Point Cloud Streaming using Temporal Hierarchical GMMs

Roland Fischer
rfischer@cs.uni-bremen.de
University of Bremen
Bremen, Germany

Tobias Gels
Haya Almaree
Gabriel Zachmann
University of Bremen
Bremen, Germany

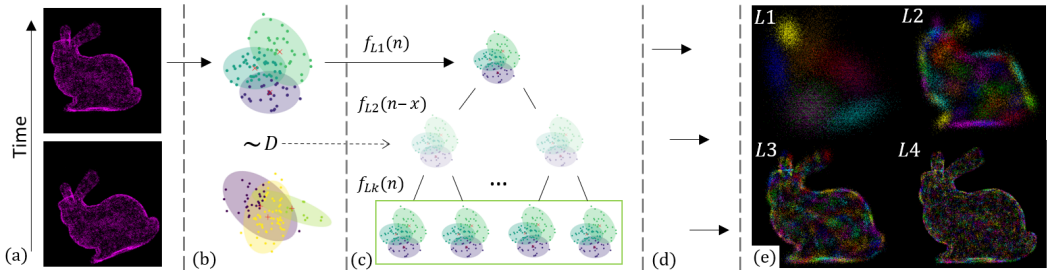

Figure 1: Our approach: Sequential point cloud input (a), computing GMM and divergence to previous frame (on level 1)(b), constructing GMM hierarchy re-using levels based on divergence (c), adaptive, progressive streaming (d), sampling LODs for reconstruction (e)(components depicted by colors). Each level significantly increases the fidelity and L4 is already quite accurate.

Abstract

Efficient processing and accurate representation of point clouds are crucial for many tasks, such as real-time 3D scene and avatar reconstruction. Especially for web/cloud-based streaming and telepresence, minimizing time, size, and bandwidth becomes paramount. We propose a novel approach for compact point cloud representation and efficient real-time streaming using a generative model consisting of a hierarchy of overlapping Gaussian Mixture Models (GMMs). Our level-wise construction scheme allows for dynamic construction and rendering of LODs, progressive transmission, and bandwidth- and computing power-adaptive transmission. Utilizing temporal coherence in sequential input, we reduce construction time significantly. Together with our highly optimized and parallelized CUDA implementation, we achieve real-time speeds with high-fidelity reconstructions. Moreover, we achieve significantly higher compression factors, up to 59 %, than previous work with only slightly lower accuracy.

CCS Concepts

• **Computing methodologies** → **Shape modeling**; **Massively parallel algorithms**.

Keywords

Point Cloud, Streaming, Gaussian Mixture Model, Generative Model

Web3D '25, Siena, Italy

ACM ISBN 979-8-4007-2038-3/2025/09
https://doi.org/10.1145/3746237.3746289

ACM Reference Format:
Roland Fischer, Tobias Gels, Haya Almaree, and Gabriel Zachmann. 2025. Point Cloud Streaming using Temporal Hierarchical GMMs. In *The 30th International Conference on 3D Web Technology (Web3D '25), September 9–10, 2025, Siena, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3746237.3746289

1 Introduction and Related Work

Point clouds play an important role in robotics [Kim et al. 2018], autonomous driving [Chen et al. 2021], and telepresence applications [Yu et al. 2021][Gamelin et al. 2021] with typical tasks such as SLAM and 3D scene/avatar reconstruction. However, noisy sensor data, huge data loads, and inhomogeneous densities make efficient processing and accurate representation challenging. This is especially true for real-time and streaming-based applications, which are heavily time- and size-constrained. With the growing popularity of (collaborative) web- and cloud-based streaming, this issue gets increasingly relevant.

Spatial data structures speed up the processing and reduce the size of 3D data, i.e., voxelization and occupancy-based methods effectively discretize space and are popular for point cloud representation. However, they suffer from artifacts and high memory consumption. The latter can be reduced using truncated signed distance functions [Oleynikova et al. 2017]. Octrees and kd-trees also effectively compress point clouds [Garcia et al. 2020][Mekuria et al. 2017]. However, additional overhead leads to higher construction times, and the requirement to initially specify the leaf size might impact the fidelity. The Normal Distributions Transform (NDT) [Saarinen et al. 2013] combines voxel grids with Gaussian distributions inside the voxels but still requires an initial voxelization.

Generative probabilistic models avoid these issues by providing a continuous parametric representation of the data. They can model complex distributions while efficiently handling uncertainty in the data. GMMs have been shown to allow for compact representations

as well as high reconstruction fidelity and have been used for tasks such as occupancy modelling [O'Meadhra et al. 2019], (point cloud) registration [Chen et al. 2023][Mei et al. 2023], segmentation [Garcia et al. 2010], collision avoidance [Dhawale et al. 2018], incremental mapping [Goel and Tabib 2023] and compression. For instance, [Song et al. 2021] and [Sun et al. 2023] proposed GMM-based point cloud compression methods, and [Navarrete et al. 2018] developed a 3D compression method based on point clustering and replacement that allows using the compressed data for registration. Clustering is done using the fast GMM variant by [Greggio et al. 2011] and the Expectation-Maximization (EM) algorithm.

Hierarchical forms of GMMs can reduce the computational cost while retaining accurate representations. For instance, [Eckart et al. 2016] proposed a hierarchy of GMMs and a parallel EM algorithm for point cloud representation and processing. They reported favourable performance compared to octrees and NDTs. However, their construction scheme doesn't allow progressive rendering and they don't consider sequential data or data transmission. Later, they adopted this method for fast point cloud registration [Eckart et al. 2018]. In contrast, [Srivastava and Michael 2018] proposed a bottom-up hierarchy of GMMs to generate a multi-fidelity environment representation, [Garcia et al. 2010] employed hierarchical GMMs for image segmentation, and [Goldberger and Roweis 2004] iteratively merges similar clusters of a big GMM into smaller ones using divergence metrics and an EM-like algorithm. GMMs have also been extensively used for SLAM, i.e., [Dong et al. 2022] proposed a GMM-based communication-efficient multi-robot mapping system with dynamic component counts, [Gao and Dong 2023] presented a hierarchical, GMM-based approach for accurate real-time compression of map information using a combination of K-means and the EM algorithm, and [Goel and Tabib 2023] employed self-organizing GMMs for incremental surface mapping and reconstruction.

Few works employ GMMs and consider sequential input but [Bouchachia and Vanaret 2011] presented an incremental GMM that is dynamically updated over time using growing and shrinking operations for online data classification. Similarly, [Dai and Zhao 2020] proposed using incremental Gaussian Mixture Models for time-varying process monitoring. Their approach allows recursively updating model parameters, adaptively adding new Gaussian components, and discarding irrelevant ones. In contrast, we propose a novel temporal hierarchical GMM-based approach that is specifically designed for real-time point cloud streaming.

2 Our Approach

We use a hierarchy of GMMs to represent the point cloud as a number of M overlapping probabilistic 3D mixtures $\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{3/2}}\frac{1}{|\Sigma|^{1/2}} e^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$ with the mean μ being a 3D vector and the covariance Σ being an 3×3 matrix. A GMM is formed by J components $\theta_j = \{\pi_j, \mu_j, \Sigma_j\}$: $p(x) = \sum_{j=1}^{J} \pi_j \mathcal{N}(x|\mu_j,\Sigma_j)$ with π_j being mixing coefficients that sum to 1. If the color should be encoded, too, the Gaussians' dimensionality increases (i.e., 4D for hue encoding). Each successive level subdivides the parent GMM into sets of smaller ones, increasing the overall fidelity. See Fig. 1 for an overview. We employ a tree structure with dynamic subdivision and pruning for size reduction. We solve the EM algorithm for each GMM in the hierarchy, resulting in an overall complexity of $O(N \log M)$ with N points and M mixtures. We ensure a valid global GMM by propagating the mixing weights down the hierarchy.

In contrast to [Eckart et al. 2016], we construct the hierarchy level-wise, which enables us to progressively stream and render the computed levels/LODs L while the next finer one is being computed, see Fig. 1 (d,e). The hierarchy also allows us to do bandwidth-adaptive transmission and visualization. A key feature is that we exploit temporal coherence in sequential point clouds, e.g., live-captured sensor data, (Fig. 1(a)) to maximize efficiency and performance. Specifically, we utilize the level-wise hierarchy construction and compare the GMM at level 1 with the corresponding one from the previous input. Depending on a divergence metric D, we skip the computation of a proportional number of levels, see Fig. 1 (b,c). Subsequently, only more detailed levels have to be computed. For this, we reuse the parent GMMs and corresponding partitionings of the previous input and recompute the Gaussians with the current coordinates. For unstructured point clouds, an alternative would be to transform the detail-level Gaussians based on the differences in the upper levels. To reconstruct the point cloud, we sample the encoded distribution using ancestral sampling. To achieve even better visualization, the Gaussian splatting rendering technique [Wu et al. 2024] can easily be applied, thanks to the GMM representation. To achieve real-time performance, we parallelized our method using the GPU and highly optimized our implementation. Our source code is available at: www.cgvr.cs.uni-bremen.de/research/pointclouds/.

2.1 Adapted EM Algorithm and GMM Hierarchy

We adapted the EM algorithm for our requirements (i.e., performance). The computation of one GMM (J=8 components) using the EM algorithm is shown in Fig. 2 (top). First, (without prior information) we normalize all points of the (root) GMM into a unit cube and initialize the components in its corners (π=$\frac{1}{8}$) in form of axis-aligned ellipsoids. We only sum over all components $\sum_{j=1}^{J} \pi_j \mathcal{N}(x|\mu_j,\Sigma_j)$ once in the E-step and reuse it for the maximum likelihood estimation. Hence, the M-step is executed after the log-likelihood evaluation. We also calculate the moments $\mathcal{M}_j^{\{0,1,2\}} \stackrel{\text{def}}{=} \left\{\sum_{i=1}^{N}\gamma_{i,j}, \sum_{i=1}^{N}\gamma_{i,j}z_i, \sum_{i=1}^{N}\gamma_{i,j}z_i z_i^T\right\}$ already in the E-step when looping over all points N. Thus, the M-step iterates over the components J with their moments $\mathcal{M}_j^{\{0,1,2\}}$ as input instead of the complete expectation matrix γ. As the covariance matrix Σ is symmetrical, we only have to represent 6 of the 9 values. To speed up the convergence, we employ Tikhonov regularization in the M-step, resulting in $\Sigma_j^{new} = \mathcal{M}_j^2/\mathcal{M}_j^0 - \mu_j^{new}\mu_j^{new^T} + \mathcal{R}$ ($\mathcal{R}$ being the regularization matrix). Lastly, the GMM gets denormalized.

Using this adapted EM algorithm, we then construct the GMM hierarchy by partitioning the points of a parent GMM (level L) into J partitions $\mathcal{P}_j = \{z_i | z_i \in \mathcal{Z} \wedge (\gamma_{i,j} > \lambda_c) \wedge (0 \le j < J) \wedge (0 \le i < N)\}$. Each one contains all points z_i for which the expectation of the j-th component lies above a threshold λ_c. Thus, we allow overlapping GMM components. This should only occur rarely but help with robustness against noise or outliers. Each partition $\mathcal{P}_j$, and the corresponding partition assignment $\mathcal{A}_j$, is used as an input for a new GMM on the next level L+1, see Fig. 2 (bottom). In contrast to the root GMM, the covariance matrices Σ get initialized with the corresponding ones from the parent GMM for faster convergence. The resulting weights π are multiplied with their associated ones from the parent GMM

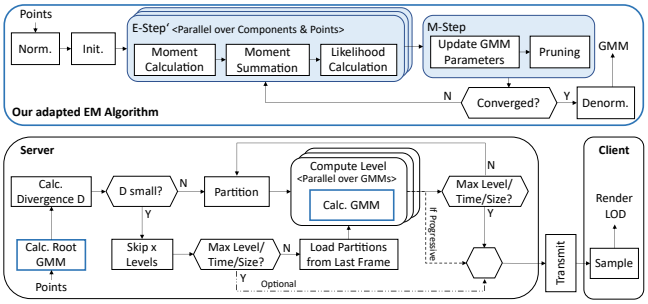

Figure 2: Top: Our adapted EM algorithm. Bottom: Our whole streaming approach (blue boxes represent adapted EM algo.).

to ensure they sum up to 1 on the level. Using this scheme, the GMM hierarchy gets further expanded in a breadth-first way until a desired maximal (detail) level L_{max} is reached or another stopping criterion is met. Theoretically, the combined component count J_L grows exponentially with the hierarchy depth: J_L=8^L. To minimize the eventual size, we prune the tree by discarding components that represent too few points and prevent further subdivision for these components. Specifically, a component is discarded when the sum of the expectations $\gamma_{i,j}$ (given by the zeroth moment M_{0j}) across all N points is below the product of the threshold parameter λ_d.

2.2 Utilizing Temporal Coherence

We first store all components J', the partitions $\mathcal{P}'$ (by reference to recognize coordinates differences), and the partition assignments $\mathcal{A}'$ of all levels of the previous frame. When computing the level 1 GMM, we initialize it by re-using the parameters from the last frame. Then, we measure the divergence/similarity D to the last frame. Monte Carlo sampling would be accurate but more efficient is to use the Kullback-Leibler divergence metric D_{KL} and compare the similarity of the two probability distributions. As there is no closed-form solution, we use the variational approximation by [Hershey and Olsen 2007] $D_{var}(J||K) = \sum_j \pi_j \log \frac{\sum_{j'} \pi_{j'} e^{-D_{KL}(\{\Sigma_j,\mu_j\}||\{\Sigma_{j'},\mu_{j'}\})}}{\sum_k \omega_k e^{-D_{KL}(\{\Sigma_j,\mu_j\}||\{\Sigma_k,\mu_k\})}}$ because it is the most accurate, while also being efficient when applied after level 1 (only 8 components). Other advantages are that it considers the divergence between all component pairs of both GMMs in both directions, and handles the weights appropriately. If the divergence is small, we skip x levels and only construct the more detailed ones, see Fig. 2 (bottom). We use the components J', partitioning $\mathcal{P}'$, and partition assignment $\mathcal{A}'$ from the previous frame to compute the Gaussians of the first of these levels with the current points. After that, the hierarchy is constructed as usual. The number of skipped levels x depends on the divergence thresholds ϵ_{KL} (vector of length L_{max}-1). If all levels can be skipped (i.e., practically equal frames), the construction stops and the most detailed level $J'_{L_{max}}$ from the last frame (or nothing) is transmitted.

2.3 Transmission and Sampling

After construction, we serialize the parameters of all components J_L for transmission to the client side, see Fig. 2 (bottom). Naturally, the data could be compressed using any (lossless) compression algorithm such as LZ4, ZStandard or the GPU-friendly, clipped Huffman encoding by [Goel et al. 2024] for further size reduction.

With our level-wise construction, we can do progressive and computing power- or bandwidth-adaptive transmission. When using the former, instead of only sending each frame's max level representation $J_{L_{max}}$, each hierarchy level J_L is transmitted and sampled on the client, replacing the previous level L-1, while the next, finer one is computed on the server. In our implementation the sampling is faster than the construction, thus, each received level can be reconstructed in time and correct order. For adaptive transmission, a max time budget and max size can be set and dynamically changed up to which the hierarchy is allowed to be built for each frame. If any of these limits are reached, the construction stops and the last fully constructed level is transmitted.

Once all components J_L of the desired level have been transferred and de-serialized, the point clouds can be reconstructed using only the component's parameters. We do this by ancestral sampling using Bernoulli distributions. The number of samples N_j for each component is given by the weight π_j and the total number of points. For evaluation purposes, we chose the same sample count as the original input. We implemented the sampling algorithm using CUDA and parallelized over the total number of samples N_s.

3 Implementation Details on Parallization

To maximize performance, we parallelized individual steps of the EM algorithm using CUDA and utilized the shared memory to a maximum extent. Most importantly, for the E-step, we parallelized over all input points and components. With 8 components per GMM, we can compute up to $\mathcal{T}$=128 points per component in parallel. As the computation of the moments requires the whole point set, we use another kernel that sums up all partial moments from all thread blocks $\mathcal{B}$. For fast summations, we use parallel prefix sums. The log-likelihood is computed efficiently using the atomicAdd() function. The maximum likelihood estimation and the M-step then greatly benefit from the described pre-computations in the E-step.

We compute all GMMs on the same level in parallel, too, by using dynamic parallelism (nested kernels). See supplementary material for pseudo code. Thus, the parallel computation of a single GMM can stay intact while another (outer) kernel handles the level-wise parallelization. This includes normalization of the partition's points and components initialization. The former requires finding the min and max points per dimension. We implemented this again using parallel prefix sums, the shared memory, and an extra summation kernel, similar to the moments' summation. To ensure that all threads from the inner E-step kernel have finished before the outer EM algorithm proceeds, we implemented a custom synchronization method using a global counter variable with an atomic state across all thread blocks. Eventually, the complexity of computing the hierarchy increases roughly linearly with the depth L.

4 Results

We measured the construction time, size required to represent the model, and the accuracy using the well-known point-to-point-based PSNR quality metric that is based on the Hausdorff distance [Cignoni et al. 1998][Tian et al. 2017]. We used an RTX4090 GPU and L_{max}=4, λ_c=0.2, λ_d=0.0001, ϵ_{KL}={0.01, 0.005, 0.0001} on 2

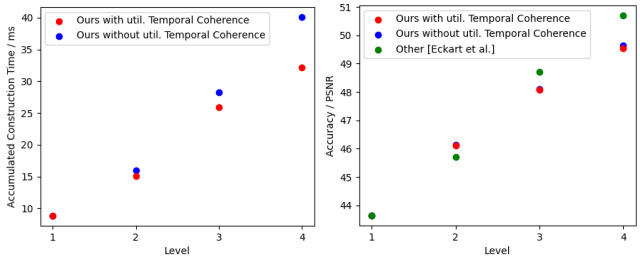

Figure 3: Our temporal approach greatly accelerates the construction (left) without significantly compromising accuracy (right). Eckart et al. has slightly higher PSNR values, though.

Table 1: Our algorithm with temporal coherence (TC) compared to no TC (Full), and Eckart et al. on PSNR, accum. construction times (AT) and compression (scene 1).

L	PSNR			AT (ms)		Compression Factor		
	Full	TC	Eckart	Full	TC	Full	TC	Eckart
1	43.641	43.641	NA	8.8	8.8	1348	1348	NA
2	**46.140**	46.114	45.7	16.0	**15.1**	183.4	**184.2**	170
3	48.115	48.072	**48.7**	28.2	**25.9**	26.27	**26.713**	21
4	49.647	49.542	**50.7**	40.1	**32.2**	3.833	**4.126**	2.6

test scenes: First, for comparability, the Stanford Bunny with ~36k points. We continuously transform it (30 frames) to get a dynamic scene with some geometric similarity between subsequent frames. The second one is a recording done with an Azure Kinect depth sensor (50 frames, up to 368,640 points/frame). It depicts a motionless room, see Fig. 5 (left), in which a person enters and waves his hand. To calculate the compression factor, we assume 4 Bytes per point coordinate, thus, 431,340 Bytes for scene 1 when transferred unprocessed/uncompressed.

Figure 1 (e) shows test scene 1 reconstructed from hierarchy level 1 to 4 (8 to 2297 components). As can be seen, the fidelity and visual quality increase significantly with each level, and with only 4 levels, the reconstructions are very accurate. Figure 3 depicts the accumulated construction time (left) and accuracy (right), both without and with utilizing temporal coherence (TC). The results show that our method is fast and accurate, and, with our temporal approach, we save a lot of time at higher levels, without a noticeable loss in accuracy. Compared to [Eckart et al. 2016], our method has a slightly lower PSNR on levels 3/4 and a slightly higher one at level 2. However, the balance between speed and accuracy can be tuned by adjusting the parameters (i.e., divergence thresholds). As they used a different GPU and no source code is available, we refrain from making a speed comparison. Figure. 4 (left) shows that our method achieves significantly higher compression than Eckart et al. (up to 58.7% higher on level 4) and our method with exploiting TC (42% frames) achieves the best compression factors: 4.1 for level 4 and 26.7 for level 3. The full data can be seen in Table 1.

Figure 5 shows test scene 2 and selected frames reconstructed with max level 4. The reconstructed scene looks quite convincing (black borders are sensor padding). Utilizing TC, the accumulated construction time decreases significantly again: 103.0 ms with and

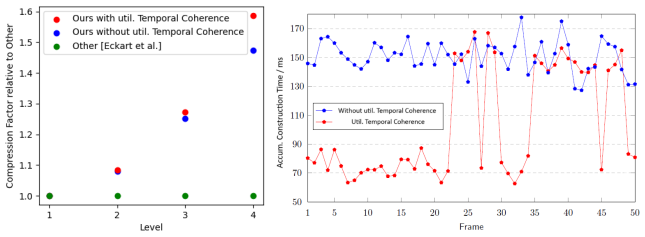

Figure 4: Left: Relative to Eckart et al., we achieve significantly higher compression factors. Right: The frame timings (scene 2) show that our temporal approach is very effective.

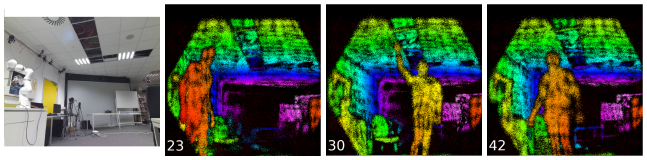

Figure 5: Frames reconstructed with max level 4 (scene 2; colors depict the components): quite good fidelity/clustering.

150.8 ms without TC. The individual frame timings, see Fig. 4 (right), show that the temporal approach works quite well. For instance, for all the first 20 frames (no movement in the scene), exploiting temporal coherency leads to roughly half the construction time. After that, roughly one-third of the frames were similar enough to temporally reuse information (62 % frames in total). Naturally, this depends heavily on the scene and the divergence thresholds. In this scene, however, the PSNR (42.794/47.548) and compression factors (6.419/6.996) were lower when utilizing TC, too. Compared to scene 1, the construction times are a lot higher, mainly due to the 10 times higher point count. Also, the achieved compression factors are a lot higher. However, interestingly, using TC performs significantly worse in this scene in terms of PSNR and achieved compression. We suspect artifacts (superfluous points seemingly in regions without valid data) to be the likely cause.

5 Conclusions and Future Work

We presented a novel approach for compact point cloud representation and real-time (web-based) streaming using a temporal hierarchical GMM-based generative model. Our level-based construction scheme successively partitions the input point cloud into a hierarchy of overlapping GMMs and allows to dynamically adjust the maximum LOD and progressively transmit and render more detailed levels. We minimize the construction cost by exploiting the temporal coherence between consecutive frames. Combined with our adapted EM algorithm and highly parallelized and optimized CUDA implementation, we achieve real-time speeds with high-fidelity reconstructions. Our results show that we achieve significantly higher compression factors, up to 59 %, than previous work with only slightly lower accuracy. Moreover, our temporal approach is highly effective and, on higher LODs, saves 20-32 % construction time in our test scenes. In the future, we plan on also encoding the color, integrating Gaussian splatting, and generalizing our approach to unstructured point clouds.

References

Abdelhamid Bouchachia and Charlie Vanaret. 2011. Incremental Learning Based on Growing Gaussian Mixture Models. In *2011 10th International Conference on Machine Learning and Applications and Workshops*, Vol. 2. 47–52.

Hong Chen, Baifan Chen, Zishuo Zhao, and Baojun Song. 2023. Point Cloud Registration Based on Learning Gaussian Mixture Models With Global-Weighted Local Representations. *IEEE Geoscience and Remote Sensing Letters* 20 (2023), 1–5.

Siheng Chen, Baoan Liu, Chen Feng, Carlos Vallespi-Gonzalez, and Carl Wellington. 2021. 3D Point Cloud Processing and Learning for Autonomous Driving: Impacting Map Creation, Localization, and Perception. *IEEE Signal Processing Magazine* 38, 1 (2021), 68–86.

Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. 1998. METRO: Measuring error on simplified surfaces. *Computer Graphics Forum* 17 (06 1998), 167 – 174.

Qingyang Dai and Chunhui Zhao. 2020. Incremental Gaussian Mixture Model for Time-varying Process Monitoring. In *2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)*. 1305–1311.

Aditya Dhawale, Xuning Yang, and Nathan Michael. 2018. Reactive Collision Avoidance Using Real-Time Local Gaussian Mixture Model Maps. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3545–3550.

Haolin Dong, Yu Jincheng, Yuanfan Xu, Zhilin Xu, Zhaoyang Shen, Jiahao Tang, Yuan Shen, and Yu Wang. 2022. MR-GMMapping: Communication Efficient Multi-Robot Mapping System via Gaussian Mixture Model. *IEEE Robotics and Automation Letters* 7 (04 2022), 1–1.

B. Eckart, K. Kim, and J. Kautz. 2018. HGMR: Hierarchical Gaussian Mixtures for Adaptive 3D Registration. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Ben Eckart, Kihwan Kim, Alejandro Troccoli, Alonzo Kelly, and Jan Kautz. 2016. Accelerated Generative Models for 3D Point Cloud Data. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5497–5505.

Guillaume Gamelin, Amine Chellali, Samia Cheikh, Aylen Ricca, Cedric Dumas, and Samir Otmane. 2021. Point-cloud avatars to improve spatial communication in immersive collaborative virtual environments. *Personal and Ubiquitous Computing* 25, 3 (2021), 467–484.

Yuan Gao and Wei Dong. 2023. An Integrated Hierarchical Approach for Real-Time Mapping With Gaussian Mixture Model. *IEEE Robotics and Automation Letters* 8, 11 (2023), 6891–6898.

Diogo C. Garcia, Tiago A. Fonseca, Renan U. Ferreira, and Ricardo L. de Queiroz. 2020. Geometry Coding for Dynamic Voxelized Point Clouds Using Octrees and Multiple Contexts. *IEEE Transactions on Image Processing* 29 (2020), 313–322.

Vincent Garcia, Frank Nielsen, and Richard Nock. 2010. Levels of Details for Gaussian Mixture Models. In *Computer Vision – ACCV 2009*, Hongbin Zha, Rin-ichiro Taniguchi, and Stephen Maybank (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 514–525.

Kshitij Goel and Wennie Tabib. 2023. Incremental Multimodal Surface Mapping via Self-Organizing Gaussian Mixture Models. *IEEE Robotics and Automation Letters* PP (12 2023), 1–8.

Rahul Goel, Markus Schütz, P. J. Narayanan, and Bernhard Kerbl. 2024. Real-Time Decompression and Rasterization of Massive Point Clouds. *Proc. ACM Comput. Graph. Interact. Tech.* 7, 3, Article 48 (Aug. 2024), 15 pages.

Jacob Goldberger and Sam Roweis. 2004. Hierarchical clustering of a mixture model. *Advances in neural information processing systems* 17 (2004).

Nicola Greggio, Alexandre Bernardino, Cecilia Laschi, Paolo Dario, and José Santos-Victor. 2011. Fast estimation of Gaussian mixture models for image segmentation. *Machine Vision and Applications - MVA* 23 (07 2011), 1–17.

John R. Hershey and Peder A. Olsen. 2007. Approximating the Kullback Leibler Divergence Between Gaussian Mixture Models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, Vol. 4. IV–317–IV–320.

Pileun Kim, Jingdao Chen, and Yong K. Cho. 2018. SLAM-driven robotic mapping and registration of 3D point clouds. *Automation in Construction* 89 (2018), 38–48.

Guofeng Mei, Fabio Poiesi, Cristiano Saltori, Jian Zhang, Elisa Ricci, and Nicu Sebe. 2023. Overlap-Guided Gaussian Mixture Models for Point Cloud Registration. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 4511–4520.

Rufael Mekuria, Kees Blom, and Pablo Cesar. 2017. Design, Implementation, and Evaluation of a Point Cloud Codec for Tele-Immersive Video. *IEEE Transactions on Circuits and Systems for Video Technology* 27, 4 (2017), 828–842.

Javier Navarrete, Diego Viejo, and Miguel Cazorla. 2018. Compression and registration of 3D point clouds using GMMs. *Pattern Recognition Letters* 110 (2018), 8–15.

Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. 2017. Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning. 1366–1373.

Cormac O'Meadhra, Wennie Tabib, and Nathan Michael. 2019. Variable Resolution Occupancy Mapping Using Gaussian Mixture Models. *IEEE Robotics and Automation Letters* 4, 2 (2019), 2015–2022.

Jari Saarinen, Henrik Andreasson, Todor Stoyanov, Juha Ala-Luhtala, and Achim J. Lilienthal. 2013. Normal Distributions Transform Occupancy Maps: Application to large-scale online 3D mapping. In *2013 IEEE International Conference on Robotics and Automation*. 2233–2238.

Fei Song, Yiting Shao, Wei Gao, Haiqiang Wang, and Thomas Li. 2021. Layer-Wise Geometry Aggregation Framework for Lossless LiDAR Point Cloud Compression. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 12 (2021), 4603–4616.

Shobhit Srivastava and Nathan Michael. 2018. Efficient, Multifidelity Perceptual Representations Via Hierarchical Gaussian Mixture Models. *IEEE Transactions on Robotics* PP (11 2018), 1–13.

Jianjun Sun, Yan Zhao, Shigang Wang, and Jian Wei. 2023. 3D Holoscopic Image Compression Based on Gaussian Mixture Model. *IEEE Transactions on Multimedia* 25 (2023), 1374–1389.

Dong Tian, Hideaki Ochimizu, Chen Feng, Robert Cohen, and Anthony Vetro. 2017. Geometric distortion metrics for point cloud compression. In *2017 IEEE International Conference on Image Processing (ICIP)*. 3460–3464.

Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 2024. 4D Gaussian Splatting for Real-Time Dynamic Scene Rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 20310–20320.

Kevin Yu, Gleb Gorbachev, Ulrich Eck, Frieder Pankratz, Nassir Navab, and Daniel Roth. 2021. Avatars for Teleconsultation: Effects of Avatar Embodiment Techniques on User Perception in 3D Asymmetric Telepresence. *IEEE Transactions on Visualization and Computer Graphics* PP (08 2021), 1–1.

Further Details and Evaluations

Further details of our temporal construction approach and how we skip levels can be found in Algorithm 1. Similarly, Algorithm 2 provides additional information about our hierarchical EM algorithm. Here, EML denotes the outer kernel that handles the parallel computation of all GMMs at one level.

Algorithm 1 Using Temporal Coherence by Level Skipping

procedure FRAME
  $L = 1$
  $J_1 = \text{EMROOT}(\mathcal{Z})$
  $L = 2$
  $\mathcal{A}_2, \mathcal{P}_2 = \text{PARTITION}(\mathcal{Z}_{J_1})$
  $D_{var} = \text{DIVERGENCE}(J_1 || J_1')$
  for $i = L$ to L_{max} **do**
    if $(D_{var} < \epsilon_{KL_i})$ **then**
      $L = i + 1$
      $J_i = J_i'$
      $\mathcal{A}_i, \mathcal{P}_i = \mathcal{A}_i', \mathcal{P}_i'$
      break
  while $(L \leq L_{max})$ **do**
    $J_L = \text{EML}(J_{L-1}, \mathcal{A}_L, \mathcal{P}_L)$
    $L = L + 1$
    $\mathcal{A}_L, \mathcal{P}_L = \text{PARTITION}(\mathcal{Z}_{J_{L-1}})$

Algorithm 2 Hierarchical EM Algorithm

procedure MAIN($\mathcal{Z}_{J_{L-1}}, J_{L-1}$)
  $\mathcal{P}_{J_L}, \mathcal{A}_{J_L} = \text{PARTITION}(\mathcal{Z}_{J_{L-1}})$
  $J_L = \text{EML}\ll J_{L-1}, 64 \gg (J_{L-1}, \mathcal{P}_{J_L}, \mathcal{A}_{J_L})$
procedure EML($\theta_{L-1}, \mathcal{P}_j, \mathcal{A}_j$) **in parallel**
  $\mathcal{B} = \mathcal{P}_j / \mathcal{T}$
  $\mathcal{U} = \mathcal{B} / \mathcal{T}$
  $\max_{\mathcal{B}}, \min_{\mathcal{B}} = \text{BOUNDS}\ll \mathcal{B}, \mathcal{T} \gg (\mathcal{P}_j)$
  $\max, \min = \text{SUMS}\ll \mathcal{U}, \mathcal{T} \gg (\max_{\mathcal{B}}, \min_{\mathcal{B}})$
  $\mathcal{P}_{j;\text{norm}} = \text{NORM}\ll \mathcal{B}, \mathcal{T} \gg (\max, \min, \mathcal{P}_j)$
  $\theta_L = \text{Init}(\theta_{L-1})$
  while do
    $\mathcal{M}_J^{\{0,1,2\}} = \text{ESTEP'}(\mathcal{A}_j, \mathcal{P}_{j;\text{norm}}, \theta_L)$
    if (HasConverged()) **then**
      $\theta_L = \text{DENORM}\ll \mathcal{B}, \mathcal{T} \gg (\max, \min, \theta_L)$
      break
    $\theta_L = \text{MSTEP}(\mathcal{M}_J^{\{0,1,2\}})$

We also evaluated the effectiveness of the dynamic pruning (default configuration) compared to building the complete hierarchy with all components. As can be seen in Fig. 6 (left) and Tab. 2, the compression factor (CF) is significantly worse without dynamic pruning while the PSNR increases only slightly. The pruning performs especially well on higher levels (i.e., 1.78 times higher compression vs 0.5 % PSNR loss on level 4).

Lastly, we investigated the impact of different parallelization configurations on the performance. Specifically, we examined which ratio of thread blocks $\mathcal{T}$ (outer kernel) and threads per thread block performs the best (on scene 1, no TC). Looking at Fig. 6 (right) and Tab. 3, we can see that the performance, overall, increases when the number of threads per thread block is reduced from 64 to 32 and the number of thread blocks is increased, respectively. While the difference is small on the lower levels, the 32 threads configuration is significantly faster on the higher hierarchy levels.

Note: This paper is an extended version of a previous SIGGRAPH poster.

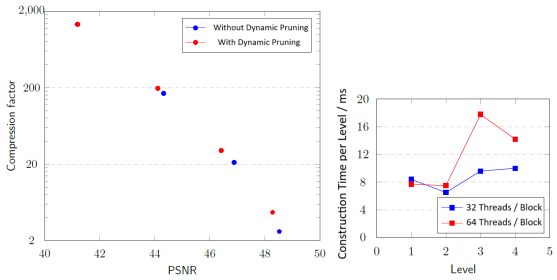

Figure 6: Left: Dynamic pruning vs. full hierarchy regarding accuracy and compression factor. Data pairs (left to right) depict levels 1 to 4. Right: Construction times per level with 32 and 64 threads/thread block (outer kernel). The 32 threads configuration is generally faster.

Table 2: Dynamic pruning vs. full GMM hierarchy regarding component count (J_L) per level, accuracy, and compression factor (CF).

L	Dyn. Pruning			Full		
	J_L	PSNR	CF	J_L	PSNR	CF
1	8	41.20	1348	8	41.20	1348
2	55	44.11	**196**	64	**44.32**	169
3	357	46.42	**30.2**	512	**46.88**	21.1
4	2299	48.28	**4.69**	4096	**48.53**	2.63

Table 3: Construction times (ms)(AT: accumulated) using 32 and 64 threads/thread block.

L	Time		AT	
	32t	64t	32t	64t
1	8.4	**7.7**	8.4	**7.7**
2	**6.5**	7.5	**14.9**	15.2
3	**9.6**	17.8	**24.5**	33.0
4	**10.0**	14.2	**34.5**	47.2