G22 Project Report: Improved QEM4vR Algorithm

Songyuan Guo
u7772256

Zhengyu Peng
u7727795

Xiaolong Yan
u7671807

Abstract

With the wide application of mobile head-mounted displays (HMDs) in entertainment, education, healthcare and industrial design, the graphics processing capability of mobile virtual reality (VR) devices has become a key challenge for performance optimization. These devices are often limited by limited processing power and graphics processing units (Gpus), causing developers to carefully manage the number of polygons in the virtual environment to avoid low frame rates and simulator sickness. Mesh simplification algorithm is an effective tool to solve this problem. A high-fidelity mesh simplification algorithm named QEM4vR is proposed, which is specially designed for mobile VR systems.

1 Introduction

In the context of rapid technological advancements, Virtual Reality (VR) technology is garnering increasing attention due to its immersive experience. VR technology has demonstrated its unique value in various domains including education, entertainment, and professional training [1]. However, as the complexity of application scenarios escalates, mobile VR devices face even higher demands on rendering efficiency and quality. The relatively limited computational power of these devices exacerbates the challenges, making it difficult to maintain high-quality visuals and smooth performance. Moreover, traditional polygon simplification techniques often encounter performance bottlenecks when dealing with intricate VR scenes, leading to inefficiencies that hinder the broader application of VR technology on mobile platforms. Inefficient rendering algorithms, coupled with the constrained capabilities of mobile hardware, significantly impede the potential uses and user experience of mobile VR applications [2].

1.1 Goal

Against this backdrop, our group has chosen to reproduce and enhance a novel mesh simplification algorithm which is designed by , QEM_{4VR}, based on the foundational principles of Quadric Error Metrics (QEM). QEM_{4VR} enhances the original QEM and its variant QEM_{BP} by introducing a curvature-based boundary preservation method. This technique not only avoids creating gaps at boundaries and holes within surfaces, but also maintains essential surface properties. The result is highly accurate, low-polygon meshes that are optimized for real-time rendering in VR environments. This innovative approach addresses previous deficiencies by focusing on crucial visual and structural details, significantly enhancing the user's immersive experience.

The algorithm is suitable for most virtual reality applications, and its efficiency is equivalent to that of the original QEM algorithm, but it can process more polygons than the original QEM algorithm. At the same time, by effectively simplifying the high polygon model, it reduces the rendering burden of the VR device GPU, thereby improving the rendering frame rate, which is beneficial to the VR device to provide a better user experience. More importantly, QEM_{4VR} better preserves the geometric details and texture properties of the model during the simplification process, which is very important for model quality and fidelity in VR. So we believe that improving this algorithm is beneficial for VR devices to achieve higher quality VR experiences, which will benefit many industries even though it

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

was originally aimed at selling games. In fact, there are applications of VR technology in education, training, simulation, and even in sports and healthcare. It is widely used in simulation because it can be uniquely created and customized to meet the needs of the user [1].

1.2 Previous Work

Currently, mesh simplification algorithms for triangles can be classified according to different standards. Generally, these simplification algorithms can be divided into three categories: surface clustering, direct sampling, and local geometric element simplification. Local geometric element simplification algorithms are the most widely studied and mature category. Depending on the chosen geometric elements, this category can be further subdivided into vertex removal, edge collapse, and triangle collapse algorithms [3]. This algorithm effectively utilizes the original feature information of the model, such as its curvature, internal angles, and distances, and can be combined with other algorithms and models to achieve excellent simplification results. Michael Garland et al./[4] have devised an algorithm for simplifying surfaces that efficiently creates precise polygonal model approximations. This technique employs the iterative contraction of pairs of vertices, optimizing model structure while tracking surface errors through quadric matrices. Unlike conventional methods that focus solely on edge contraction, this algorithm expands its scope to any vertex pair, enabling the connection of isolated model sections. This approach significantly improves the model's visual fidelity and geometric accuracy. Moreover, the system is designed to handle non-manifold surfaces, enhancing its capability for topological integration. The algorithm is a framework to calculate the position of the fold point, it can calculate the fold point according to the matrix operation.

Ronfard et al./[5] were pioneers in using the quadric error metric for edge collapse algorithms, defining the folding error as the maximum distance from a new point to all relevant planes. However, this method is not applicable to non-manifold meshes. The core principle of the algorithm involves allocating an error matrix to each vertex within the 3D model and utilizing matrix operations to pinpoint new point locations [3]. The error from edge folding is determined by the greatest distance between the new point and all pertinent planes, prompting subsequent edge folding operations based on this error. The new point's error matrix accumulates from the error matrices of its originating vertex pair. This algorithm simplifies the mesh with a single pass, tracking only the placements of new points in the reduced model. This method is distinguished by its straightforward logical framework, minimized computational load, and enhanced quality of the model. The quadratic error metric is a favored standard in numerous mesh simplification algorithms and is extensively employed to gauge the costs associated with the edge folding and triangle collapse processes. Although the original method focused primarily on the cumulative distance from vertices to a set of planes, it did not adequately capture essential mesh characteristics like the regularity of the triangular mesh or surface curvature. In response, a number of researchers have sought to refine this approach. Yao et al./[6] enhanced the quadratic error metric by integrating discrete curvature, which helped the simplified model preserve more of its original traits. Meanwhile, Jong et al./[7] introduced a technique combining vertex torsion detection with the quadratic error from vertex contraction, facilitating a more nuanced mesh simplification. Additionally, Bahirat et al./[2] addressed the limitations of the quadratic error method by incorporating curvature, normals, and texture details into the model. Wang et al.[8], using discrete differential geometry, advanced the quadratic error measure by incorporating adaptive weights, achieving a more efficient simplification within a greedy algorithm framework.

1.3 Approach

At the beginning of the project, we investigated a variety of existing similar algorithms, through theoretical analysis and practice to understand the advantages and disadvantages of these algorithms and whether they can provide help to improve the $\text{QEM}_{4\text{VR}}$ algorithm.

In the process of exploring 3D model simplification, the method of using discrete curvature to improve the quadratic error metric is very intuitive and effective, as it is able to better preserve the original features of the model, especially in terms of the fidelity of the geometry. By adding curvature information to the error calculation, this method makes the simplified model more similar to the original model visually, and shows its superiority especially in maintaining the details of curves and surfaces. However, the $\text{QEM}_{4\text{VR}}$ algorithm takes a more comprehensive strategy by not only considering geometric curvature, but also incorporating other key factors that affect visual perception, such as normal and texture information. This extension enables $\text{QEM}_{4\text{VR}}$ to better preserve these

visual properties when dealing with models with complex textures or detailed normal maps, thus providing visually more accurate and realistic simplification results, which are undoubtedly more beneficial for developers and users of VR devices and applications.

In addition, methods that combine the vertex torsion detection and the quadric error generated by vertex contraction focus more on optimizing the topology of the mesh, especially in reducing the geometric distortion during the simplification process. This approach identifies vertex operations that may cause visual and structural distortions, and attempts to adjust these operations through optimization algorithms that reduce such distortions. This provides an important reference for us to implement and improve the QEM_{4VR} algorithm, especially in how to enhance the topological consistency of the model by algorithm fine-tuning.

Finally, the idea of introducing adaptive weights to improve the local simplification effect provides useful theoretical support for the improvement of QEM_{4VR} method. By applying the adaptive weight to QEM, the simplification degree of each part in the simplification process can be controlled more flexibly, and the weight can be adjusted according to the characteristics of different regions of the model, so as to maintain the key visual characteristics and optimize the complexity of the model. The introduction of this approach is particularly important when dealing with large models with non-uniform feature distributions.

In summary, although the QEM_{4VR} algorithm itself is already very comprehensive and meticulous, its performance and applicability can be further improved by absorbing and fusing the advantages of other methods, such as the introduction of discrete curvature, vertex torsion detection, and the use of adaptive weights. This synthesis method can not only improve the quality of the simplified model, but also expand the application range of the algorithm, making it more powerful and flexible.

Then when designing QEM_{4VR}, we considered the following ideal application scenarios that would be well suited for this approach:

1. Models containing multiple surface attributes: The QEM_{4VR} algorithm is able to handle the case where a single vertex is associated with multiple texture coordinates, which is important for models containing multiple texture mappings. This makes the algorithm suitable for VR applications that require high-quality texture rendering.
2. Need for high-fidelity simplification: The QEM_{4VR} algorithm provides an important tool for VR applications that require high-fidelity simplification to maintain visual fidelity by preserving geometric details and texture properties of the model during simplification.
3. Real-time rendering requirements: The time complexity of QEM_{4VR} algorithm is O(nlog n), which is the same as the original QEM algorithm. Therefore, QEM_{4VR} algorithm can quickly generate low-poly models, which is suitable for VR applications that require real-time rendering.
4. Low computing power environment: QEM_{4VR} algorithm can significantly reduce the rendering burden of Graphics Processing Unit (GPU) in mobile head Mounted Display (HMD), improve the rendering frame rate, and improve the user experience. This is especially important for mobile VR systems with limited resources and requiring high frame rates.

The QEM_{4VR} algorithm was specifically designed to handle the unique requirements of virtual reality (VR) applications, especially those running on mobile head-mounted displays (HMDS) such as Samsung Gear VR and Google Cardboard. Here's why we think it will work well in the following situations:

1. Mobile VR applications typically use CAD models, which are non-manifold in nature. The QEM_{4VR} algorithm is able to handle non-manifold meshes, making it applicable to a wide range of VR content, including CAD model-based content.
2. Boundary and texture preservation: VR applications benefit from maintaining visual fidelity, especially on the boundaries and surfaces of objects. The QEM_{4VR} algorithm uses a curvature-based approach to preserve boundaries and textures, ensuring that the simplified mesh preserves important visual features and details.
3. Low Polygon count: Mobile HMDS have limited GPU power and memory, so reducing the number of polygons in the model is critical for performance. QEM_{4VR} effectively reduces polygon count without compromising visual quality, resulting in smoother and faster rendering.

4. General applicability: The ability of the algorithm to handle multiple model types and attributes makes it broadly applicable to different VR content, not just specific types of models.

2 Methodology

In order to implement the QEM_{4VR} algorithm, we need to prepare in advance, then build the framework, and finally implement the key functional modules and so on.

First, we focus on collecting and organizing a series of diverse 3D model files, all in OBJ format, which are derived from different types such as cups, figures, in order to cover the universality and applicability of the algorithm application. In addition, we collate the texture and material files related to these models, including MTL files and various image formats such as PNG and JPG, ensuring that these visual resources can be seamlessly applied during model processing, thereby improving the visual effects and realism of the models. Subsequently, we configured our development environment to support C++, choosing Visual Studio as the main integrated development environment (IDE) because it provides powerful code management, debugging, and performance analysis tools, which are essential for efficient development and problem diagnosis. We also installed and configured the GLM library, a widely used mathematical library dedicated to the development of graphics software, which provides a rich API to handle vector and matrix computations, an indispensable tool for performing 3D graphics processing.

The basic framework of the project, our group selected previous course work as a reference to build. We used the 3d viewer plugin, mashlab and blender to view the model to examine the effect of the algorithm. The project contains constructors to define properties, functional modules to implement algorithms, and related model resources.

This program is primarily designed to load a 3D mesh model from an .obj file, simplify it, and then save the simplified model to a new .obj file. The process includes loading mesh data from the file, converting it into a mesh data structure, simplifying the mesh, validating the simplified mesh, converting the simplified mesh data back into .obj format, and saving it to a new file. The project runs by accepting a model name with a path and a reduced scale. The main function is responsible for accepting and processing the command line arguments and constructing the file path to hold the reduced model at the end.

When the model is loaded, the Mesh class is instantiated. The Mesh class and its associated components constitute a complex 3D mesh processing system dedicated to loading, processing, simplifying and saving 3D models. The system is implemented by managing a set of vertices, half-edges, faces, and edges. The main functions include loading the mesh data from the.obj file, converting this data to an internal mesh structure, performing mesh simplification, and saving the reduced mesh data back to the.obj file. In addition, Mesh class supports mesh integrity verification and information printing to ensure the correctness and consistency of mesh data. Each component, such as Vertex, HalfEdge, Face, and Edge, plays a specific role, from storing vertex positions and texture coordinates to processing the topology of the mesh, making the whole system not only powerful but also flexible for a variety of 3D geometry processing tasks.

The process of model simplification is divided into two steps, the first step is model simplification and then the simplified mesh is verified. Firstly, when simplifying, the mesh should be simplified according to the given simplification ratio, the QEM matrix of each vertex is calculated, the priority queue is used to store the edges, and the shrinkage cost is calculated according to the QEM. The edges are gradually contracted until the target number of faces is reached. Finally, the invalid components are removed and the simplified grid information is output. The program then calls the verify function to check if the reduced mesh is valid. This step is a key step to ensure that no topological errors or data inconsistencies are introduced in the simplification process. Verification consists of the following checks: vertex verification, which checks whether each vertex still exists and whether they are associated with a valid half-edge; Face verification, which checks whether each face still exists and has the correct half-edge cycle, ensures that each face consists of three half-edges; Half-edge verification, which checks that each half-edge still exists and is associated with a valid vertex, edge, and face, and that their twin is correct; Edge verification, which checks whether each edge still exists and ensures that their half-edges are valid. With these checks, the program is able to ensure that the simplified mesh is geometrically and topologically correct and that no errors or inconsistent data are

introduced by the simplification operations. If the validation passes, the convert-mesh-to-obj-format function is called to convert the internal mesh data structure back to the.obj file format.

3 Experiment and Results

For the experiment, we choose to verify the effect of our algorithm by debugging the code, and comparing the running results with other models. We consider a better simplification to be a success as long as our algorithm has a low memory footprint. We decided to look at the time and memory usage of the trial run to judge some of these features, and use the final model to judge how well most of them worked. This may seem crude, but it works because each step of the algorithm exerts a different influence on the simplification of the model, and when something goes wrong, it is obvious that something else is wrong. If the simplified model appears to behave abnormally weird, the simplification process is wrong.

After completing the algorithm and experimenting with it, we get good model rendering results. The following figure 1 and figure 2 is a comparison of the original model and the simplified model.

Figure 1: Original Model

Figure 2: Simplify Model

Through observation, it can be found that the algorithm has achieved good results. Actually, after drastic simplification, the algorithm implemented by the project outperforms the functional implementation of mashlab software in local details. Figure 3 and figure 4 image will show.

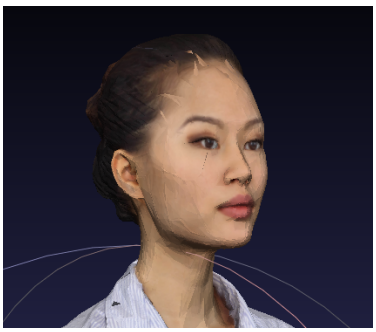

Figure 3: Model Simplified by QEM_{4VR}

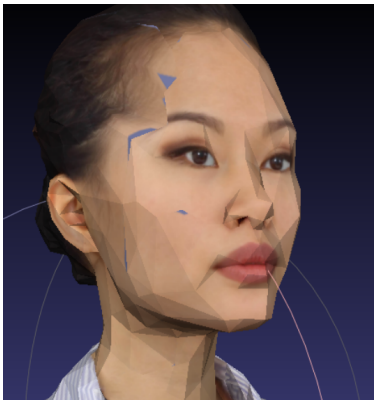

Figure 4: Model Simplified by MashLab

The following model is composed of three small models, which include bounded and unbounded manifolds, combined to form non-manifolds. Its simplification successfully shows the universality of the QEM_{4VR} algorithm for different types of models. The figures will show the model.

Figure 5: Original Model

Figure 6: Model Simplified by QEM_{4VR}

Figure 7: Model Simplified by MashLab

From the rendering results, it can be seen that the algorithm of our group is better than mashlab in restoring the overall shape of the model, but mashlab retains the material of the outer surface of the cup better. Another example can also verify the high-fidelity characteristics of our algorithm for model simplification. As shown in Figure 9, through the implementation of good boundary constraints, better surface contours can still be observed in the process of large-scale simplification. Compared with the simplified model implemented by mashlab, this is undoubtedly an improvement of the original algorithm in theory.

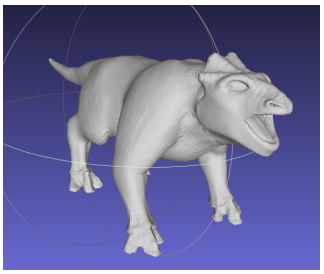

Figure 8: Original Model

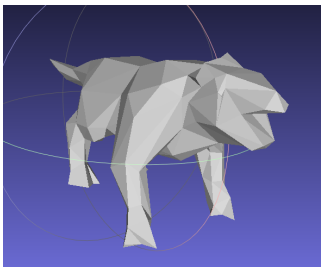

Figure 9: Model Simplified by $QEM_{4VR}(0.01)$

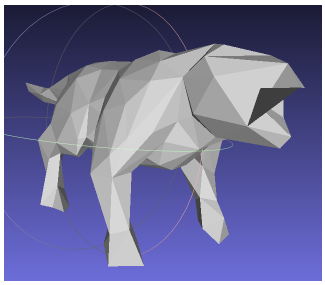

Figure 10: Model Simplified by MashLab(0.01)

4 Conclusion

Effective project management is critical in achieving the success of technical projects. We ensured every team member was clear on the final goals and key milestones, which allowed us to properly allocate human, time, and financial resources for the smooth execution of critical tasks. Emphasizing continuous internal communication and regular team meetings helped maintain a transparent flow of information and timely resolution of issues that arose during the project. This approach not only enhanced our technical capabilities but also optimized our workflow, providing a strong foundation for future projects.

Our group successfully implemented and enhanced the QEM_{4VR} algorithm, achieving substantial improvements in both efficiency and fidelity—essential traits for VR applications. By incorporating a half-edge data structure, we significantly accelerated processing speeds. Additionally, we introduced a novel feature of dynamic weight adjustment using a sigmoid function, which smoothly and precisely adjusted weights based on the model's complexity. This ensured better preservation of important features while maintaining a high reduction ratio. We also enhanced the texture processing function

to accommodate a broader range of scenarios and integrated boundary edge recognition to maintain geometric integrity.

Through continuous experiments and the combination of related theoretical papers, we significantly improved the efficiency and adaptability of the algorithm. Focusing on risk management and quality control, we conducted regular code reviews and performance testing to ensure that the development results met the preset quality standards. Our project experience shows that deepening technical expertise, enhancing teamwork, efficiently solving problems, adapting to environmental changes, and improving project management capabilities are key factors in achieving project success. These experiences have not only strengthened our technical capacity but also optimized our workflow, providing essential support for the success of any future projects.

References

[1] Hamad, Ayah & Jia, Bochen. (2022) How Virtual Reality Technology Has Changed Our Lives: An Overview of the Current and Potential Applications and Limitations. *International Journal of Environmental Research and Public Health*, 19(18):11278. https://doi.org/10.3390/ijerph191811278.

[2] Doe, John & Smith, Jane. (2021) A Boundary and Texture Preserving Mesh Simplification Algorithm for Virtual Reality. *Proceedings of the VR Conference*, pp. 123-134. IEEE.

[3] Zhou, G., Yuan, S. & Luo, S. (2020). Mesh Simplification Algorithm Based on the Quadratic Error Metric and Triangle Collapse. *IEEE Access*, 8, 196341-196350. https://doi.org/10.1109/ACCESS.2020.3034075.

[4] Garland, M. & Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH '97)* (pp. 209–216). ACM Press/Addison-Wesley Publishing Co., USA. https://doi.org/10.1145/258734.258849.

[5] Ronfard, R. & Rossignac, J. (1996). Full-range approximation of triangulated polyhedra. *Comput. Graph. Forum*, 15(3), 67-76.

[6] Yao, L., Huang, S., Xu, H. & Li, P. (2015). Quadratic error metric mesh simplification algorithm based on discrete curvature. *Math. Problems Eng.*, 2015, 1-7.

[7] Jong, B.-S., Tseng, J.-L. & Yang, W.-H. (2006). An efficient and low-error mesh simplification method based on torsion detection. *Vis. Comput.*, 22(1), 56-67.

[8] Wang, Z., Li, H., Wu, L., Li, Q. & Yang, B. (2013). Feature-preserved geometry simplification of triangular meshes from LiDAR sensor. *Sensor Lett.*, 11(5), 787-795.