaussian Point Encoder with PointNet achieves the lowest latency out of all the models tested, achieving approximately **2.7**× the throughput of a standard PointNet on a mobile GPU and **2.9**× on a lower power CPU. Interestingly, despite the fact that both GPointNet and LUTI-MLP have lower FLOPs counts, both methods have substantially higher latency. Our latency advantage over these methods also holds on CPU, where both GPointNet and LUTI-MLP become prohibitively expensive, with throughputs under 4 samples per second. In the case of LUTI-MLP, this may be a result of the indexing operations required for interpolating the lookup table only being efficient with custom CUDA kernels.

4.1.3. Comparison to Advanced Architectures

Among the Transformer and Mamba architectures, all methods perform comparably on ModelNet40. On ScanObjectNN, Mamba3D performs the best, with our 3D Gaussian Point Encoder performing similarly to PCM. Nonetheless, in comparison to these architectures, our model achieves the lowest FLOPs, latency, and memory. In fact, our model achieves the second lowest memory usage across all model types, highlighting how impactful the encoder design can be towards total memory usage. Compared to Mamba3D, our encoder reduces FLOPs by approximately **54%** and memory by **42%**, while increasing throughput by **1.27**×. This performance advantage holds when compared to VecKM as a patch encoder, though to a lesser extent.

4.2. Ablations

We ablate the impact of training each of the Gaussian parameters as well as optimization methods. All ablations are carried out on ScanObjectNN with evaluation performed on the validation set. An additional ablation on filtering meth-

ods is included in Sec. 2 in the supplemental.

Table 2. **Comparisons on N_G and Optimization Methods.** Results are class-averaged accuracies on the validation split of ScanObjectNN averaged over 10 trials, listed alongside standard deviation. X denotes incompatibility. Mahalanobis and Fisher refer to the Mahalanobis distance and Fisher information metric natural gradients respectively.

	mAcc. (%)				
N_G	Distill	Mahalanobis	Fisher	Adam	SOAP
3DGPE					
16	68.6 ± 1.0	65.9 ± 5.2	66.3 ± 4.1	43.7 ± 28.1	60.0 ± 6.8
24	71.1 ± 4.4	72.4 ± 3.6	69.9 ± 5.1	65.8 ± 4.3	68.3 ± 5.3
32	81.6 ± 4.9	78.2 ± 4.3	77.4 ± 2.4	52.4 ± 23.6	72.9 ± 6.5
64	82.8 ± 3.7	81.3 ± 4.2	79.5 ± 3.1	78.4 ± 5.0	77.4 ± 7.2
3DGPE + Mamba3D					
16	84.3 ± 2.4	85.1 ± 2.2	82.6 ± 8.5	76.7 ± 5.3	X
24	85.8 ± 3.2	85.7 ± 3.7	83.8 ± 4.6	77.5 ± 6.9	X
32	87.6 ± 2.0	86.9 ± 1.5	87.5 ± 2.1	78.6 ± 6.6	X
64	88.2 ± 2.1	87.6 ± 2.6	87.2 ± 4.2	81.4 ± 7.6	X

4.2.1. Varying the Number of Gaussians

We report the validation accuracies as we tweak the number of Gaussians, N_G in Tab. 2. Intuitively, performance generally increases as N_G increases, as the Gaussian mixtures are better able to approximate PointNet's activation volumes. However, the performance does not significantly improve when increasing N_G from 32 to 64.

4.2.2. Optimization Techniques

To validate the impact of natural gradients and distillation, we train both of our 3D Gaussian Point Encoder-based models from scratch with Adam [18], and only our PointNet model with SOAP [33], as Mamba3D immediately returns NaN loss with it. In Tab. 2 we demonstrate that training both models end-to-end with Adam necessitates a substantial increase in N_G to achieve acceptable performance, consequently resulting in increased computational cost. Moreover, we observe that, for most values of N_G, models trained end-to-end with standard optimizers exhibit significantly higher variability, and even their best-performing trials consistently underperform compared to trials utilizing either PointNet guidance or natural gradients. We hypothesize that this elevated variance arises from both the limited number of tunable parameters, which makes the optimization process more fragile, and heightened sensitivity to parameter initialization, especially with respect to the Gaussian means. We believe the preconditioned mean updates from both natural gradient methods, and to a lesser extent, SOAP, allow them to mitigate some of this sensitivity.

4.2.3. Trainable Parameters

We experiment with fixing the means, lower triangular covariance entries, and diagonal covariance entries of each of the Gaussians in our PointNet experiments. Fixing the lower triangular elements to zero makes the Mahalanobis Distance calculation more efficient but constrains the Gaussians to be axis-aligned, while fixing all the covariance entries constrains all Gaussians to have identity covariances. The results of this experiment are shown in Tab. 3. In general, we find that all three Gaussian parameters contribute strongly to the model performance, with an especially sharp reduction in performance with diagonal covariance.

Table 3. **Ablations on Trainable Gaussian Parameters.** Results are class-averaged accuracies on the validation split of ScanObjectNN, and are averaged over 5 training runs with distillation. Performance generally decreases as more parameters are fixed.

	Trainable Parameters			
N_G	Mean	L. Triang.	Diag.	mAcc. (%)
16	✓	✓	✓	68.6
	✗	✓	✓	64.3 (-4.3)
	✗	✗	✓	59.9 (-8.7)
	✗	✗	✗	60.6 (-8.0)
32	✓	✓	✓	81.6
	✗	✓	✓	62.2 (-19.4)
	✗	✗	✓	66.0 (-15.6)
	✗	✗	✗	65.5 (-16.1)
64	✓	✓	✓	82.8
	✗	✓	✓	75.4 (-7.4)
	✗	✗	✓	72.2 (-10.6)
	✗	✗	✗	69.7 (-13.1)

5. Discussion and Conclusion

In this paper, we introduced the *3D Gaussian Point Encoder*, a novel point embedding architecture inspired by the explicit geometry of 3D Gaussian Splatting. Our experiments demonstrate that, when trained via natural gradients or 3D Knowledge Distillation, the 3D Gaussian Point Encoder achieves performance comparable to PointNet while significantly surpassing it in computational efficiency, delivering **2.7**× higher throughput while using **46%** less memory. Furthermore, the encoder integrates well into modern architectures like Mamba3D, improving throughput by **1.27**× and reducing memory by **42%**.

5.1. Limitations

Our 3D Gaussian Point Encoder requires more careful optimization techniques than PointNet, and will likely not be suitable in cases where PointNet embeddings do not perform adequately. On Mamba3D experiments, 3DGPE was unable to optimize to the full level of performance as the original model. Furthermore, we only focus on classification. Higher dimensional inputs, such as decorators used in semantic segmentation and detection, may present unexpected challenges in fitting the Gaussian representation.

References

[1] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998. 2, 4

[2] Amrijit Biswas, Md Ismail Hossain, MM Elahi, Ali Cheraghian, Fuad Rahman, Nabeel Mohammed, and Shafin Rahman. 3d point cloud network pruning: When some weights do not matter. *arXiv preprint arXiv:2408.14601*, 2024. 6, 7

[3] Binjie Chen, Yunzhou Xia, Yu Zang, Cheng Wang, and Jonathan Li. Decoupled local aggregation for point cloud learning. *arXiv preprint arXiv:2308.16532*, 2023. 6, 7

[4] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning (ICML)*, 2024. 2

[5] Stavros Diolatzis, Tobias Zirr, Alexander Kuznetsov, Georgios Kopanas, and Anton Kaplanyan. N-dimensional gaussians for fitting of high dimensional functions. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11, 2024. 1, 3, 5

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011. 2, 5

[7] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, Zhangyang Wang, et al. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *Advances in neural information processing systems*, 37: 140138–140158, 2025. 3, 5

[8] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. In *European Conference on Computer Vision*, pages 165–181. Springer, 2024. 1, 3, 5

[9] Ankit Goyal, Hei Law, Bowei Liu, Alejandro Newell, and Jia Deng. Revisiting point cloud shape classification with a simple and effective baseline. In *International conference on machine learning*, pages 3809–3820. PMLR, 2021. 7

[10] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. 2

[11] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. Pct: Point cloud transformer. *Computational visual media*, 7(2):187–199, 2021. 2, 7

[12] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pages 1842–1850. PMLR, 2018. 2, 5

[13] Xu Han, Yuan Tang, Zhaoxuan Wang, and Xianzhi Li. Mamba3d: Enhancing local features for 3d point cloud analysis via state space model. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 4995–5004, 2024. 2, 3, 4, 7

[14] Alex Hanson, Allen Tu, Geng Lin, Vasu Singla, Matthias Zwicker, and Tom Goldstein. Speedy-splat: Fast 3d gaussian splatting with sparse pixels and sparse primitives. *arXiv preprint arXiv:2412.00578*, 2024. 1, 3, 6

[15] Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein. Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. *arXiv preprint arXiv:2406.10219*, 2024. 1, 3

[16] Andrew Jones. Natural gradients. https://andrewcharlesjones.github.io/journal/natural-gradients.html. Accessed: 2025-09-19. 5

[17] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. 1, 2, 6

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015. 2, 5, 8

[19] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019. 1

[20] Dingkang Liang, Xin Zhou, Wei Xu, Xingkui Zhu, Zhikang Zou, Xiaoqing Ye, Xiao Tan, and Xiang Bai. Pointmamba: A simple state space model for point cloud analysis. *Advances in neural information processing systems*, 37:32653–32677, 2025. 2, 7

[21] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking network design and local geometry in point cloud: A simple residual mlp framework. *arXiv preprint arXiv:2202.07123*, 2022. 1, 2, 6, 7

[22] Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. *arXiv preprint arXiv:2403.13806*, 2024. 1, 2, 3, 5

[23] Yatian Pang, Eng Hock Francis Tay, Li Yuan, and Zhenghua Chen. Masked autoencoders for 3d point cloud self-supervised learning. *World Scientific Annual Review of Artificial Intelligence*, 1:2440001, 2023. 2

[24] Jooyoung Park and Irwin W Sandberg. Approximation and radial-basis-function networks. *Neural computation*, 5(2): 305–316, 1993. 2

[25] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 2, 3, 7

[26] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 1, 6, 7

[27] Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. Pointnext: Revisiting pointnet++ with improved training and scaling strategies. *Advances in neural information processing systems*, 35:23192–23204, 2022. 2, 6, 7

[28] Yusuke Sekikawa and Teppei Suzuki. Tabulated mlp for fast point feature embedding. *arXiv preprint arXiv:1912.00790*, 2019. 1, 2, 6, 7

[29] Teppei Suzuki, Keisuke Ozawa, and Yusuke Sekikawa. Rethinking pointnet embedding for faster and compact model. In *2020 International Conference on 3D Vision (3DV)*, pages 791–800, 2020. 1, 2, 3, 7

[30] Linda S L Tan. Analytic natural gradient updates for cholesky factor in gaussian variational approximation. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 2025. 5

[31] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *International Conference on Computer Vision (ICCV)*, 2019. 2, 6

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2

[33] Nikhil Vyas, Depen Morwani, Rosie Zhao, Mujin Kwun, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. Soap: Improving and stabilizing shampoo using adam, 2025. 2, 5, 8

[34] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point transformer v2: Grouped vector attention and partition-based pooling. *Advances in Neural Information Processing Systems*, 35:33330–33342, 2022. 2

[35] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point transformer v3: Simpler faster stronger. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4840–4851, 2024. 2

[36] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 2, 6

[37] Xu Yan. Pointnet/pointnet++ pytorch. `https://github.com/yanx27/Pointnet_Pointnet2_pytorch`, 2019. GitHub repository. 6

[38] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 1

[39] Zhifan Ye, Chenxi Wan, Chaojian Li, Jihoon Hong, Sixu Li, Leshu Li, Yongan Zhang, and Yingyan Celine Lin. 3d gaussian rendering can be sparser: Efficient rendering via learned fragment pruning. *Advances in Neural Information Processing Systems*, 37:5850–5869, 2025. 1, 3

[40] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. 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*, pages 19313–19322, 2022. 2

[41] Dehao Yuan, Cornelia Fermüller, Tahseen Rabbani, Furong Huang, and Yiannis Aloimonos. A linear time and space local point cloud geometry encoder via vectorized kernel mixture (veckm). In *Proceedings of the 41st International Conference on Machine Learning*. JMLR.org, 2024. 2, 6, 7

[42] Tao Zhang, Haobo Yuan, Lu Qi, Jiangning Zhang, Qianyu Zhou, Shunping Ji, Shuicheng Yan, and Xiangtai Li. Point cloud mamba: Point cloud learning via state space model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 10121–10130, 2025. 2, 7

[43] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip H.S. Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16259–16268, 2021. 2

[44] Jianqiao Zheng, Xueqian Li, Sameera Ramasinghe, and Simon Lucey. Robust point cloud processing through positional embedding. In *2024 International Conference on 3D Vision (3DV)*, pages 1403–1412, 2024. 1, 2

[45] Kun Zhou, Zhong Ren, Stephen Lin, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Real-time smoke rendering using compensated ray marching. In *ACM SIGGRAPH 2008 papers*, pages 1–12. 2008. 2

[46] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4490–4499, 2018. 1