Accelerated Generative Models for 3D Point Cloud Data

Ben Eckart[1,2] Kihwan Kim[2] Alejandro Troccoli[2] Alonzo Kelly[1] Jan Kautz[2]
[1]The Robotics Institute, Carnegie Mellon University [2]NVIDIA Research

https://research.nvidia.com/publication/accelerated-generative-models

Abstract

Finding meaningful, structured representations of 3D point cloud data (PCD) has become a core task for spatial perception applications. In this paper we introduce a method for constructing compact generative representations of PCD at multiple levels of detail. As opposed to deterministic structures such as voxel grids or octrees, we propose probabilistic subdivisions of the data through local mixture modeling, and show how these subdivisions can provide a maximum likelihood segmentation of the data. The final representation is hierarchical, compact, parametric, and statistically derived, facilitating run-time occupancy calculations through stochastic sampling. Unlike traditional deterministic spatial subdivision methods, our technique enables dynamic creation of voxel grids according the application's best needs. In contrast to other generative models for PCD, we explicitly enforce sparsity among points and mixtures, a technique which we call expectation sparsification. This leads to a highly parallel hierarchical Expectation Maximization (EM) algorithm well-suited for the GPU and real-time execution. We explore the trade-offs between model fidelity and model size at various levels of detail, our tests showing favorable performance when compared to octree and NDT-based methods.

1. Introduction

Given the recent commoditization of different types of active range sensors (*e.g.*, TOF, Lidar, structured light), spatial processing and visualization of large collections of 3D point clouds has become one of the most important stages in 3D imaging/vision pipelines [15]. 3D point cloud processing introduces several new challenging problems such as (1) uneven sampling density, (2) unstructured organization of the incoming data, (3) level-of-detail processing given varying speed and memory requirements, and (4) measurement uncertainty from sensor noise. Additionally, modern depth sensors generate millions of data points per second, making it difficult to utilize all incoming data effectively in real-time for devices with limited computational resources.

Many current techniques for processing large amounts of point cloud data (PCD) either simply subsample the data or apply some sort of discretization, either through dense, sparse [16] or hierarchical [7] voxelization techniques. Representing continuous geometry through voxels creates discretization artifacts and offers no clear way of handling noise or data uncertainty. Furthermore, the discrete nature of voxels and sub-sampled point clouds greatly complicate spatial processing procedures that require continuous derivatives or high quality normal estimates.

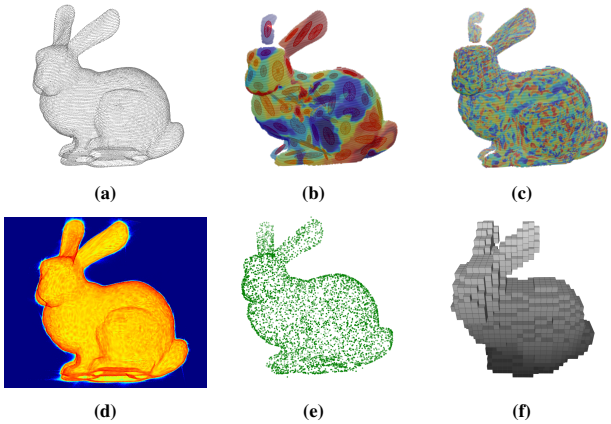


Figure 1. **Processing PCD with a Hierarchy of Gaussian Mixtures:** (a) Raw PCD from Stanford Bunny (*35k vertices*), (b) and (c) Two levels of detail extracted from the proposed model. Each color denotes the area of support of a single Gaussian and the ellipsoids indicate their one σ extent. Finer grained color patches therefore indicate higher statistical fidelity but larger model size, (d) a log-scale heat-map of a PDF from a high fidelity model. (e) stochastically re-sampled PCD from the model (*5k points*), (f) occupancy grid map also derived directly from the model.

We address these challenges with a hierarchical and probabilistic representation of 3D point cloud data (PCD) in the form of a hierarchy of Gaussian Mixture Models (GMMs). As a representation of 3D space, a GMM model has several advantages. First, being a continuous probability density function (PDF), the GMM does not require the discretization of 3D space. Second, the uncertainties of data measurements are embedded in the covariance matrices of the GMM, which combined with a special cluster to handle outliers, provide an effective way of handling noisy mea-

surements. Finally, the storage requirements for a GMM are much lower than for the original PCD.

Though GMMs have been used before for PCD representation [4, 8], we introduce a novel top-down hierarchical model which confers the following benefits: (1) dynamic allocation of the number of mixtures, with new clusters being added in areas of high-frequency detail, (2) efficient parallel coarse-to-fine construction by recursively partitioning the points in the PCD into their most influential mixture/s, and (3) multiple levels of detail for point cloud re-sampling and occupancy map creation.

Many applications in 3D vision require grid-based occupancy estimates of space, including path planning [19], semantic perception [6], and 3D modeling [11]. We show how our model may augment these applications by allowing dynamic run-time estimates of occupancy over sparse grids. Since the spatial extent and voxel resolution of these estimates can be determined dynamically at run-time, one can avoid many of the common problems with traditional techniques: constrained extent in large scenes, discretization artifacts as a result of coarse voxel sizes, or memory bottlenecks resulting from dense high-resolution voxel grids.

Although a generative or continuous parametric representation for PCD can facilitate many important applications such as registration, surface extraction, semantic segmentation, this is not our focus. Instead, we focus on a more basic and fundamental problem: how one might efficiently construct, sample, and integrate over these generative models. This work therefore can be used to augment all the aforementioned applications.

Our main contribution is a highly efficient and parallelizable method for hierarchical top-down GMM clustering that, as opposed to previous GMM-based techniques, applies sparse constraints on point to cluster assignments, thus enabling construction time logarithmic with respect to the overall model size. In addition, we present a novel importance sampling technique that allows for efficient integration of the PDF over a discretized volume of space that can be used to construct arbitrarily sized probabilistic occupancy maps.

2. Related Work

In most spatial processing applications that rely on point cloud data, using the raw points directly can be nearly intractable. Thus, most common operations one might want to perform: nearest neighbor queries, denoising, geometric or semantic inference, etc., stand to benefit from imposing some type of structure to the raw data. Table 1 summarizes typical data structures used for point cloud data.

Voxelization and occupancy grids [6, 19] have been established as a popular method to discretize raw PCD over a dense grid, but memory problems emerge when needing fine resolution or large grids. Especially in cases of 3D points, many voxels may be unoccupied, leading to inefficient memory usage. Octrees and kd-trees can be much more space efficient [11], as the construction of a regularly subdivided hierarchy effectively compresses empty space. These structures incur additional overhead compared to dense grids, however, requiring superlinear construction time with respect to the size of the PCD.

Data Structure	Hierarchical	Generative	Voxel Free	Construction Complexity
Voxel Hash List [17]				N
Octree [11]	✓			$N \log N$
3D-NDT [1]		✓		N
3D-NDT-Octree [14]	✓	✓		$N \log N$
GMM [4]		✓	✓	NJ
Hierarchical GMM [10]	✓	✓	✓	N^2
Proposed Method	✓	✓	✓	$N \log J$

Table 1. **A Comparison of 3D Point Cloud Data Structures** *Hierarchical*: Hierarchical methods compress free space and are therefore more compact than dense grids. *Generative*: Generative models add parametric structure to PCD, facilitating statistical inference, maximum likelihood, or continuous optimization methods. *Voxel Free*: The lack of voxelization present in the model avoids discretization errors, allowing higher fidelity at smaller model sizes. *Construction complexity*: N is the number of points in the PCD, and J the number of mixtures in a GMM, with $J \ll N$ for most typical applications.

Whereas voxels and octrees rely on discretization to obtain structure from PCD, another class of algorithms instead model the data as a set of independent samples from some unknown distribution. These algorithms use the principle of maximum data likelihood to optimize a set of latent parameters that describe the original PCD. Known as generative models, these models can by construction provide robust probabilistic inference. Their trade-off, however, is the potentially high construction or inference cost and need for *a priori* knowledge.

For modeling 3D PCD, the most common generative model used in the literature is the Gaussian Mixture Model (GMM). Typically, GMMs are used to facilitate robust point cloud registration techniques [1, 4, 5, 12, 8]. The work of Jian and Vemuri [12], for example, convert a point cloud into a GMM by placing a covariance around each point. Though this minimizes the setup cost, inference then becomes very slow as the GMM is larger than the original raw points. Others, such as Eckart *et al.* [4, 5] perform a maximum data likelihood optimization during the construction of the model in order to reduce its size, but this construction does not scale well when large amounts of mixtures are needed. In contrast to these methods, our proposed technique is exponentially faster with respect to the size of the model. A "flat" version must iterate linearly $O(J)$ through all J mixtures, whereas our method is $O(\log J)$. Furthermore, we do not need to specify the number of mixtures *a priori*. Our coarse-to-fine construction allows us to both fil-

ter out low-frequency details quickly (floors and walls) and drill down to deeper levels for high-frequency details.

The Normal Distributions Transform (NDT) [1, 18] is a widely used and elegant technique that attempts to merge the concepts of a voxel grid or octree with a GMM by simply recording the mean and covariance of all points that fall into each voxel. The GMM can then be constructed as a weighted sum of the voxel's respective Gaussian parameters. Though the construction of such a data structure is very efficient, the requirement to voxelize at the beginning can cause a loss of fidelity.

Other work has experimented with hierarchical forms of GMMs for applications like 2D image segmentation [9]. Typically, these methods operate bottom-up, repeatedly grouping together like clusters of points using divergence measures to split and merge the data. For example, Goldberger *et al*. [10] construct an iterative EM-like algorithm using KL-Divergence in order to repeatedly merge candidate clusters. In contrast, we adopt a top-down hierarchical approach, motivated by the need to keep the calculation of point-mixture correspondences sparse for 3D point clustering. As such, our approach is more amenable to parallel hardware and is much more computationally efficient (see Table 1). Another similar top-down construction is that of Kalaiah *et al*. [13], though this method is not generative.

3. Method Overview

Our model uses overlapping basis functions (anisotropic Gaussian mixtures) for representing 3D geometry. These functions are recursively applied in a top-down fashion to create a hierarchy of overlapping patches that approximate the original 3D PCD. The creation of this model is cast as the solution to a Maximum Likelihood Estimation (MLE) hierarchical Gaussian Mixture segmentation problem that can be solved by recursively employing the Expectation Maximization (EM) algorithm over increasingly smaller partitions of the point data.

3.1. Model Definition

Our world model is composed of J overlapping probabilistic mixtures $\mathbf{\Theta}_j$ plus a (J+1)$^{\text{th}}$ noise distribution. We choose our J mixtures to be weighted 3-dimensional multivariate Gaussians, $\mathbf{\Theta}_j = \{\pi_j, \boldsymbol{\mu}_j, \mathbf{\Sigma}_j\}$, with π_j being the weight and $\boldsymbol{\mu}_j$ and $\mathbf{\Sigma}_j$ being the mean and covariance, respectively. Our noise distribution is chosen to be a uniform distribution over the bounding box of the point data. Together, these basis distributions produce a mixture model, which is itself a valid probability distribution.

Given a point cloud $\mathcal{Z}$ of size N, its probability of being generated by our model, given that each point is an *iid* sample of the world, is:

$$p(\mathcal{Z}|\mathbf{\Theta}) = \prod_{i=1}^{N} p(\mathbf{z}_i|\mathbf{\Theta}) = \prod_{i=1}^{N}\sum_{j=1}^{J+1} \pi_j p(\mathbf{z}_i|\mathbf{\Theta}_j), \quad (1)$$

$$p(\mathbf{z}_i|\mathbf{\Theta}_j) = \begin{cases} \mathcal{N}(\mathbf{z}_i|\mathbf{\Theta}_j), & \text{for } 1 \le j \le J, \\ \frac{1}{\eta}, & \text{for } j = J+1, \end{cases} \quad (2)$$

where η is the size of the volume for which the noise cluster is active.

To find the basis functions to best fit the point cloud data we employ the EM algorithm [2], which has been established as a way to iteratively maximize data likelihood when there is no closed form solution to the maximizer, yet there is a way of finding a maximum of joint data likelihood of the data and a set of associated latent variables. We define a set C of latent variables c_{ij} that represents the binary associations between points $\mathbf{z}_i \in \mathcal{Z}$ and mixtures $\mathbf{\Theta}_j$. In the *E-Step*, we calculate the posterior for all $c_{ij} \in \mathcal{C}$ given $\mathbf{\Theta}$:

$$E[c_{ij}] = \frac{\pi_j p(\mathbf{z}_i|\mathbf{\Theta}_j)}{\sum_{j'=1}^{J+1} \pi_{j'} p(\mathbf{z}_i|\mathbf{\Theta}_{j'})} \quad (3)$$

In the *M-Step*, we maximize the expected log-likelihood with respect to $\mathbf{\Theta}$, using our current $E[c_{ij}] \stackrel{\text{def}}{=} \gamma_{ij}$:

$$\max_{\mathbf{\Theta}} \sum_{ij} \gamma_{ij} \{\ln \pi_j + \ln p(\mathbf{z}_i|\mathbf{\Theta}_j)\} \quad (4)$$

Given a fixed set of expectations, one can solve for the optimal parameters in closed form at iteration k:

$$\boldsymbol{\mu}_j^{k+1} = \frac{\sum_i \gamma_{ij}\mathbf{z}_i}{\sum_i \gamma_{ij}} \quad (5)$$

$$\mathbf{\Sigma}_j^{k+1} = \frac{\sum_i \gamma_{ij}\mathbf{z}_i\mathbf{z}_i^T}{\sum_i \gamma_{ij}} - \boldsymbol{\mu}_j^{k+1}\boldsymbol{\mu}_j^{k+1^T} \quad (6)$$

$$\pi_j^{k+1} = \sum_i \frac{\gamma_{ij}}{N} \quad (7)$$

3.2. Expectation Sparsity

Given the above definitions and a sufficiently high number of mixtures J, the posterior over correspondences will be sparse due to the nature of 3D geometry. We can see this fact intuitively: Consider that in an indoor scene, for example, the geometric structure of a light fixture will not be statistically informative to a point sampled on a couch beneath it. Thus, given a point cloud of size N, if we naively try to calculate all NJ point-subsurface expectations (γ_{ij}), most will be zero or near-zero and not contribute meaningfully to the calculation. Therefore, one could save vast amounts of computation when trying to calculate γ_{ij} by restricting the summation to only those $\{\mathbf{z}_i, \mathbf{\Theta}_j\}$ tuples that are known to have sufficiently non-zero conditional probability. We show in the next section how to solve this problem by constructing a top-down hierarchy of GMMs.

3.3. A Top-Down Hierarchy of Mixtures

We can formally define our hierarchical Gaussian Mixture Model recursively by looking at the probabilistic form

for a point $\mathbf{z}_i \in \mathbb{R}^3$. At the root of our tree, level 1 ($l = 1$), our model consists of a Gaussian Mixture of size $\hat{J}$, with a $\hat{J}+1$th noise cluster:

$$p(\mathbf{z}_i|\mathbf{\Theta}^{l=1}) = \sum_{j=1}^{\hat{J}+1} \pi_j^{l=1} p(\mathbf{z}_i|\mathbf{\Theta}_j^{l=1}) \tag{8}$$

Each $\mathbf{\Theta}_j^{l=1}$ can then be refined as another Gaussian Mixture, using its correspondence variable:

$$p(z_i|c_i^{l=1}, \Theta^{l=2}) = \sum_{k=1}^{\hat{J}+1} \pi_k^{l=2|1} p(\mathbf{z}_i|\mathbf{\Theta}_k^{l=2|1}), \tag{9}$$

where the superscript indicates the selection of Gaussian parameters at level 2 given the parent node at level 1. The above is a proper Gaussian Mixture that satisfies $\sum_{k=1}^{\hat{J}+1} \pi_k^{l=2|1} = 1$. Our model is then fully defined by the set of all $\mathbf{\Theta}_k^l$ and π_k^l.

If we begin with a coarse decomposition into $\hat{J} \ll J$ mixtures, after convergence, the posterior over correspondences gives us a natural maximum likelihood partitioning of data into $\hat{J}$ coherent geometric regions. We can then use this posterior as a partitioning function over our data, and reduce our problem into $\hat{J}$ subproblems of roughly $1/\hat{J}$th the size. Recursing this process multiple times generates a tree of GMMs, requiring many small EM algorithms of size $\hat{J}$. The number of levels in the hierarchy would be $l = \log_{\hat{J}}(J)$, where each level produces $\hat{J}^{l-1}$ EM problems of size $\hat{J}$. Thus, we would need $O(\frac{J-1}{\hat{J}-1})$ EM algorithms of size $O(N_l \hat{J})$, where $N_l \approx \frac{N}{J^{*l-1}}$. The entire procedure will be logarithmic in the number of mixtures and linear in the number of points, $O(N \log_{\hat{J}}(J))$.

In order to maintain a valid global GMM, however, we need to share context between parents and children. Mathematically, we can derive this relation by assigning causal relationships to a set of l latent correspondence variables, $\mathcal{C}^l$, as depicted in Fig. 2. Using the model, we can calculate the probability of our observed variable by marginalizing over the latent variables. In the two layer case,

$$\begin{aligned} p(\mathbf{z}_i|\mathbf{\Theta}^{l=2}) &= \sum_{\mathcal{C}^{l=1}} \sum_{\mathcal{C}^{l=2}} p(\mathbf{z}_i, c_i^{l=1}, c_i^{l=2}|\mathbf{\Theta}^{l=2}) \\ &= \sum_{\mathcal{C}^{l=1}} \sum_{\mathcal{C}^{l=2}} p(\mathbf{z}_i|\mathbf{\Theta}^{l=2}, c_i^{l=2}) p(c_i^{l=2}|c_i^{l=1}) p(c_i^{l=1}) \\ &= \sum_{j'=1}^{\hat{J}+1} \sum_{j=1}^{\hat{J}+1} \pi_{j'}^{l=1} \pi_j^{l=2|1} p(\mathbf{z}_i|\mathbf{\Theta}_j^{l=2|1}) \end{aligned} \tag{10}$$

We can clearly see that for multiple levels, the correct mixing value must be propagated down the tree to the leaf node, forming a multiplicative chain.

3.4. Sparsification: Hard and Soft Partitioning

When we recurse into a new tree level, we utilize the set of posteriors γ_{ij} of the parent level to obtain a partition of our PCD. We call this process *expectation sparsification*. One possible partitioning strategy is to simply assign a point to the mixture for which its parent expectation was the highest. We will refer to this as *hard partitioning*. However, though this method retains mixture overlap inside every group of $\hat{J}$ children, we will have no such overlap among groups of children from different parents.

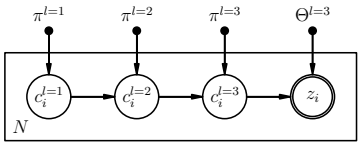


Figure 2. **Graphical model of hierarchical GMM** An example of a three-level hierarchy, where a series of causally linked latent variables are used to identify salient geometric regions of influence for each observed point $\mathbf{z}_i$.

We can use a *soft partitioning* scheme to constrain the amount of geometric context sharing among children of different parents while still maintaining logarithmic efficiency with respect to the number of mixtures. To do this, we introduce a parameter, λ_p, that relaxes the hard partitioning constraint but still keeps γ_{ij} sparse. Alg.1 describes the procedure in detail. To avoid double-counting observed points in the final GMM, we introduce a per-point weighting factor into the E-Step, called p_i. The total collection of all weights is denoted $\mathcal{P}$. ξ is a normalization constant such that $\frac{p_i}{\xi}$ over all active partitions sums to 1.0.

Algorithm 1 Expectation Sparsification Algorithm

procedure PARTITION($\mathcal{Z}$, $\mathcal{P}$, Θ, $\hat{J}$, λ_p)
  for $\mathbf{z}_i, p_i \in \{\mathcal{Z}, \mathcal{P}\}$ **in parallel do**
    calculate $\gamma_{ij}, \forall j \in \hat{J}$
    for $\gamma_{ij} \geq \lambda_p$ **do**
      add z_i to partition j as $\{z_i, \frac{p_i \gamma_{ij}}{\xi}\}$
    end for
  end for
  return partitions$_j, \forall j \in \hat{J}$
end procedure

In this formulation, a point can now contribute to multiple partitions, but an additional piece of information, p_i, needs to be recorded such that the observed point in a given partition contributes exactly $\sum_{\hat{J}} \gamma_{ij} = p_i$. In this way, we can use λ_p to control the amount of context sharing among children of different parents. Given that λ_p is sufficiently large, the algorithm will remain both highly efficient and parallelizable as only a small amount of "border points" will need to be counted multiple times in different partitions.

3.5. Parallel Construction

Additionally, we can further accelerate the calculation of expectations by parallelization on the GPU. Inspecting Equations 5-7 reveals that one only needs to keep track of J zeroth, first and second moments, weighted by their expec-

tations,

$$\{T_j^0, T_j^1, T_j^2\} \stackrel{\text{def}}{=} \{\sum_i \gamma_{ij}, \sum_i \gamma_{ij}\mathbf{z}_i, \sum_i \gamma_{ij}\mathbf{z}_i\mathbf{z}_i^T\} \quad (11)$$

These constitute sufficient statistics for the GMM. For the purposes of parallelization, the calculation of the above three quantities can be done in two steps: (1) Calculation of each γ_{ij} and (2) a weighted sum over all zeroth, first, and second moments. The former requires information about all J clusters but no information needs to be shared among points. The latter requires information about all N points but no information is needed from the J clusters once γ_{ij} are calculated. This allows for point-level parallelism in calculating γ_{ij} and an efficient point-level reduction sum when calculating the weighted moments.

4. Implementation Details

We first review the implementation of our hierarchical EM algorithm using hard partitions, and then in Sec. 4.1 discuss a generalization to soft partitioning.

Algorithm 2 shows the pseudocode for implementing the hard partitioned variant.

Algorithm 2 Hierarchical EM with Hard Partitions

procedure HIERARCHICAL_EM($\mathcal{Z}$, L, λ_s, λ_d)
  Init: parentIdx $\leftarrow \{-1\}_N$; $\mathbf{\Theta} \leftarrow \mathbf{\Theta}_{\text{init}}$
  for $l = 0 \dots L-1$ **do**
    while !Converged(λ_s) **do**
      $\{T^0, T^1, T^2, \text{currIdx}\} \leftarrow$ E_step($\mathcal{Z}$, Θ, parentIdx)
      $\mathbf{\Theta} \leftarrow$ M_step($T^0, T^1, T^2, l, \lambda_d$)
    end while
    parentIdx $\leftarrow$ currIdx
  end for
end procedure
procedure E_STEP($\mathcal{Z}$, Θ, parentIdx)
  for $i \in$ size($\mathcal{Z}$) **in parallel do**
    for $j \in$ Children(parentIdx[i]) **do**
      $\gamma_{ij} \propto \pi_j \mathcal{N}(\mathbf{z}_i|\mathbf{\Theta}_j)$
      $\{T_j^0, T_j^1, T_j^2\} \leftarrow$Accumulate($T_j^0, T_j^1, T_j^2, \gamma_{ij}, \mathbf{z}_i$)
    end for
    currIdx[i] $\leftarrow j$ s.t. $\max(\gamma_i) = \gamma_{ij}$
  end for
  return $\{T^0, T^1, T^2, \text{currIdx}\}$
end procedure
procedure M_STEP(T^0, T^1, T^2, l)
  for $j \in$ Level(l) **in parallel do**
    $\mathbf{\Theta}_j \leftarrow$ ML_Estimator(T_j^0, T_j^1, T_j^2)
    if !Supported(T_j^0, λ_d) **then** $\pi_j \leftarrow 0$
  end for
  return $\mathbf{\Theta}$
end procedure

HIERARCHICAL_EM: The algorithm takes as an input a point cloud $\mathcal{Z}$, the maximum number of levels of recursion, L, and two convergence parameters λ_s, λ_d. The first convergence parameter controls the stopping condition for a given set of EM steps, and the second convergence parameter controls the degree of geometric complexity of the final output by dropping clusters with insufficient support. To initialize Θ, we set our means to be the corners of the unit cube centered around zero. Note that during the execution of our algorithm we implicitly and recursively scale and offset the data to fit within the unit cube. The mixing weights are initially equal. Since these values are the same regardless of the recursion level for every new set of $\hat{J}$ mixtures, we only need to set these once at the very beginning of the algorithm. Likewise, we need to initialize an integer array *parentIdx* of size N to the value of -1, which will give us the correct child indices when $l = 0$ ($[0 \dots 7]$) to iterate over inside the first level's E step. After initialization is complete, we then iterate through L levels of the EM algorithm. After a given level has converged, we update our *parentIdx* array to point to the Maximum Likelihood estimates of subsurface expectation, recorded in *currIdx* during each iteration of the E step.

E_STEP: The E step calculates expectations over the child mixtures given the ML expectation of every point to the set of parent mixtures. The weighted moments $\{T^0, T^1, T^2\}$ (Eq. 11) can be calculated efficiently and in parallel using sum reductions or CUDA's *atomicAdd* functionality.

M_STEP: While the E step parallelizes over points, the M step parallelizes over subsurfaces (see Section 3.5). The *ML_Estimator* updates the model according to the standard MLE equations for GMM-based EM (cf. Eq. 5-7). Tikhonov regularization is done on the covariances to prevent numerical instability. Finally, clusters are dropped with insufficient support.

Note that if we implicitly encode the Gaussian Mixture tree in a large flat statically allocated array, the indexing functions *Children* and *Level* can be calculated in constant time: $\text{Children}(i) = [(i+1)\hat{J} \dots (i+2)\hat{J} - 1]$ and $\text{Level}(l) = [\frac{\hat{J}(\hat{J}^l - 1)}{\hat{J}-1} \dots \frac{\hat{J}(\hat{J}^{l+1}-1)}{\hat{J}-1} - 1]$.

Algorithm 3 E Step with Soft Partitions

procedure E_STEP($\{\mathcal{Z}, \mathcal{P}\}_K$, Θ)
  for $\mathbf{z}_i, p_{ik} \in \{\mathcal{Z}, \mathcal{P}\}_k, \forall k = 1 \dots K$ **in parallel do**
    for $j \in$ Children(k) **do**
      $\gamma_{ij} \propto \pi_j \mathcal{N}(\mathbf{z}_i|\mathbf{\Theta}_j)$
      $\{T_j^0, T_j^1, T_j^2\} \leftarrow$Accumulate($T_j^0, T_j^1, T_j^2, p_{ik}\gamma_{ij}, \mathbf{z}_i$)
    end for
  end for
  return $\{T^0, T^1, T^2\}$
end procedure

By looking at the construction of the algorithm and noting that $L = \log_{\hat{J}}(J)$ and $J \ll N$, we can see that the algorithm will run in $O(k \log_{\hat{J}}(J)(\hat{J}N))$, where k is the number of EM iterations until convergence. The normal "flat" EM algorithm would execute in $O(kNJ)$. Thus, we have produced an algorithm that speeds up model creation exponentially with respect to J, the total number of mixtures in the model. Furthermore, we have liberated J as a parameter that must be set *a priori*, instead letting the convergence

criterion λ_s and low support threshold λ_d determine when the point cloud has been sufficiently segmented.

4.1. Soft Partitioning

For hard partitions, updating the pointer *parentIdx* after EM convergence is all that is necessary for hierarchical construction since in the subsequent level we can then use the updated *parentIdx* in conjunction with the *Children* function as our index array into the tree.

To generalize the algorithm presented in Alg.2 to soft partitions, however, we need to record a few more pieces of data. Instead of a single *parentIdx* array, we need to record all expectations that fall above λ_p, as per the partitioning function outlined in Alg.1. Thus, we need to store both the index of partitioned points and their respective soft partitioning weights, $\frac{p_i \gamma_{ij}}{\xi}$. To do this, we modify line 8 of Alg.2 to instead call the *Partition* function from Alg. 1. The E step is then modified according to Alg.3. The only other change is that now inside *ML_Estimator* of the M Step: the new mix value must now be normalized using $\sum_i^N p_i$ and not the normal N (cf. Eq.5-7). With the modifications, a point in multiple subsurfaces will get distributed recursively throughout the hierarchy to all branches containing those subsurfaces in such a way that its expected contribution still sums to one among them. This important bookkeeping operation keeps consistency among the different paths down the tree.

5. PCD Processing with Generative Models

Our work in this paper focuses on the basic and fundamental problem of how one might efficiently construct, sample, and integrate over a generative model for PCD. We have so far explained how one might efficiently construct the model (Sec 3), but once the model is obtained, it is not trivial to see how one might sample and integrate over it, two operations that are fundamental for spatial processing applications. In this section, we describe two algorithms to efficiently perform sampling and integration, which we apply to point cloud reconstruction and occupancy grid generation, respectively. Our contribution is a novel importance sampling algorithm for GMMs that allows us to significantly reduce the number of samples required during integration by Monte Carlo sampling. Because these algorithms are not specific to hierarchical GMMs, we simplify the notation to the PDF as defined in Equation 1.

5.1. Point Cloud Reconstruction

To regenerate a set of N points we sample the distribution defined by the GMM as shown in algorithm 4. First we determine how many samples H_j to generate from each cluster j in $[1, J]$ according to the mixture weights π_j. Then we generate the H_j for each cluster according to the normal distribution defined by $\mathbf{\Theta}_j$. Details on this technique, known as *ancestral sampling*, can be found in Bishop *et al*. [2]. We present it here for completeness as this is an important operation on our model. Refer to Figure 1e for a graphical example of this result.

Algorithm 4 Point Cloud Reconstruction

1: **procedure** PCD_RECONSTRUCT(Θ, J, N)
2: calculate $\Pi_j = \sum_{i=1}^{j} \pi_i, \forall j \in J$
3: $S \leftarrow N$ random uniform samples in $[0, 1)$
4: $H \leftarrow$ histogram(S, Π) , Π provides bins extents
5: **for** $j = 1 \ldots J$ **do**
6: $P_j \leftarrow H_j$ points sampled from $\mathcal{N}(\mu_j|\Sigma_j)$
7: **end for**
8: **return** $P_j, \forall j \in J$
9: **end procedure**

5.2. Occupancy Grid Generation

To construct an occupancy grid we can stochastically sample points directly from the model to perform a Monte Carlo estimation of the probability that a given region of space is occupied. More formally, to build a discrete occupancy voxel grid we would like to spatially integrate our PDF over each voxel to obtain its probability estimate,

$$p(V_k|\mathbf{\Theta}) = \int_{V_k} \sum_{j=1}^{J+1} \pi_j p(v|\mathbf{\Theta}_j) dv, \tag{12}$$

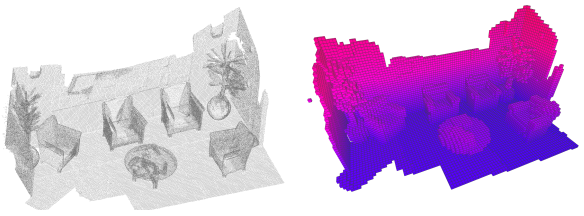
where V_k is a particular indexed cube or voxel.

Since there is no analytical solution to this integral, we resort to Monte Carlo sampling. However, uniform sampling the PDF over the space of the voxel grid will likely yield estimates with very high variance since the previously discussed sparsity will render many areas to essentially zero probability. Thus, we employ the use of importance sampling. To see how importance sampling is quite efficient in this context, we need to re-interpret the GMM as a weighted sum of zero-mean isotropic Gaussians that have been skewed and shifted through 3D space. To do this, we perform a Cholesky decomposition on the covariances, $\mathbf{\Sigma}_j = \mathbf{U}_j^T \mathbf{U}_j$. Then the multivariate normal equation is,

$$\mathcal{N}(x_i|\mathbf{\Theta}_j) = \xi_j^{-1} e^{-\frac{1}{2}(x_i - \boldsymbol{\mu}_j)^T \mathbf{U}_j^T \mathbf{U}_j (x_i - \boldsymbol{\mu}_j)} \tag{13}$$

$$= \xi_j^{-1} e^{-\frac{1}{2}\|\mathbf{A}_j x_i - \mathbf{b}_j\|^2}, \tag{14}$$

where $\mathbf{A}_j = \mathbf{U}_j$ and $\mathbf{b}_j = \mathbf{U_j}\boldsymbol{\mu}_j$, and ξ_j is a normalization factor. Thus, we can interpret each input x_i as undergoing an affine transformation before being evaluated through a zero-mean Gaussian function with identity covariance. To efficiently sample from the GMM therefore we first sample uniformly over $[0, 1]$ in 3 dimensions and then transform the values through the Φ^{-1} (probit) function. Our derived affine transformations of the samples $X \sim \mathcal{N}(0|I)$ for each of J subsurfaces place them in the GMM space. Furthermore, since the GMM is simply a linear combination of many Gaussians, once we have a collection of transformed samples, one only needs to keep track of what proportion



(a) Input PCD (1.6 *mil* pts) [21] (b) Occupancy grid map

Figure 3. **Occupancy estimates:** Left: Raw PCD. Right: an example high resolution occupancy map obtained by sampling a hierarchical GMM (max depth 5) produced from the points.

of these samples, per cluster, fall into a particular voxel and then multiply this ratio by the appropriate mixing parameter. Thus,

$$p(V_k|\mathbf{\Theta}) \approx \sum_{j=1}^{J+1} \frac{\pi_j}{N} \sum_{i=1}^{i=N} I_{V_k}(x_i) \tag{15}$$

where $x_i \sim \mathcal{N}(\boldsymbol{\mu}_j|\boldsymbol{\Sigma}_j)$ and I is an indicator function for whether x_i falls within the bounds of V_k.

Since the sampling function matches the underlying PDF (up to a multiplicative constant), these calculations yield unbiased estimates of the voxel probabilities and have low variance for even a fairly small number of samples. Furthermore, we precalculate all the samples X before reconstruction so that the entire process amounts to simply binning (voxelizing) the results of J different affine transformations over a relatively small static set of random or pseudorandom points. Since the model itself contains no voxels, as opposed to voxel-based or NDT methods, we are free to dynamically choose the extent and resolution of the occupancy grid at run-time, according to any number of application-specific constraints, arbitrarily defined axes, frustum culling, or locus of attention.

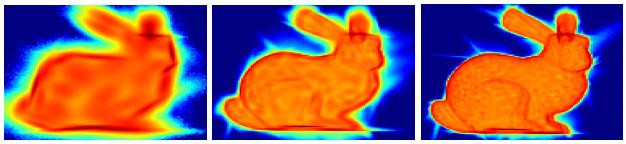
Figure 3 demonstrates this process for a large scene. Once we create a model from the points (in this case, a hierarchical GMM with a max depth of 5), we no longer need the raw PCD, and instead can at runtime produce a high quality occupancy map using the model only (Figure 3b).

6. Model Evaluation

We evaluate our model with respect to reconstruction fidelity and construction execution time. We used for testing the Stanford Bunny (∼36k points) and the Stanford 3D scene dataset, containing scenes with approximately ∼1-3 million points [20, 21]. On all these experiments, we statically set $\hat{J} = 8$ and our sparsity constraint, $\lambda_p = 0.1$.

6.1. Reconstruction Fidelity

The reconstruction fidelity provides a measure of how well our model can re-create the original point cloud. For this purpose, we use a PSNR (Peak Signal to Noise Ratio) metric derived from the Hausdorff distance as suggested



L2: 45.7 PSNR, 2.5kB 170x less data | **L3**: 48.7 PSNR, 20kB 21x less data | **L4**: 50.7 PSNR, 159kB 2.6x less data

Figure 4. **Levels of Fidelity:** The colors in each heatmap shows the accumulated PDF values projected onto the screen space. PSNR, model size, and *the reduced storage size* are shown for each level. We vary the level of detail (L2 to L4) to show the trade-off between storage size and fidelity. The original PCD is 421 kB.

by [3]. Specifically, given a reference a point cloud of size N, we stochastically generate an equivalent N amount of points from our model as described in section 5.1. Then, for every point in the original point cloud, we find the nearest neighbor in the reconstructed cloud. The logarithm of the inverse root mean squared error for all points relative to the bounding box size around the point cloud gives our PSNR metric. Note that the PSNR is on a logarithmic scale. Thus, linear increases in PSNR correspond to exponential reductions in average point-to-point recreation error. By computing the PSNR at different levels of details of our model we can provide an insight on the trade-off between model size and fidelity and fairly compare these trade-offs against other generative techniques.

Figure 4 shows a visual representation of the PDF, the PSNR and model size for three different levels of Bunny model. Though many different potential GMMs may be extracted from the hierarchy, we restrict our comparisons to GMMs consisting of leaf nodes are different max levels. For example, a "Level 3" GMM would the GMM extracted from the hierarchy by taking all the leaves of the GMM tree at a max depth of 3. As seen in Figure 4, the PDF from Level 2 provides a fairly good approximation while at the same time using ∼170 less storage than the original PCD. By level 4, the fidelity is such that points generated from the model are virtually indistinguishable from the original PCD. Because model construction using the hierarchical GMM framework can be viewed as the sparse embedding of 3D points into a higher dimensional 10D space, we can save on storage space with respect to the original size of the PCD, while still retaining a high level of geometric fidelity. We don't claim that we provide a state-of-the-art compression algorithm, but we note instead that our model allows for important savings in storage, in addition to its other properties.

We compare our model against the 3D-NDT representation [1] and its octree-based variant [14]. We chose the 3D-NDT as it is a widely used state-of-the-art generative model. To generate the trendlines for 3D-NDT, we calculate models at increasingly finer voxel resolution, and for the octree variant, we use increasingly smaller splitting thresholds. As a baseline we compute a subsampled version of the

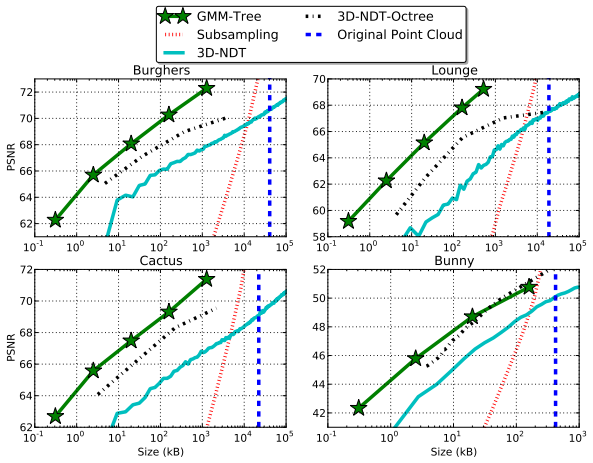


Figure 5. A comparison of data structure size vs fidelity over several standard point cloud datasets. The blue dashed line indicates the original point cloud size. Note the x-axis is on log scale. The star markers indicate different levels in the GMM hierarchy. At similar size models, the hierarchical GMM has much better PSNR (reconstruction performance) with respect to the original data when compared against the 3D-NDT, 3D-NDT-Octree, and a simple subsampling strategy for point cloud reduction.

original point cloud. Randomly subsampling of the PCD can be seen as a basic way to reduce the data size and is a common preprocessing technique for many 3D point algorithms. Fig. 5 shows the fidelity reconstruction vs storage size results. In every case but the bunny, we find that our model performs favorably to both NDT variants. This is because our optimization procedure yields a more compact representation without the need for voxelization. In the case of the bunny, however, the original point cloud is small enough that at higher model sizes we see diminishing returns over the octree-based NDT. In terms of statistical downsampling, both the hierarchical GMM and 3D-NDT are clearly much better choices to downsize the data while still retaining high geometric fidelity: for example, the level 5 GMM for Burghers achieves 72 PSNR with respect to the original point cloud, whereas a subsampled point cloud of equivalent size only yields 60 PSNR. Similarly, the smallest GMM more than doubles the PSNR over downsampling. To give another interpretation of this data: the level 3 GMM for Burghers contains as much reconstruction fidelity as if the point cloud were subsampled to about a fifth of its size, however, the level 3 GMM uses about 450x less memory.

6.2. Computational Speed and Scalability

We now take a look at the execution time of the hierarchical model construction. We implemented our algorithm in C++/CUDA and are able to run it on desktop and mobile platforms. Table 2 shows the construction times for each level in the hierarchy over different data sets using hard partitions. Note that, even on the mobile platform, the PCD

	Burghers (D) 307k pts	Bunny (D) 44k pts	Raw depth (M) 60k pts
L1	32.6ms (31Hz)	2.6ms (385Hz)	10.6ms (95Hz)
L2	57.8ms (17Hz)	4.3ms (233Hz)	15.3ms (65Hz)
L3	72.6ms (14Hz)	7.7ms (130Hz)	22.0ms (45Hz)
L4	92.2ms (11Hz)	11.7ms (85Hz)	31.5ms (32Hz)

Table 2. **Hierarchy construction time.** L1 to L4 refers to a level of the hierarchy (i.e., L3 denotes the the process including L1 to L3). D refers to a desktop computer (i5-3500/GTX660) used for the computation, and M denotes a Mobile device (NVIDIA Shield tablet). Raw depth refers to the point cloud directly captured from Softkinetic DS325 depth camera.

#J	Hierarchy	Flat	Speed-up
8	79.2 ms	61.6 ms	0.78×
64	145.6 ms	166.5 ms	1.14×
512	184.2 ms	1145.4 ms	6.22×
4096	213.8 ms	8750.8 ms	40.93×
32768	251.0 ms	71061.7 ms	283.11×

Table 3. **Construction speed-up relative to a flat GMM model.** The table compares the E Step execution time of the hierarchical GMM compared with a flat GMM having the same number of mixtures on the full Burghers model (∼4.5M pts). At higher detail levels, the proposed hierarchical GMM is significantly faster to build than the flat GMM.

from a depth camera can be processed at rates higher than 60FPS for a level 2 decomposition. In addition, we compare the construction time of our hierarchical model against the flat GMM model. We show in Table 3 that our hierarchical method becomes increasingly faster relative to a flat version with larger numbers of clusters, making our method more suitable in applications where high fidelity is desired yet construction times need to remain low.

7. Conclusion and Future Work

We introduced a hierarchical data structure based on Gaussian mixture models that comprises a compact and generative representation for 3D point cloud data. Model creation is accelerated by producing bounds on spatial interactions between the points and model. Thus, hierarchical model construction is orders of magnitude faster than the equivalent flat model, which follows as a result of replacing a large EM problem into multiple smaller ones. We demonstrated that the PDF can be used to effectively model the original data at different levels of detail, reconstruct point clouds of arbitrary sizes, and compute probabilistic occupancy estimates, fundamental components of many spatial perception and 3D modeling applications. Compared to discrete methods or hybrid approaches such as the NDT, our model yields higher fidelity results at smaller sizes, with modest construction time trade-offs even on mobile hardware. Future work will explore techniques for hierarchical shape-based object recognition, and locally rigid globally non-rigid surface registration with our generative models.

References

[1] P. Biber and W. Straßer. The normal distributions transform: A new approach to laser scan matching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2743–2748, 2003. 2, 3, 7

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. 3, 6

[3] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998. 7

[4] B. Eckart and A. Kelly. REM-Seg: A robust EM algorithm for parallel segmentation and registration of point clouds. In *IROS*, pages 4355–4362, 2013. 2

[5] B. Eckart, K. Kim, A. Troccoli, A. Kelly, and J. Kautz. Mlmd: Maximum likelihood mixture decoupling for fast and accurate point cloud registration. In *IEEE International Conference on 3D Vision*. IEEE, 2015. 2

[6] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989. 2

[7] J. Elseberg, D. Borrmann, and A. Nüchter. One billion points in the cloud–an octree for efficient processing of 3d laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing*, 76:76–88, 2013. 1

[8] G. D. Evangelidis, D. Kounades-Bastian, R. Horaud, and E. Z. Psarakis. A generative model for the joint registration of multiple point sets. In *Computer Vision–ECCV 2014*, pages 109–122. Springer, 2014. 2

[9] V. Garcia, F. Nielsen, and R. Nock. Levels of details for Gaussian Mixture Models. In *ACCV*, pages 514–525. Springer, 2010. 3

[10] J. Goldberger and S. T. Roweis. Hierarchical clustering of a mixture model. In *Advances in Neural Information Processing Systems*, pages 505–512, 2004. 2, 3

[11] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, pages 189–206, 2013. 2

[12] B. Jian and B. C. Vemuri. Robust point set registration using gaussian mixture models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(8):1633–1645, 2011. 2

[13] A. Kalaiah and A. Varshney. Statistical geometry representation for efficient transmission and rendering. *ACM Transactions on Graphics*, 24(2):348–373, 2005. 3

[14] M. Magnusson, A. Lilienthal, and T. Duckett. Scan registration for autonomous mining vehicles using 3d-ndt. *Journal of Field Robotics*, pages 803–827, 2007. 2, 7

[15] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation*, 2011 2011. 1

[16] J. Ryde and H. Hu. 3d mapping with multi-resolution occupied voxel lists. *Autonomous Robots*, 28(2):169–185, 2010. 1

[17] J. Ryde and H. Hu. 3d mapping with multi-resolution occupied voxel lists. *Auton. Robots*, 28(2):169–185, 2010. 2

[18] J. Saarinen, H. Andreasson, T. Stoyanov, J. Ala-Luhtala, and A. J. Lilienthal. Normal distributions transform occupancy maps: Application to large-scale online 3d mapping. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2233–2238. IEEE, 2013. 3

[19] S. Thrun and A. Bü. Integrating grid-based and topological maps for mobile robot navigation. In *AAAI'96*, pages 944–950, 1996. 2

[20] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 311–318. ACM, 1994. 7

[21] Q.-Y. Zhou and V. Koltun. Dense scene reconstruction with points of interest. *ACM Transactions on Graphics*, 32(4):112, 2013. 7