Efficient Mesh Simplification Using Variational Shape Approximation via Quadric Error Metrics in Open3d for Embedded Systems

James Earl Cubillas
Department of Computer Science, College of Computing and Information Sciences, Caraga State University

Mike Andre Puspus
Department of Computer Science, College of Computing and Information Sciences, Caraga State University

Mike Leoford Goyo
Department of Computer Science, College of Computing and Information Sciences, Caraga State University

https://hdl.handle.net/2324/7395564

出版情報：Proceedings of International Exchange and Innovation Conference on Engineering & Sciences (IEICES). 11, pp.518-525, 2025-10-30. International Exchange and Innovation Conference on Engineering & Sciences
バージョン：

Efficient Mesh Simplification Using Variational Shape Approximation via Quadric Error Metrics in Open3d for Embedded Systems

James Earl Cubillas[1], Mike Andre Puspus[2], Mike Leoford Goyo[3]
[123] Department of Computer Science, College of Computing and Information Sciences, Caraga State University
jdcubillas@carsu.edu.ph, mikeandre.puspus@carsu.edu.ph, mikeleoford.goyo@carsu.edu.ph,

Abstract: *Inspired by the advantages of the quadric error metrics initially shaped for mesh decimation, this study proposes an efficient mesh reconstruction method for 3D point clouds. This technique proceeds through clustering points enhanced with quadric error metrics, where each cluster gets a generator, an ideal 3D point for minimizing the total quadric error of the cluster. The approach places generators on prominent features and distributes errors evenly across clusters. It reconstructs the mesh using cluster adjacency and a constrained binary solver, while adaptively refining based on error. This approach investigates the effectiveness of Variational Shape Approximation via Quadric Error Metrics in handling point clouds and preserving the original shape of 3D models effectively. The proposed VSA-QEM integration aims to provide a more accurate and efficient simplification process, which is particularly beneficial for deployment on resource-constrained embedded systems.*

Keywords: mesh simplification, 3D point cloud, variational shape approximation, quadric error metrics, open3d

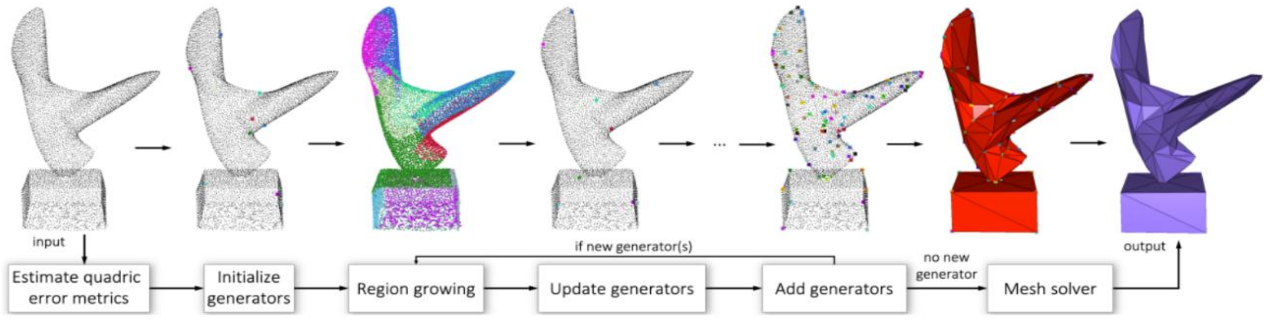

Fig. 1. Variational approximation via quadric error metrics (Zhao et al., 2023).

1. INTRODUCTION

Mesh simplification is a method in 3D graphics that lowers the number of polygons in models. Variational Shape Approximation (VSA) is an advanced technique for mesh simplification that optimizes shape approximation by partitioning the input data into clusters, each approximated by a simple geometric primitive, such as a plane [1]. Open3D is a library that facilitates rapid software development with 3D data. It is open-source and offers tools and algorithms for 3D data processing. Yet, the current version of Open3D's existing implementations of Quadric Error Metrics (QEM) in Open3D often produce simplified meshes that fail to accurately preserve the original shape of the models. Although the individual techniques, VSA and QEM, have been well-researched in mesh simplification, there is limited to no existing literature demonstrating the integration of VSA into QEM within the Open3D framework for embedded systems.

Additionally, optimization is necessary to maintain the efficiency and scalability of mesh simplification operations, especially for large and complex meshes. Low-powered, small devices require lightweight and optimized libraries for 3D data processing. Enhancing Open3D's QEM with VSA capabilities would significantly improve the user experience by offering a comprehensive and reliable solution for mesh simplification within the framework, eliminating the need for external tools or custom implementations. This study thus introduces a novel methodology that combines these approaches to enhance real-time 3D processing performance in hardware-constrained environments.

2. RELATED WORK

We first reviewed the quadric error metrics. We then focus on variational approaches and 3D libraries for implementing this study on embedded systems.

2.1. Quadric Error Metrics

Quadric Error Metrics (QEM) is a mathematical framework that simplifies 3D meshes, reducing the number of vertices in a mesh while maintaining the overall shape and appearance by iteratively collapsing edges and minimizing the associated error. Using a quadratic form to express this error metric, the error caused by collapsing edges may be efficiently calculated. According to Garland and Heckbert [2], the technique iteratively collapses the edge with the smallest error. This influential concept has emerged as one of the most widely adopted methods for mesh decimation. Pellizzoni & Savio [3] extended this approach with curvature information for improved simplification. Potamias et al. [4], Bergman et al. [5], and Barda et al. [6] utilize quadric error metrics for shape evolution and editing. Legrand et al. [7] introduced a mesh smoothing and clustering method through filtering quadrics.

While QEMs have been widely used for mesh simplification due to their computational efficiency, they come with several challenges. The initial study reveals challenges in effectively handling textured surfaces. It is further emphasized in subsequent studies, where the complexity of maintaining texture fidelity during the simplification process remains a significant hurdle [8]. Yadav et al. [9] discussed the challenge of balancing noise removal with feature preservation in point set denoising, where QEM may not maintain important geometric features effectively. This challenge is also addressed in the work of Lee et al. [10], who propose alternative methods for feature-aware simplification that attempt to retain critical geometric details.

2.2. Variational Approach

Introduced by Cohen-Steiner et al. [11], VSA is a technique for simplifying complex shapes into simpler, more manageable forms, maintaining essential geometric properties. VSA optimizes clusters of connected mesh triangles by minimizing a one-sided error metric formulated between triangles and planar proxies. The VSA process involves initializing random proxies, iteratively partitioning the mesh, and fitting proxies to minimize geometric distortion. This iterative cycle continues until convergence, after which the final mesh is generated and refined through constrained Delaunay triangulation. The result is a simplified yet geometrically accurate representation of the original mesh. One of the recent studies by Yu & Lafarge [12] developed a flexible method to segment a 3D point cloud into planar sections. Starting with an initial division, they optimized a multi-objective function using five operations (insert, exclude, transfer, merge, split) to balance accuracy, simplicity, and coverage. This optimized partition was used in a piecewise-planar reconstruction approach, Bauchet & Lafarge [13]. This method allows for a refined and efficient reconstruction of 3D models by effectively managing the trade-offs between different reconstruction goals. Similarly, Skrodzki et al. [14] adapted VSA to 3D point sets and added a switch operator to guarantee convergence. While partitioning helps organize the data, connecting optimized partitions of polygon elements is still challenging.

Zhao et al. [1] recent study proposes quadric error metrics (QEM) to cluster points and generate an optimized mesh directly, avoiding dense reconstruction followed by decimation. This is a foundational technique for the clustering process in reconstructing 3D surfaces from point clouds.

2.3. Open3D and Embedded Systems

Open3D is an open-source library that supports rapid software development for 3D data. Open3D has evolved from an Eigen-based API to a tensor-based framework, improving GPU acceleration and 3D data processing flexibility, with enhanced visualization tools and CUDA-accelerated reconstruction pipelines [15]. However, despite these improvements, meshes simplified using native QEM often fail to effectively retain essential geometric details and visual accuracy, particularly with high-density and textured surfaces [16, 17, 18]. In 3D data processing, embedded systems offer several advantages [19]. They can be integrated into devices that require real-time processing of 3D data [20]. Implementing mesh simplification techniques in embedded systems poses unique challenges and opportunities [21]. Integrating advanced mesh simplification techniques like VSA and QEM into embedded systems can greatly enhance performance.

3. APPROACH

The approach takes a 3D point cloud from a closed surface and produces a surface triangle mesh. It begins by estimating each point's Quadric Error Metric (QEM). Initial cluster generators are randomly selected, and clustering is performed through region growing and updating generators until each cluster meets an error tolerance. Areas with high error are refined by adding more generators. Finally, the mesh is extracted by solving a constrained binary problem. This process is shown in Figure 1.

3.1. Quadric Estimation

A detailed quadric estimation method adapts the quadric error metric (QEM) approach for mesh reconstruction from 3D point clouds. Originally, QEM was designed for mesh decimation by initializing a quadric for each vertex based on the mesh triangles. The method involves the following steps:

For each input point p_i in the 3D point cloud PA planar quadric is estimated using the local normal. n_i n_i And the point coordinates. This process involves computing the plane quadric. Q_{pi} As follows:

$$Q_{pi} = (n_{xi}, n_{yi}, n_{zi}, -n_i \ * \ p_i^T)^T \\ * \ (n_{xi}, n_{yi}, n_{zi}, -n_i \ * \ p_i^T) \qquad (1)$$

Here, n_{xi}, n_{yi}, n_{zi} are the components of the average vector n_i and p_i^T represents the coordinates of the point p_i.

To capture the local geometry around each point, we define a neighborhood and support area using the k-Nearest Neighbor (KNN) graph. The support area a_{pi} for point p_i is estimated as follows:

$$a_{pi} = \frac{1}{2k^2}\Big(\sum_{(p_j,p_i)\in KNN(P)} ||p_i - p_j||\Big)2 \qquad (2)$$

The diffused quadric for a point p_i is then computed by summing the quadrics of its neighboring points weighted by their respective support areas:

$$Q_{vi} = \sum_{(p_j,p_i)\in KNN(P)} a_{pj} \ * \ Q_{pj} \qquad (3)$$

This method ensures that the diffused quadric reflects an approximation of the local geometry and point density. Each quadric captures the essence of the local surface curvature and distribution, crucial for accurate mesh reconstruction from point clouds.

3.2. Initialization

The initial clustering of input points starts by randomly choosing a few as the initial cluster centers, denoted by $\{ci \mid 1 \leq i \leq m\}$. These initial centers are actual input points at the beginning. Still, as the process progresses, they will be updated to become optimal QEM 3D points, meaning they will no longer necessarily coincide with the original input points.

3.3. Clustering

Partitioning employs a region growing on the K-Nearest Neighbor (KNN) graph, using a priority queue to minimize the cost E by combining the Quadric Error Metric (QEM) and Euclidean distance. The cost of adding a point p_i to cluster l_j is given by:

$$E(p_i, l_j) = [c_j, 1]^T * Q_{v_i} * [c_j, 1] + \lambda * ||p_i - c_j||^2 \quad (4)$$

Generator Updating involves recomputing the optimal generator for each cluster by solving a linear system to minimize the sum of squared distances from the cluster points to the generator point. The linear system is:

$$Ac^* = (Q^{11}\ Q^{12}\ Q^{13}\ Q^{14}\ Q^{12}\ Q^{22}\ Q^{23}\ Q^{24}\ Q^{13}\ Q^{23}\ Q^{33}\ Q^{34}\ 0\ 0\ 0\ 0\ )c^* = (0\ 0\ 0\ 1\ ) \quad (5)$$

When A is not invertible, a Singular Value Decomposition (SVD) solver is used.

The process terminates when no generator changes or a maximum number of iterations is reached. Post-clustering, areas with maximum QEM errors exceeding a set tolerance are split, and new generators are added to improve accuracy. Independent sets of clusters are split to avoid over-refinement. Finally, a graph of edge candidates is derived from the cluster adjacency, facet candidates are computed, and a constrained binary program is solved to extract the surface triangle mesh.

3.4. Batch Splitting

The batch-splitting process begins by monitoring the maximum Quadric Error Metric (QEM) error for all points within a cluster once clustering terminates. A splitting operator inserts a new generator in the cluster where the maximum error occurs. For each cluster l_j, the point labeled j that maximizes Q_{c_j} is found using the following equation:

$$p_{max}(l_j) = arg\ [p_i, 1]^T * Q_{c_j} * [p_i, 1] \quad (6)$$

If the corresponding QEM exceeds a user-defined maximum tolerance, $p_{max}(l_j)$ is added as a new candidate for cluster splitting. An independent set of clusters to be split is computed greedily, ensuring that two adjacent clusters are not split in the same batch to avoid over-refinement. Instead of directly defining a meaningful QEM tolerance, a Euclidean distance tolerance parameter is used and converted to a QEM scale according to the relationship between these metrics.

The alternation between partitioning and batch splitting continues until all clusters satisfy the user-defined tolerance error or a maximum number of iterations is reached.

3.5. Meshing

The meshing process connects the vertices resulting from the current partition of clusters, which are optimal in the Quadric Error Metric (QEM) sense, by identifying and forming triangle facets. After partitioning, a graph of edge candidates is derived from the adjacency between clusters in the input points' K-Nearest Neighbor (KNN) graph. From this edge graph, 3-cycles are searched by selecting all triplets of vertices mutually connected by an edge, yielding a set of triangle facet candidates.

From the set of facet candidates, the goal is to retain those that fit well, cover the input 3D point cloud adequately, and favor a 2-manifold mesh. Building on the PolyFit approach, an objective function is minimized, rewarding a data fitting term F_f and a data coverage term F_c, under a 2-manifold constraint. Each facet candidate f_i and each edge e_i are assigned binary labels b_{f_i} and b_{e_i}, respectively. Label one indicates inclusion in the final mesh, and label zero indicates exclusion. The optimization problem is formulated as follows:

$$max\ \{b_{f_i}, \dots, b_{f_n}\} \sum_{i=1}^{n} b_{f_i} * (F_f(f_i) + F_e(f_i)) \quad (7)$$

subject to:

$$2b_{e_i} - \sum_{f_j} b_{f_i} = 0, \forall_{e_i} \quad (8)$$

This formulation ensures that the selected facets fit and cover the input points while forming a valid 2-manifold mesh. The fitting term F_f for each facet candidate is computed as:

$$F_f(f_i) = \sum_{p_j | d(p_j, f_i) < \epsilon} (1 - \frac{d(p_i, f_i)}{\epsilon}) \quad (9)$$

The coverage term F_c ensures that large triangles do not cover empty areas. For each facet candidate, the ϵ-selected input points are projected onto its supporting plane, constructing a 2D alpha shape with alpha set to s times the average spacing of P. The coverage term is then calculated as:

$$F_c(f_i) = (1, \frac{area(\alpha(\{p_j | d(p_j, f_i) < \epsilon\}))}{area(f_i)}) \quad (10)$$

Where s is set to 5 by default.

This meshing process ensures that the final mesh is optimal regarding data fitting and coverage, forming a 2-manifold mesh that accurately represents the input 3D point cloud.

4. IMPLEMENTATION

To apply the VSA technique in Open3D, the researchers needed to enhance the library's mesh processing functionalities by developing custom modules that simulate the clustering, quadric error computation, and meshing logic characteristics of VSA. The components were developed as Python wrapper functions utilizing Open3D's C++ backend for fundamental geometry operations. The integration facilitated modularity and scalability, enabling the VSA capabilities to function concurrently with existing Open3D QEM tools without interfering with native operations. In addition, efforts were made to synchronize data types and memory access patterns with Open3D's core architecture to ensure computational performance on embedded systems.

4.1. Embedded Systems

Implementing the Variational Shape Approximation (VSA) technique within the Open3D framework involves several key strategies to optimize rendering for embedded systems like the Raspberry Pi and Jetson Nano. These include using efficient data structures to minimize memory usage, employing memory profiling tools to identify and reduce memory leaks, and managing data to reduce its footprint using in-place operations. Computational efficiency is achieved by refining algorithms to reduce complexity, leveraging optimized libraries for numerical computations, and precomputing reusable values. Additionally, parallel processing techniques, such as multi-threading and multi-processing, are utilized to distribute workloads across multiple cores. Profiling tools are employed to identify performance bottlenecks, allowing for targeted optimizations to enhance the efficiency of the VSA technique on resource-constrained devices like the Raspberry Pi and Jetson Nano.

5. RESULTS

The researchers implemented the approach entirely in Python, utilizing Open3D (v0.15.0) for point cloud and mesh processing, NumPy [1] and SciPy for numerical and linear algebra operations, and scikit-learn [2] for accuracy evaluation. Custom modules were developed to integrate the Variational Shape Approximation (VSA) into the existing Quadric Error Metrics (QEM) pipeline, enabling efficient mesh simplification on resource-constrained devices.

Development and initial testing were conducted on a Lenovo ThinkPad with a 1.60 GHz octa-core Intel i5 10th Gen processor and 8 GB RAM. Experimental evaluations were carried out on two embedded platforms: the Raspberry Pi 4 (1.5 GHz quad-core ARM Cortex-A72, 8 GB RAM) and the NVIDIA Jetson Nano (128-core Maxwell GPU, 4 GB LPDDR4). These systems were chosen for their accessibility and suitability for embedded 3D processing despite their limited computational power.

Using Euclidean distance, the K-Nearest Neighbors (KNN) model was configured with k = 20. Figures 2–4 illustrate the mesh reconstruction of a bunny model using VSA-enhanced QEM and its results at various mesh complexities and clustering levels. Figure 5 demonstrates the reconstruction of a bolt model, showing improved detail as cluster count increases. In Figure 6, the method reconstructs a point cloud at varying angles and cluster sizes. Figure 7 confirms that higher cluster counts (up to 1000) yield finer mesh details by approximating surfaces with geometric primitives while preserving sharp features.

Performance benchmarking (Figure 11) revealed increasing computation time with iteration count: 12 seconds at 1 iteration, 20 seconds at 5, and 100 seconds at 20, with clustering and meshing times increasing exponentially (Figures 12–13). Despite this, memory usage remained stable across all scenarios. Figures 14–16 show consistent peak memory usage (80–100 MB) across platforms and iterations, indicating efficient memory management by the VSA pipeline.

Accuracy evaluations presented in Figures 17-20 and another in Table 1-3 compare Open3D's QEM and the VSA-enhanced version on various models. While VSA typically yields fewer errors, its advantage varies by model and simplification level. Some models, such as Gear and Armadillo, pose greater challenges for accurate simplification. Approximation errors suggest VSA may not outperform QEM in early iterations, but it shows consistent improvements with more iterations. The figures and Tables 1-3 confirm similar trends on the Jetson Nano, indicating that while error magnitudes vary by hardware, relative performance remains consistent and model-dependent.

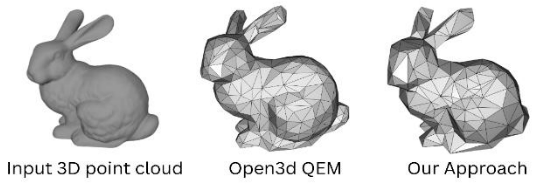

Fig. 2. Reconstructing a bunny. Left: input 3D point cloud. Middle: Open3d QEM with 250 clusters. Right: our approach with 250 clusters.

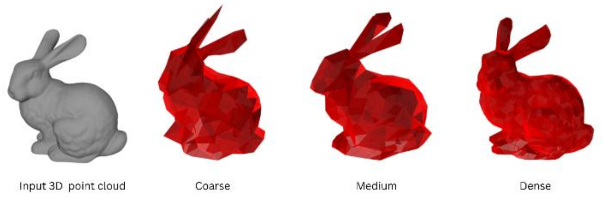

Fig. 3. Reconstructing a bunny in Raspi4. Left: input 3D point cloud. Right: reconstructions with increasing mesh complexity.

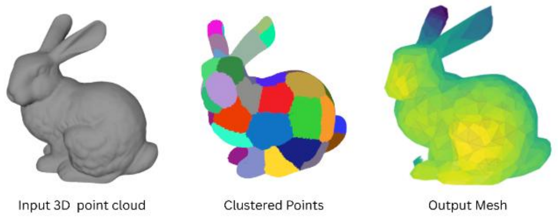

Fig. 4. Reconstructing a bunny in Jetson Nano. Left: input 3D point cloud. Middle: input points after clustering. Right: output meshes and vertices.

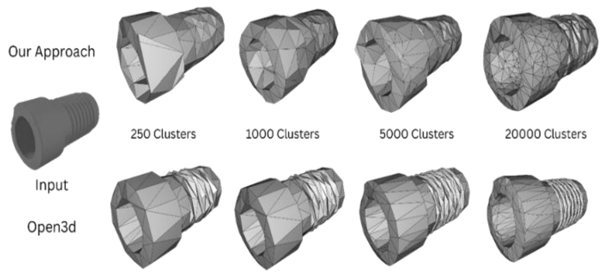

Fig. 5. Reconstructing a bolt model in Lenovo ThinkPad. Left: input 3D points. Top: our reconstruction with increasing numbers of clusters. Bottom: Open3D reconstruction with its native QEM.

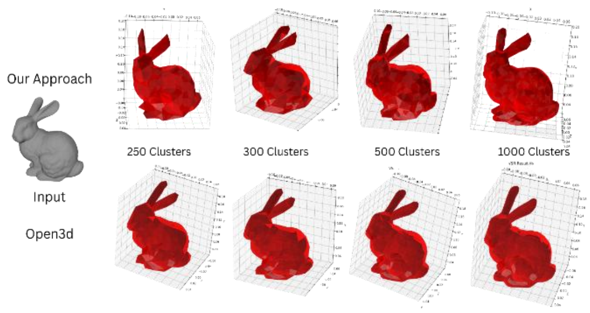

Fig. 6. Reconstructing a bunny model in Raspi4. Left: input 3D points. Top: our reconstruction with increasing numbers of clusters. Bottom: Open3D reconstruction with its native QEM.

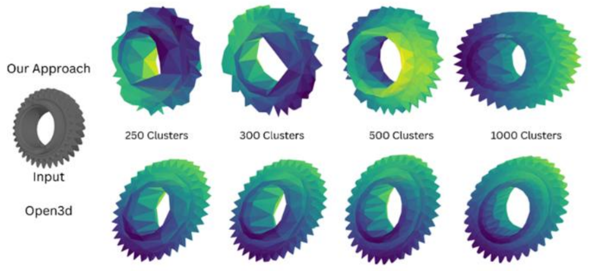

Fig. 7. Reconstructing a bunny model in Jetson Nano. Left: input 3D points. Top: our reconstruction with increasing numbers of clusters. Bottom: Open3D reconstruction with its native QEM.

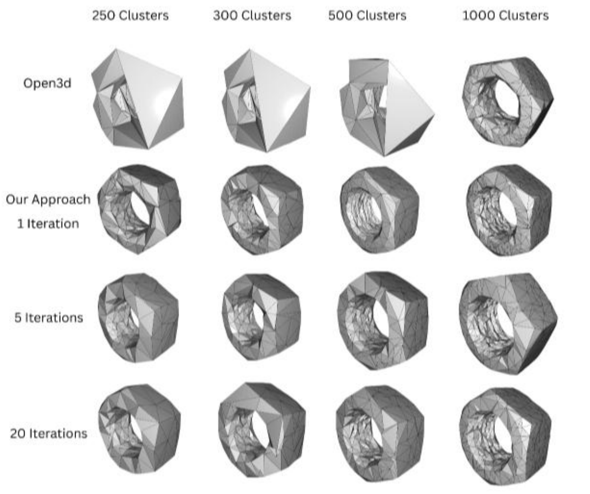

Fig. 8. Comparison of the metal nut model reconstructions using Open3D's native QEM and our VSA-integrated approach with varying cluster sizes (250, 300, 500, 1000) and iterations (1, 5, 20). Lenovo ThinkPad.

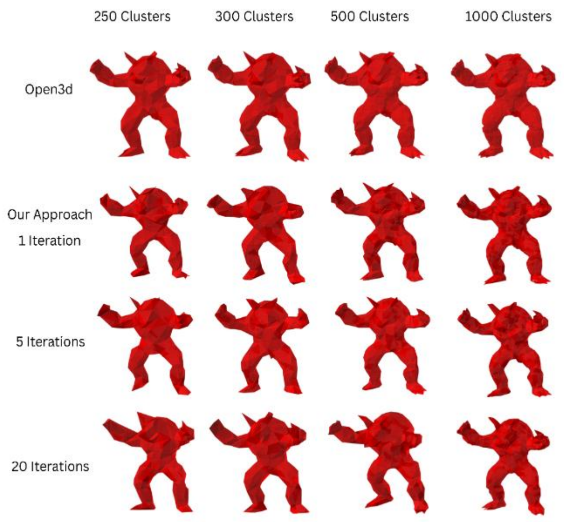

Fig. 9. Comparison of the metal nut model reconstructions using Open3D's native QEM and our VSA-integrated approach with varying cluster sizes (250, 300, 500, 1000) and iterations (1, 5, 20). Raspi4.

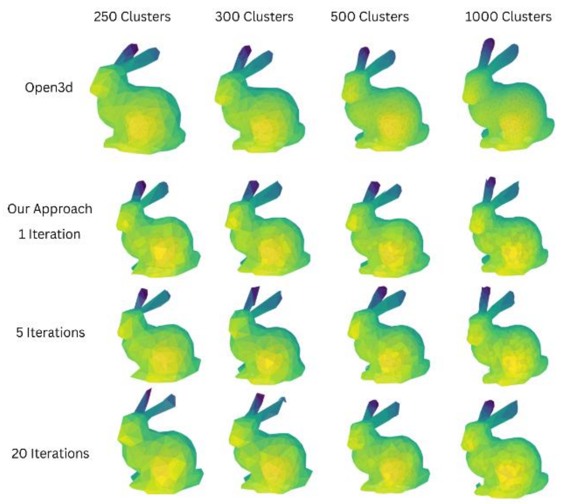

Fig. 10. Comparison of the metal nut model reconstructions using Open3D's native QEM and our VSA-integrated approach with varying cluster sizes (250, 300, 500, 1000) and iterations (1, 5, 20). Jetson Nano.

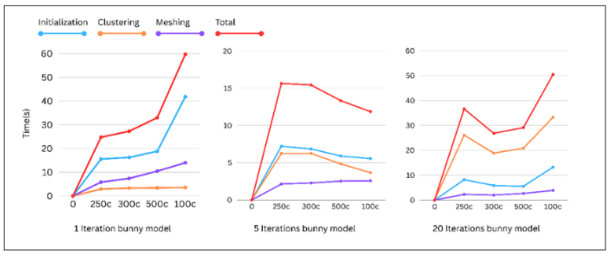

Fig. 11. Computational Time for VSR Mesh Reconstruction on Bunny Model at Varying Iterations (1, 5, and 20) in Lenovo Thinkpad.

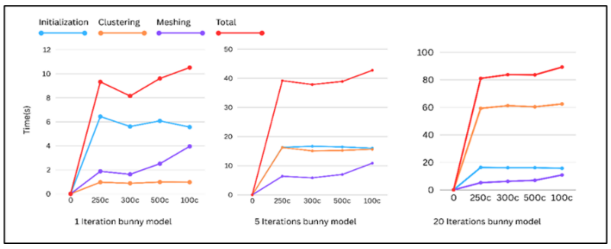

Fig. 12. Computational Time for VSR Mesh Reconstruction on Bunny Model at Varying Iterations (1, 5, and 20) in Raspi4.

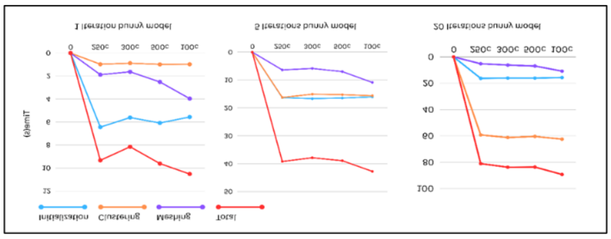

Fig. 13. Computational Time for VSR Mesh Reconstruction on Bunny Model at Varying Iterations (1, 5, and 20) in Jetson Nano.

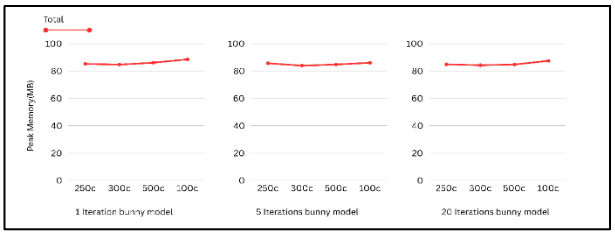

Fig. 14. Memory Usage for VSR Mesh Reconstruction on Bunny Model at Varying Iterations (1, 5, and 20) in Lenovo Thinkpad.

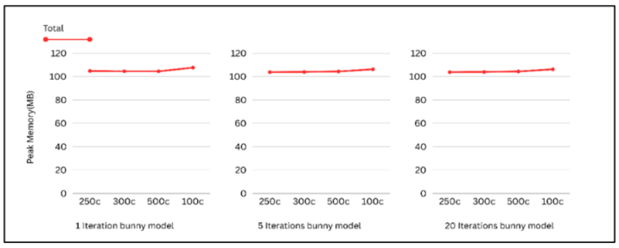

Fig. 15. Memory Usage for VSR Mesh Reconstruction on Bunny Model at Varying Iterations (1, 5, and 20) in Raspi4.

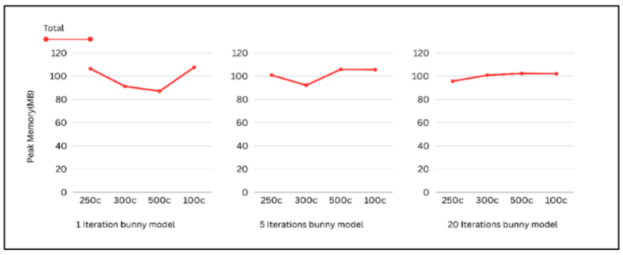

Fig. 16. Memory Usage for VSR Mesh Reconstruction on Bunny Model at Varying Iterations (1, 5, and 20) in Jetson Nano.

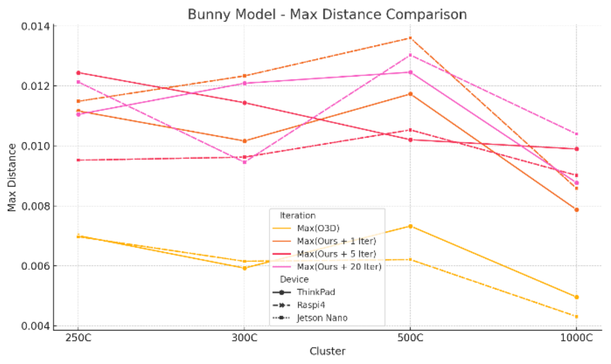

Fig. 17. Maximum distances from 3D input clouds (Bunny model) to output meshes in Lenovo ThinkPad, Raspberry Pi 4, and Jetson Nano. Open3D followed VSA-QEM

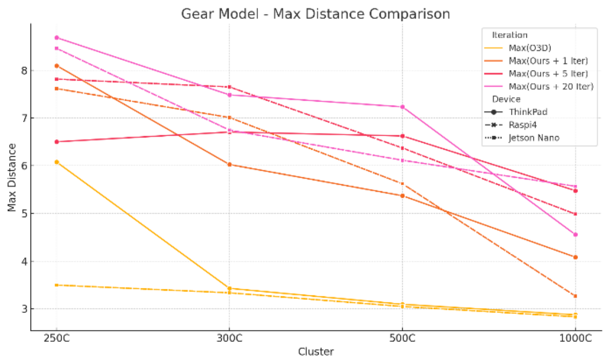

Fig. 18. Maximum distances from 3D input clouds (Gear model) to output meshes in Lenovo ThinkPad, Raspberry Pi 4, and Jetson Nano. Open3D followed VSA-QEM

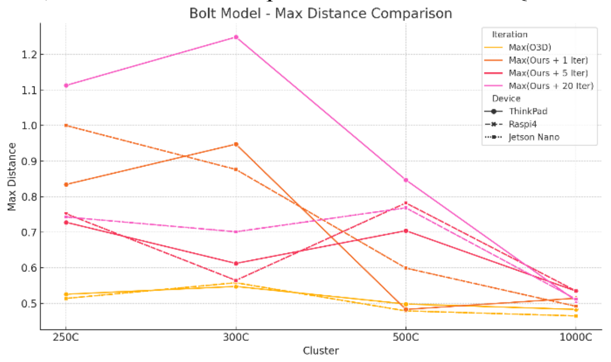

Fig. 19. Maximum distances from 3D input clouds (Bolt model) to output meshes in Lenovo ThinkPad, Raspberry Pi 4, and Jetson Nano. Open3D followed VSA-QEM

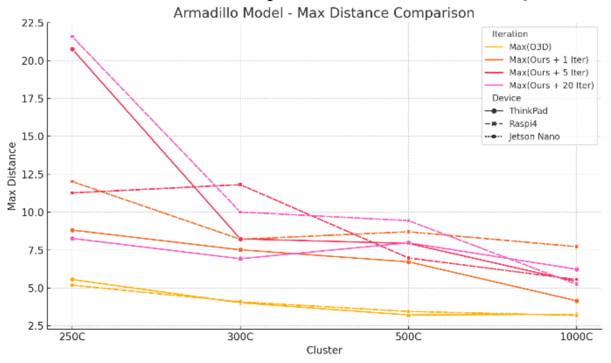

Fig. 20. Maximum distances from 3D input clouds (Armadillo model) to output meshes in Lenovo ThinkPad, Raspberry Pi 4, and Jetson Nano. Open3D followed VSA-QEM

Table 1. Maximum distances from 3D input clouds (Metal Nut) to output meshes in Lenovo ThinkPad. Open3D followed VSA-QEM.

Model	MetalNut(250C)	MetalNut (300C)	Metal Nut (500C)	MetalNut (1000C)
Max(O3D)	Error	Error	4.51878	0.579759
Max(Ours + 1 Iter)	1.120752	0.998271	0.80669	0.682478
Max(Ours + 5 Iter)	0.85153	0.836487	0.73994	0.62326
Max(Ours + 20 Iter)	0.79647	1.036705	0.759702	0.738345

Table 2. Maximum distances from 3D input clouds (Metal Nut) to output meshes in Raspi4. Open3D followed by VSA-QEM.

Model	Metal Nut (250C)	Metal Nut (300C)	Metal Nut (500C)	Metal Nut (1000C)
Max(O3D)	Eror	Error	1.02081	0.579033
Max(Ours + 1 Iter)	0.74082	0.777852	0.686714	0.695344
Max(Ours + 5 Iter)	1.185856	0.867912	0.704319	0.641099
Max(Ours + 20 Iter)	0.807109	1.116414	0.763474	0.712021

Table 3. Maximum distances from 3D input clouds (Metal Nut) to output meshes in Jetson Nano. Open3D followed by VSA-QEM

Model	Metal Nut (250C)	Metal Nut (300C)	Metal Nut (500C)	Metal Nut (1000C)
Max(O3D)	Eror	Error	1.02081	0.579033
Max(Ours + 1 Iter)	0.74082	0.777852	0.686714	0.695344
Max(Ours + 5 Iter)	1.185856	0.867912	0.704319	0.641099
Max(Ours + 20 Iter)	0.807109	1.116414	0.763474	0.712021

6. CONCLUSION

A Variational Shape Approximation (VSA) enhanced Quadric Error Metrics (QEM) is implemented in an open-source 3d library (Open3D) for mesh simplification and shape approximation. The results of this study showed that the number of clusters and iterations significantly impact the process of simplifying input point clouds into an output mesh. On three experimented embedded systems (Lenovo ThinkPad, Raspberry Pi 4, and Jetson Nano), the VSA-enhanced method shows progressive improvement in retaining complex structures, especially in areas such as the face, limbs, and edges of models like the armadillo, bunny, and bolt. With 1 iteration, the reconstructions are relatively coarse, but at 5 and 20 iterations, the mesh becomes noticeably smoother and more faithful to the original shape, especially at cluster sizes of 500 and 1000.
Moreover, the refinement capability of the VSA-enhanced Quadric Error Metrics (QEM) approach significantly outperforms Open3D's native QEM method. Unlike Open3D QEM, which simplifies meshes in a single pass without iterative improvement, the VSA-QEM approach refines mesh quality progressively with each iteration.

7. SIGNIFICANCE

This study contributes to the ongoing discourse on efficient 3D model processing and embedded system optimization. Specifically, it advances the optimization of point clouds into 3D models, supporting applications such as biomedical 3D printing [22, 23], real-time 3D rendering, and LiDAR-based 3D meshing.

8. ACKNOWLEDGEMENT

The researchers gratefully acknowledge the Department of Science and Technology – Philippine Council for Industry, Energy and Emerging Technology Research and Development (DOST-PCIEERD) for funding the PAMANA: Cultural Preservation and Virtual Experience of the Cultural Dances of Caraga project. The researchers also extend their sincere appreciation to Caraga State University, National Commission for Indigenous Peoples (NCIP-Caraga), (CSU), and the National Museum of the Philippines (Butuan) for their invaluable support throughout the implementation of this initiative.

9. REFERENCES

[1] Zhao, T., Busé, L., Cohen-Steiner, D., Boubekeur, T., Thiery, J. M., & Alliez, P. (2023, July). Variational shape reconstruction via quadric error metrics. In ACM SIGGRAPH 2023 Conference Proceedings (pp. 1-2).

[2] M. Garland, P.S. Heckbert, Surface simplification using quadric error metrics, 3D (1997) 209–216.

[3] P. Pellizzoni, G. Savio, Mesh simplification by curvature-enhanced quadratic error metrics, J. Comput. Sci. 16 (2020) 1195–1202.

[4] R.A. Potamias, G. Bouritsas, S. Zafeiriou, Revisiting point cloud simplification: A learnable feature preserving approach, in: Lect. Notes Comput. Sci., (2022), 586–603.

[5] A. Bergman, P. Kellnhofer, Y. Wang, E. Chan, D. Lindell, G. Wetzstein, Generative neural articulated radiance fields, (2022) 334-341.

[6] A. Barda, V.G. Kim, N. Aigerman, A.H. Bermano, T. Groueix, MagicClay: Sculpting meshes with generative neural fields, arXiv preprint arXiv:2403.00000, (2024).

[7] H. Legrand, J. Thiery, T. Boubekeur, Filtered quadrics for high-speed geometry smoothing and clustering, Comput. Graph. Forum 38 (1) (2019) 663–677.

[8] T. Boubekeur, M. Alexa, Phong tessellation, ACM Trans. Graph. 27 (5) (2008) 1–9.

[9] S.K. Yadav, U. Reitebuch, K. Polthier, Robust and high fidelity mesh denoising, IEEE Trans. (2019)

[10] C.H. Lee, A. Varshney, D.W. Jacobs, Feature-aware mesh simplification, IEEE Trans. Vis. Comput. Graph. 19 (4) (2013) 657–666.

[11] D. Cohen-Steiner, P. Alliez, M. Desbrun, Variational shape approximation, ACM Trans. Graph. 23 (3) (2004) 905–914.

[12] M. Yu, F. Lafarge, Finding good configurations of planar primitives in unorganized point clouds, in: Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., (2022) 6367–6376.

[13] J.P. Bauchet, F. Lafarge, Kinetic shape reconstruction, ACM Trans. Graph. 39 (5) (2020) 1–14.

[14] M. Skrodzki, E. Zimmermann, K. Polthier, Variational shape approximation of point set surfaces, Comput. Aided Geom. Des. 80 (2020) 101875.

[15] Q.Y. Zhou, J. Park, V. Koltun, Open3D: A modern library for 3D data processing, arXiv preprint arXiv:1801.00001, (2018)

[16] J. Munier, Investigation of the use of meshfree methods for haptic thermal management of design and simulation of MEMS, CORE Reader, (2017).

[17] Q. Peng, Z.S. GharehTappeh, Simplification and unfolding of 3D mesh models: Review and evaluation of existing tools, Procedia CIRP 92 (2021) 205–210.

[18] H. Xu, B. He, C. Zhang, H. Lin, X. Kuai, R. Guo, Quality-preserving multilevel mesh generation for building models, Int. J. Digit. Earth 17 (1) (2024).

[19] V. Tiwari, M. Meribout, A GPU-enabled tensor core-based multi-frame image reconstruction method for 3D electrical tomography systems, SSRN Electron. J., (2023).

[20] S. Schnitzer, Real-time scheduling for 3D rendering on automotive embedded systems, Doctoral dissertation, Universität Stuttgart, Institut für Parallele und Verteilte Systeme (IPVS), (2019).

[21] Z. Hu, S. Yuan, C. Benghi, J. Zhang, X. Zhang, D. Li, M. Kassem, Geometric optimization of building information models in MEP projects: Algorithms and techniques for improving storage, transmission and display, Autom. Constr. 107 (2019) 102941.

[22] Rashid, A., Rashid, H., Ramlee, M., et.al. (2024). Enhancing 3D Printed Bed-Resting Ankle-Foot Orthosis Design through Topology Optimization. Proceedings of International Exchange and Innovation Conference on Engineering & Sciences (IEICES). Vol. 10, Page 142-147.

[23] Addullah, A., Alimarizan, F., Aziz, N., et.al. Development of Ophthalmology Eye Examination Device Using 3D Printing Technology. (2024). Proceedings of International Exchange and Innovation Conference on Engineering & Sciences (IEICES). Vol. 10, Page 457-462.