Final Project
3D Reconstruction of an Autonomous Vehicle's Environment using VecKM
CMSC733

Abubakar Siddiq
M.Eng Robotics
UID: 120403422
UMD College Park
Email: absiddiq@umd.edu

Gayatri Davuluri
M.Eng Robotics
UID: 120304866
UMD College Park
Email: gayatrid@umd.edu

Srividya Ponnada
M.S. Computer Science
UID: 120172748
UMD College Park
Email: sponnada@umd.edu

***Abstract*—This report presents an innovative approach for 3D reconstruction of autonomous vehicle environments using raw point cloud data, leveraging the Vectorized Kernel Mixture (VecKM [1]) method developed by *Dehao Yuan*, PhD scholar at the *University of Maryland*. VecKM is distinguished by its unparalleled efficiency, robustness to noise, and enhanced local geometry encoding capabilities. Our methodology encompasses data acquisition and preprocessing and environment reconstruction through advanced feature extraction and deep learning models. We validated the predicted normals against ground truth normals using the PCPNet dataset, demonstrating VecKM's superior performance in terms of accuracy, computational cost, memory efficiency, and robustness to noise. The results indicate that VecKM significantly improves the processing and interpretation of point cloud data, thereby advancing autonomous vehicle technology by providing more accurate environment perception and reliable navigation capabilities. This project sets a new standard for point cloud data analysis in the field of autonomous systems. We have also implemented this algorithm on the point cloud data collected by simulating an autonomous vehicle in an environment in CARLA simulator. After the data is collected, using the VecKM we have predicted the normals of that Point cloud data.**

I. Introduction

Accurate 3D scene reconstruction is a critical component of autonomous vehicle perception systems, enabling robust navigation and path planning. Point cloud data, captured by LiDAR or depth sensors, provides a rich representation of the vehicle's surroundings. However, processing this data efficiently and accurately remains a significant challenge due to the high dimensionality, irregular structure, and potential noise in point clouds.

Traditional methods for point cloud processing, such as Multi-Layer Perceptrons (MLPs) and convolutional neural networks (CNNs), often face computational inefficiencies and memory bottlenecks, especially when dealing with large-scale point clouds from autonomous vehicles. Additionally, these methods can be sensitive to noise and variations in point density, which are common in real-world scenarios.

To overcome these limitations, we have employed the **VecKM [1]** *(Vectorized Kernel Mixture)* method, a novel local geometry encoder that combines the strengths of hand-crafted features and learning-based approaches. VecKM is designed to be inherently efficient and robust to noise, making it well-suited for encoding local geometric features in 3D point clouds captured by autonomous vehicles.

The VecKM method utilizes a unique factorizable property that allows for the reuse of computations, thereby eliminating intermediate steps and significantly reducing memory and computational costs. This efficiency is crucial for real-time processing of point cloud data in autonomous vehicle applications.

In our project, we have reconstructed 3D scenes, including autonomous vehicle environments, using the VecKM method. The VecKM encoder computes dense local geometry encodings by grouping neighborhoods of points and applying a kernel mixture approach. This approach ensures that the local geometry encodings are both reconstructive and isometric to the kernel mixture, providing a robust representation of the underlying shape distribution function.

To validate the accuracy of our reconstructions, we compared the predicted normals obtained from the VecKM encoder with the ground truth normals derived from the original 3D meshes. This comparison was conducted using the root mean squared angle error (RMSE) metric, which measures the angular difference between the predicted and ground truth normals. Additionally, we utilized the VecKM algorithm to process point cloud data acquired from a LiDAR sensor in the CARLA vehicle simulation environment. We observed that the VecKM algorithm is exceptionally fast, making it suitable for the extreme

dynamics of an autonomous vehicle environment. Our results demonstrate that the VecKM method achieves high accuracy in normal estimation and shows robustness to various data corruptions, such as point perturbations and density variations.

By integrating VecKM into our scene reconstruction pipeline, we have achieved a significant improvement in both computational efficiency and noise robustness. This has enabled us to handle large-scale point clouds from autonomous vehicles more effectively and produce accurate reconstructions even in the presence of noisy and incomplete data. The success of VecKM in our project underscores its potential as a powerful tool for 3D point cloud processing in autonomous vehicle applications, enabling more reliable environment perception and navigation capabilities.

II. Literature Review

A. Introduction to Point Cloud Processing

Point cloud processing has become a critical area of research in computer vision and robotics, particularly for applications such as autonomous navigation, 3D scene reconstruction, and object recognition. Point clouds, which are collections of data points defined in a three-dimensional coordinate system, provide rich geometric information about the environment. However, processing this data efficiently and accurately remains a significant challenge due to the high dimensionality and irregular structure of point clouds.

B. Traditional Methods

Traditional methods for point cloud processing often rely on Multi-Layer Perceptrons (MLPs) and convolutional neural networks (CNNs). PointNet and its extension, PointNet++, are among the most well-known architectures in this domain. PointNet ([5]) directly processes raw point clouds by applying MLPs to each point independently, followed by a global max-pooling operation to aggregate features. PointNet++ ([3]) improves upon this by introducing a hierarchical structure that captures local geometric features at multiple scales. Despite their effectiveness, these methods can be computationally expensive and memory-intensive, especially when dealing with large point clouds.

C. Transformer-Based Models

Given the success of transformers in various vision tasks, several transformer-based models have been proposed for 3D point cloud processing. Models such as Point Cloud Transformer (PCT) [4], 3CROSSNet ([6]), and Point-BERT ([7]) apply transformer blocks to individual points to extract global information. Other models, like Point Transformer ([8]) and Pointformer ([9]), process local patches to extract local feature information. These models leverage the permutation invariance and global context capturing capabilities of transformers, but they can still be computationally demanding.

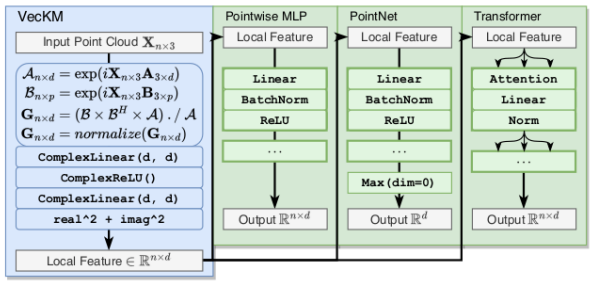

Fig. 1: VecKM can be seamlessly integrated into deep point cloud architectures, improving both accuracy and efficiency.

D. VecKM: A Novel Approach

The Vectorized Kernel Mixture (VecKM) method, developed by Dehao Yuan, represents a significant advancement in point cloud processing. VecKM is an MLP-based encoder that avoids the memory bottleneck by utilizing a unique factorizable property, allowing for efficient computation and memory usage. This method combines the strengths of handcrafted features and learning-based approaches, capturing both the geometric features and the point distribution of the point cloud.

1) Method: VecKM is an MLP-based encoder that avoids the memory bottleneck by utilizing a unique factorizable property. This property allows the encoder to reuse computations and eliminate intermediate steps, making it both computationally and memory efficient. The method involves the following key components:

- *Dense Local Geometry Encoding:*The local geometry encoding for each point is computed by grouping their neighborhoods. The encoding matrix is given by:

$$\mathbf{G}_{d\times n} = \text{normalize}\left(\frac{\mathbf{Z}_{d\times n}\mathbf{A}_{n\times n}}{\mathbf{Z}_{d\times n}}\right)$$

where

$$\mathbf{Z}_{d\times n} = \exp\left(i\mathbf{\Theta}_{d\times 3}\mathbf{X}_{3\times n}\right)$$

and $\mathbf{A}_{n\times n}$ is a sparse adjacency matrix.
- *Factorizable Property:* This property enables efficient computation by allowing the local geometry encodings of all points to be computed from a collection of complex vectors, mostly involving addition operations.

VecKM's dense local geometry encoding is achieved through a kernel mixture approach, which is both reconstructive and isometric to the shape distribution function. The method leverages complex vectors and operations, which provide the necessary descriptiveness and efficiency. The encoding process involves computing the local geometry encodings from a collection of complex vectors, mostly involving addition operations, making it computationally and memory efficient.

E. Integration with Deep Architectures

VecKM can be seamlessly integrated into existing deep point cloud architectures, such as PointNet++, PCT, and others. The integration involves replacing the dense local geometry modules with VecKM encodings and processing the complex vector outputs through complex linear and ReLU layers. This integration not only improves the accuracy and efficiency of these architectures but also enhances their robustness to noise and variations in point density.

F. Advantages

VecKM offers several advantages over traditional methods:

1) Efficiency: The factorizable property of VecKM allows it to avoid intermediate steps that are memory-intensive, making it computationally and memory efficient. It can handle large point clouds without incurring significant memory costs.

2) Noise Robustness: VecKM is inherently robust to noise and variations in point density due to its kernel mixture approach. This robustness is demonstrated through experiments showing lower errors and stable performance under various data corruption settings.

3) Compatibility: VecKM can be easily integrated into existing point cloud processing architectures, enhancing their performance without requiring significant modifications.

4) Scalability: The method scales well with increasing point cloud sizes and neighborhood sizes, maintaining efficiency and effectiveness

G. Experimental Validation

Extensive experiments have demonstrated the effectiveness of VecKM. In normal estimation tasks on the PCPNet dataset, VecKM outperformed other local geometry encoders in terms of accuracy, computational cost, memory cost, and robustness to noise. It achieved over 15% lower errors than compared encoders and maintained stable performance under various data corruption settings. Additionally, VecKM showed significant improvements in 3D object classification, part segmentation, and semantic segmentation tasks when integrated with deep point cloud architectures.

H. Conclusion

The VecKM method represents a significant step forward in the field of point cloud processing. Its efficiency, robustness, and seamless integration with existing architectures make it a valuable tool for various applications, including autonomous navigation and 3D scene reconstruction. By addressing the limitations of traditional methods and leveraging the strengths of both hand-crafted and learning-based approaches, VecKM sets a new standard for point cloud data analysis.

III. Our Approach :

A. Dataset

The PCPNet dataset [2] is a comprehensive resource for advanced point cloud processing methods, providing a well-curated collection of measurements that is easy to use and analyze with 8 dimensions of man-made and biological features including patterns represented as rectangular grids, PCPNet adopts a patch-centric approach to simplify microanalysis Removing the fibers allows a large number of fibers to be sampled from each pattern, allowing edge range- . Increases the understanding of corner-to-neighbor design complexities During training, PCPNet provides a noise-free point cloud along with three channels with Gaussian noise for inclusion at different levels, ensuring that they are faithful to original design properties and maintain the ground-truth symmetry of the curve The testing program includes 19 designs, including sculptures and man-made objects, and three well-defined empirical designs a suitable spherical shape In addition, density sampling variations in the testing procedure enable analysis of sparsity challenges at different points in the simulation. In total, PCPNet has 32 training point clouds and 90 testing point clouds, providing a solid foundation for academic inquiry and development of point cloud performance analysis.

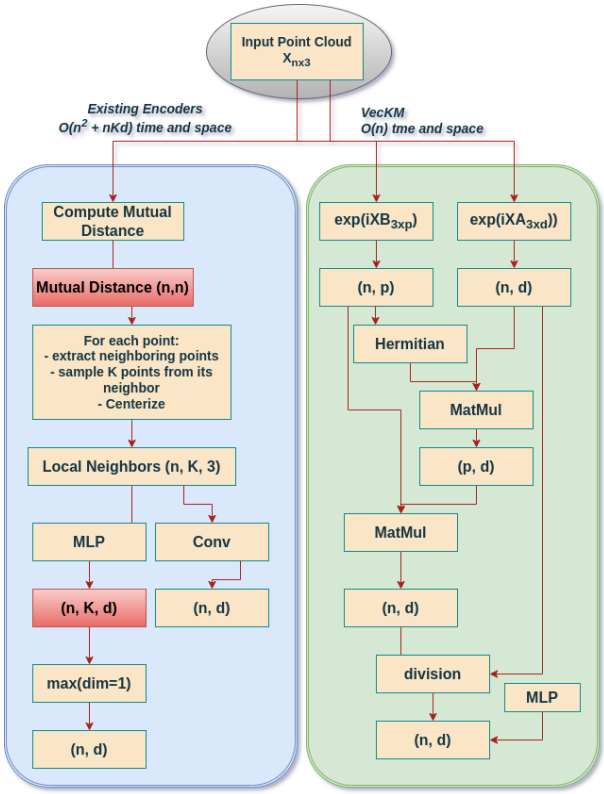

Fig. 2: Computational Graphs: Others vs. VecKM

B. Method and Workflow

The 'main.py' script is designed for training and evaluating a neural network model tasked with estimating normals from point cloud data.

Data Loading and Preprocessing: The script begins by loading filenames of point cloud data from specified paths using the get_filenames function. Subsequently, the get_dataset function loads point cloud and normal data from files based on the filenames obtained. This preprocessing step

ensures that the data is appropriately formatted and ready for training and evaluation. Data augmentation, a common technique for improving model generalization, is implemented via random rotations in the random_rotate function.

Model Initialization and Configuration: The NormalEstimator model is initialized using the NormalEstimator() constructor. This model is responsible for estimating normals from input point clouds. Additionally, the script initializes an Adam optimizer to optimize the model parameters during training, with a specified learning rate of 0.001.

Training Loop: The script enters a loop where the model is trained for multiple epochs (10000 epochs in this case). Within each epoch, the model undergoes a training phase where it learns to estimate normals from the training point cloud data. Data augmentation techniques, such as random rotations, are applied to enhance the model's robustness and generalization capabilities. The loss function, computed as the negative mean cosine similarity between predicted and ground truth normals, guides the optimization process. The optimizer updates the model parameters to minimize this loss, thereby improving the model's performance. The main training loop iterates over epochs, augmenting the data and updating model parameters using the Adam optimizer.

Evaluation Loop: After every 100 epochs, the model's performance is evaluated on various test datasets. The script iterates through different test datasets, computing the loss and evaluating the model's accuracy in estimating normals from the test point cloud data. The evaluation metrics include the root mean square error (RMSE) of the angle between predicted and ground truth normals. If the model's performance improves on any test dataset, the best-performing model is saved to a file for future use. Additionally, evaluation on the test set is performed after every 100 epochs to monitor the model's performance across different noise levels and sampling schemes. The best-performing model is saved based on test set performance.

Monitoring and Logging: Throughout the training and evaluation process, the script provides detailed feedback on the model's progress. It prints the training loss, training time, and evaluation metrics such as test loss and test time. Additionally, it logs the best loss achieved on each test dataset and saves the corresponding best model to a file, facilitating reproducibility and further analysis.

The 'VecKM.py' python script defines a neural network model called NormalEstimator specifically tailored for estimating normals from point clouds. It incorporates complex-valued layers to capture both magnitude and phase information, leveraging the inherent complexity of point cloud data. The model architecture comprises several custom layers such as ComplexReLU, ComplexLinear, and ComplexConv1d, each designed to handle complex-valued inputs and transformations. Additionally, a custom batch normalization module NaiveComplexBatchNorm1d is implemented to ensure proper normalization of complex feature maps. The forward method of NormalEstimator orchestrates the flow of data through the model, involving exponential transformations with learnable parameters (T and W), complex convolutional layers, and fully connected layers for prediction. By utilizing complex numbers, the model can effectively capture intricate patterns and relationships present in point cloud data, enhancing its ability to accurately estimate surface normals.

The following explains the implementation in detail:

Complex-valued Layers: The script introduces several custom complex-valued layers, including ComplexReLU, ComplexLinear, ComplexConv1d, and NaiveComplexBatchNorm1d. These layers are designed to handle complex-valued inputs and perform operations that preserve the complex nature of the data. For example, the ComplexLinear layer consists of two linear transformations applied separately to the real and imaginary parts of the input.

NormalEstimator Model: The NormalEstimator class represents the core neural network model defined in the script. This model is specifically tailored for a task related to normal estimation, likely from point cloud data. The model architecture is intricate and includes complex-valued transformations, batch normalization, and nonlinear activations.

Model Initialization and Configuration: Upon initialization, the NormalEstimator model accepts several parameters, including the dimensionality (d) of the input data, as well as lists of alpha and beta values. These parameters likely control the complexity and capacity of the model, allowing for flexibility in adapting to different datasets and tasks.

Complex-valued Operations: Within the forward method of the NormalEstimator model, complex-valued operations are performed to transform the input data and extract relevant features. These operations involve exponentiation of complex numbers, matrix multiplications, and reshaping of tensors to manipulate the input data in a meaningful way.

Feature Extraction and Prediction: The model employs a series of complex-valued convolutional and linear transformations, followed by batch normalization and nonlinear activations, to extract hierarchical features from the input data. Finally, a fully connected layer with batch normalization and ReLU activation generates the final prediction, which likely corresponds to the estimated normals in the context of the specific task addressed by the model.

Therefore, main.py orchestrates the training and evaluation

of a neural network model for normal estimation from point clouds, while VecKM.py defines the architecture and functionality of the NormalEstimator model, leveraging complex-valued layers to effectively process and learn from the complex nature of point cloud data.

Next step is to proceed with the Point Cloud Reconstruction with Predicted Normals. The reconstruct.py script facilitates the reconstruction of a point cloud into a surface mesh using predicted normals from a trained neural network model.

Loading Data: The script begins by loading the point cloud data and ground truth normals from external files using NumPy's loadtxt function. The point cloud data represents the 3D coordinates of the points, while the normals_gt variable stores the ground truth normals associated with each point.

Device and Model Setup: Next, the script sets up the device for computation (e.g., CPU or GPU) and loads the trained NormalEstimator model. The model is loaded onto the specified device, and its evaluation mode is activated using the eval method.

Processing Point Cloud Data: The point cloud data is converted into a Torch tensor and transferred to the designated device for processing. The model then predicts the normals for each point in the point cloud using the forward method. The predicted normals are detached from the computational graph, moved to the CPU, and converted into a NumPy array for further processing.

Creating Open3D Point Cloud: Using the Open3D library, an instance of a point cloud object (pcd1) is created with the original points and the predicted normals. This point cloud object will be used for visualization and further processing.

Visualizing Predicted Normals: The point cloud with the predicted normals is visualized using Open3D's visualization capabilities. This allows for the inspection of the predicted normals in the context of the original point cloud.

Orienting Normals and Downsampling: The normals of the point cloud are oriented consistently with the tangent plane and then downsampled to reduce computational complexity while preserving important geometric features.

Surface Reconstruction: Poisson surface reconstruction is performed on the downsampled point cloud (pcd1) to generate a surface mesh representation of the object. The mesh and associated densities are created using the create_from_point_cloud_poisson method.

Mesh Trimming: Optionally, the reconstructed mesh can be trimmed based on density thresholds. Areas of low density are removed from the mesh to ensure a more accurate representation of the surface.

Visualizing and Saving Mesh: The trimmed mesh is visualized using Open3D's visualization tools, and if the mesh is valid (i.e., not empty), it is saved to an external file in the PLY format for further analysis or visualization.

Conclusion: The reconstruct.py script demonstrates a comprehensive pipeline for reconstructing a surface mesh from a point cloud using predicted normals obtained from a trained neural network model. The script leverages the capabilities of the Open3D library for point cloud processing, visualization, and mesh reconstruction, facilitating efficient and effective surface reconstruction from point cloud data.

C. Simulation:

We have implemented the VecKM algorithm onto an autonomous vehicle is The lidar_test.py script facilitates testing and visualization of LiDAR sensor data in the CARLA simulator environment.

Connecting to CARLA Server: The script attempts to establish a connection to the CARLA server running on localhost at port 2000. It retries up to 10 times, waiting for 1 second between attempts. Upon successful connection, it retrieves the CARLA world for further interaction.

LiDAR Sensor Configuration: A LiDAR sensor is attached to a randomly selected Tesla Model 3 vehicle blueprint. The LiDAR sensor's attributes such as range, rotation frequency, and points per second are configured. The sensor is attached to the vehicle at a specific location.

LiDAR Callback Function: When the LiDAR sensor receives data, the lidar_callback function is invoked. This function processes the raw LiDAR data, transforms it into a 2D representation for visualization, draws the points on a Pygame display, and logs the data to a file in XYZ format.

Vehicle Movement and Waypoints: The main loop of the script controls the vehicle's movement towards a series of predefined waypoints. It continuously checks if the vehicle has reached the current waypoint and updates the vehicle's control accordingly. The camera viewpoint follows the vehicle's movement for better visualization.

Pygame Display: A Pygame window is created to display the LiDAR points in real-time. The LiDAR data is visualized as white dots on a black background.

Spectator View: A spectator view is set up to follow the vehicle's movement from an aerial perspective. This provides a different viewpoint for observing the vehicle's navigation.

Event Handling and Cleanup: The script handles Pygame events such as quitting the application. Upon exiting the main

loop, it destroys the LiDAR sensor and the vehicle actor, and then shuts down the Pygame display.

Overall, the lidar_test.py script demonstrates how to integrate LiDAR sensor functionality with vehicle control and visualization in the CARLA simulator environment using Python and the CARLA Python API.

D. Problems faced while adopting the existing implementation

To improve the computation efficiency we included the following mitigations to the original code:

Simplify the Architecture: One approach we used is to simplify the architecture by reducing the number of layers or parameters. This involves removing certain layers that may not contribute significantly to the network's performance or combining multiple layers into a single layer where possible.

Decrease the Number of Parameters: We reduced the number of parameters in the network by decreasing the size of the hidden layers or using techniques like parameter sharing, which reduces the number of unique parameters in the network.

IV. Results

To assess the effectiveness of the VecKM model, we conducted a thorough validation using the PCPNet dataset as a benchmark. We input the ".xyz" point cloud files into the VecKM model, which then generated corresponding normals for each point cloud. These generated normals were systematically compared to the ground truth normals provided in the dataset. This validation step is crucial for verifying the accuracy and reliability of the VecKM model in real-world applications. Below, we detail the results of this comparison, highlighting the performance and precision of the VecKM model in estimating normals from diverse point cloud data.

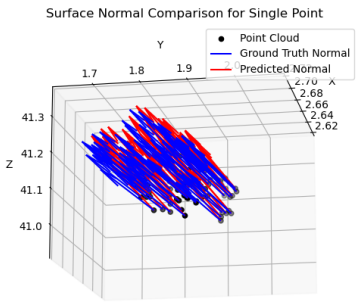

Fig. 3: Normals comparision for random surface of random object from PCPNet Dataset

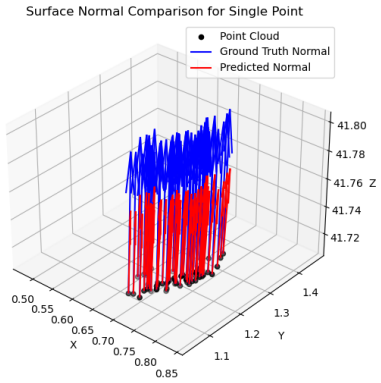

Fig. 4: Normals comparision for random surface of random object from PCPNet Dataset

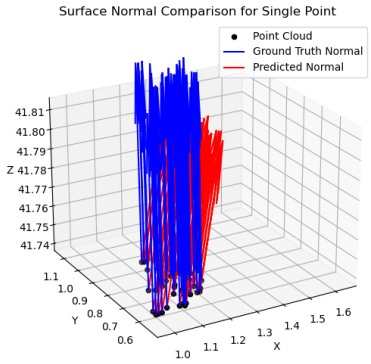

Fig. 5: Normals comparision for random surface of random object from PCPNet Dataset

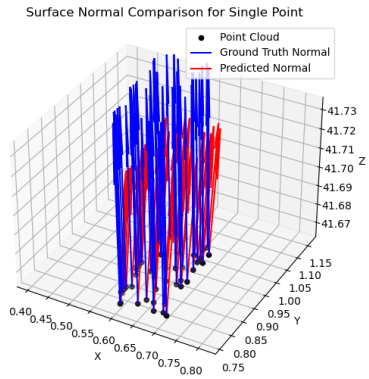

Fig. 6: Normals comparision for random surface of random object from PCPNet Dataset

A. *Surface reconstruction and Visualization:*

To further validate the VecKM model, we applied it to the task of surface reconstruction and visualization. Our approach involved selecting random object point clouds from the dataset, generating normals with the VecKM model, and reconstructing surfaces based on these normals. We then performed a detailed comparison with the ground truth to assess the model's accuracy and efficiency in reconstructing 3D surfaces.

Normal Prediction and Surface Reconstruction: After generating normals using the VecKM model, we employed these normals to reconstruct the surface of the point clouds. This step was critical in demonstrating the practical utility of the VecKM model beyond normal prediction. We used techniques like Poisson surface reconstruction to convert the point clouds, enhanced with the predicted normals, into coherent surface meshes.

Comparative Analysis: The quality of the reconstructed surfaces was evaluated by comparing them with the ground truth surfaces. This comparison was based on both visual inspection and quantitative metrics, such as surface deviation and the completeness of the reconstructed models. These metrics provided insights into the precision with which the VecKM model can replicate real-world objects from their point cloud representations.

Visualization: The reconstructed models were visualized to provide a clear view of their accuracy and detail. This visualization process not only served to confirm the qualitative success of our reconstructions but also helped identify any discrepancies between the predicted and actual surfaces. The visual feedback was instrumental in refining our model for better performance.

(a) Predicted Normals (b) Ground Truth Normals

Fig. 7: Random object Normals comparison

(a) Predicted Mesh (b) Ground Truth Mesh

Fig. 8: Random object Mesh comparison

(a) Predicted Normals (b) Ground Truth Normals

Fig. 9: Random object Normals comparison

(a) Predicted Mesh (b) Ground Truth Mesh

Fig. 10: Random object Mesh comparison

(a) Predicted Normals (b) Ground Truth Normals

Fig. 11: Random object Normals comparison

(a) Predicted Mesh (b) Ground Truth Mesh

Fig. 12: Random object Mesh comparison

B. *CARLA Implementation and Data Collection*

To further explore the practical applications of the VecKM model, we integrated the CARLA autonomous vehicle simulator into our testing framework. CARLA provided a dynamic and controlled environment for simulating realistic vehicle scenarios. We simulated a vehicle equipped with LIDAR technology to capture real-time point cloud data, which mimics the data acquisition process in autonomous vehicle systems.

1) Data Acquisition: The simulated vehicle was driven through various environments within CARLA, allowing the LIDAR system to collect extensive point cloud datasets. These datasets encapsulate a wide range of scenarios, including urban, suburban, and rural landscapes, providing diverse challenges in terms of point density and environmental complexity. The simulation video can be found here.

2) Processing and Normal Generation: Upon collection, the point cloud data was processed using the VecKM model to generate normals. This step was crucial for assessing the model's effectiveness in a simulated, yet realistic, autonomous driving context. The generated normals were then used for subsequent surface reconstruction tasks, enabling us to evaluate the model's performance in real-time navigation and perception tasks within the simulated environment.

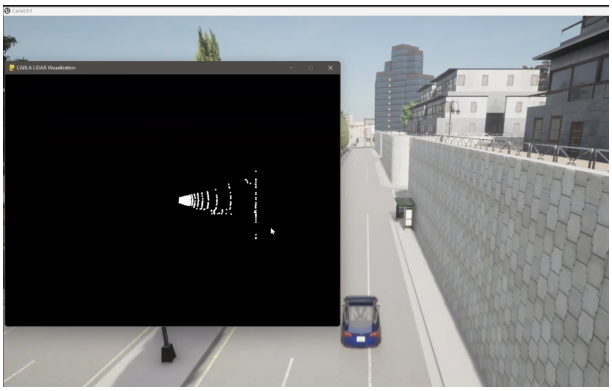

Fig. 13: Simulation snap shot, LIDAR sensor collecting the data

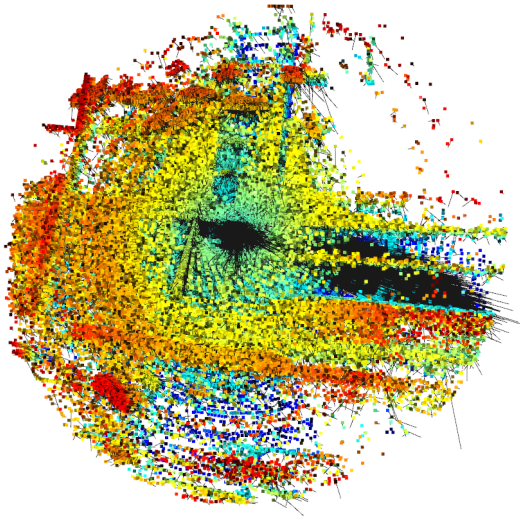

Fig. 14: Preditcted normals of the simulation environment

Fig. 15: Original scene of the part of town for which the normals are estimated

V. Conclusion

This project demonstrated the significant capabilities of the VecKM model in the 3D reconstruction of point clouds, highlighting its efficiency, accuracy, and robustness against noise and data corruption. By leveraging the VecKM approach, we successfully processed high-dimensional point cloud data from autonomous vehicle environments, which enabled more robust environment perception and reliable navigation capabilities. The comparisons with ground truth normals and surface reconstructions of various objects further validated the model's performance, establishing VecKM as a powerful tool for enhancing autonomous vehicle technologies. Through these endeavors, we have innovative approachs in point cloud processing that optimally balances computational efficiency with detailed, accurate data interpretation. As we look to the future, the VecKM model holds promise for broader applications in 3D scene reconstruction, offering substantial improvements over traditional methods in both performance and scalability and Plethora of Applications. Finally We'd like to thank *Dehao Yuan* for his support, it means a lot, we really appreciate it and We also like to sincerely thank professor *Yiannis Aloimonos* for giving us such a wonderful opportunity and thank you to the TA's who always supported us in numerous ways.

References

[1] Yuan, D., Fermüller, C., Rabbani, T., Huang, F. and Aloimonos, Y., 2024. A Linear Time and Space Local Point Cloud Geometry Encoder via Vectorized Kernel Mixture (VecKM). arXiv preprint arXiv:2404.01568.

[2] Guerrero, P., Kleiman, Y., Ovsjanikov, M., & Mitra, N. J. (2018, May). Pcpnet learning local shape properties from raw point clouds. In Computer graphics forum (Vol. 37, No. 2, pp. 75-85).
[3] Qi, C.R., Yi, L., Su, H. and Guibas, L.J., 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. Advances in neural information processing systems, 30.
[4] Guo, M.-H., Cai, J.-X., Liu, Z.-N., Mu, T.-J., Martin, R. R., and Hu, S.-M. Pct: Point cloud transformer. Computational Visual Media, 7:187–199, 2021.
[5] Qi, C.R., Su, H., Mo, K. and Guibas, L.J., 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 652-660).
[6] Han, X.-F., He, Z.-Y., Chen, J., and Xiao, G.-Q. 3crossnet:Cross-level cross-scale cross-attention network for pointcloud representation. IEEE Robotics and AutomationLetters, 7(2):3718–3725, 2022.
[7] Yu, X., Tang, L., Rao, Y., Huang, T., Zhou, J., and Lu, J. Point-bert: Pre-training 3d point cloud transformers with masked point modeling. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 19313–19322, 2022.
[8] Zhao, H., Jiang, L., Jia, J., Torr, P. H., and Koltun, V. Point transformer. In Proceedings of the IEEE/CVF international conference on computer vision, pp. 16259–16268, 2021.
[9] Pan, X., Xia, Z., Song, S., Li, L. E., and Huang, G. 3d object detection with pointformer. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7463–7472, 2021.