# Designing and Evaluating a Mesh Simplification Algorithm for Virtual Reality

KANCHAN BAHIRAT, CHENGYUAN LAI, RYAN P. MCMAHAN,
and BALAKRISHNAN PRABHAKARAN, The University of Texas at Dallas



With the increasing accessibility of the mobile head-mounted displays (HMDs), mobile virtual reality (VR) systems are finding applications in various areas. However, mobile HMDs are highly constrained with limited graphics processing units (GPUs) and low processing power and onboard memory. Hence, VR developers must be cognizant of the number of polygons contained within their virtual environments to avoid rendering at low frame rates and inducing simulator sickness. The most robust and rapid approach to keeping the overall number of polygons low is to use mesh simplification algorithms to create low-poly versions of pre-existing, high-poly models. Unfortunately, most existing mesh simplification algorithms cannot adequately handle meshes with lots of boundaries or nonmanifold meshes, which are common attributes of many 3D models.

In this article, we present $QEM_{4VR}$, a high-fidelity mesh simplification algorithm specifically designed for VR. This algorithm addresses the deficiencies of prior quadric error metric (QEM) approaches by leveraging the insight that the most relevant boundary edges lie along curvatures while linear boundary edges can be collapsed. Additionally, our algorithm preserves key surface properties, such as normals, texture coordinates, colors, and materials, as it preprocesses 3D models and generates their low-poly approximations offline.

We evaluated the effectiveness of our $QEM_{4VR}$ algorithm by comparing its simplified-mesh results to those of prior QEM variations in terms of geometric approximation error, texture error, progressive approximation errors, frame rate impact, and perceptual quality measures. We found that $QEM_{4VR}$ consistently yielded simplified meshes with less geometric approximation error and texture error than the prior QEM variations. It afforded better frame rates than QEM variations with boundary preservation constraints that create unnecessary lower bounds on overall polygon count reduction. Our evaluation revealed that $QEM_{4VR}$ did not fair well in terms of existing perceptual distance measurements, but human-based inspections demonstrate that these algorithmic measurements are not suitable substitutes for actual human perception. In turn, we present a user-based methodology for evaluating the perceptual qualities of mesh simplification algorithms.

CCS Concepts: • **Computing methodologies** → *Mesh geometry models*;

Additional Key Words and Phrases: Mesh simplification, quadric error metric, virtual reality

This material is based on work supported by the National Science Foundation (NSF) under Grant No. 1012975 and US Army Research Office (ARO) W911NF-17-1-0299. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF and ARO.

Authors' addresses: K. Bahirat, C. Lai, R. P. Mcmahan, and B. Prabhakaran, Department of Computer Science, Erik Jonsson School of Engineering and Computer Science, The University of Texas at Dallas, 800 W. Campbell Road, MS EC31, Richardson, TX 75080 U.S.A; email: {Kanchan.Bahirat, Chengyuan.Lai, rymcmaha, bprabhakaran}@utdallas.edu.


https://doi.org/10.1145/3209661

**ACM Reference format:**
Kanchan Bahirat, Chengyuan Lai, Ryan P. Mcmahan, and Balakrishnan Prabhakaran. 2018. Designing and Evaluating a Mesh Simplification Algorithm for Virtual Reality. *ACM Trans. Multimedia Comput. Commun. Appl.* 14, 3s, Article 63 (June 2018), 26 pages.
https://doi.org/10.1145/3209661

## 1 INTRODUCTION

Mobile virtual reality (VR) has garnered a lot of public attention during the past couple years. The main reason for this attention has been the commercial introduction of mobile head-mounted displays (HMDs) that run on smartphones fitted into head-based peripherals, such as the Samsung Gear VR and Google Cardboard. These mobile HMDs have several advantages over traditional desktop-based VR systems. First, because these systems are self-contained and tetherless [1], mobile HMDs can be used nearly anywhere, including outdoors and in social gatherings [2]. Second, these systems are relatively affordable for the general population, as they only require a smartphone and a head-mounted peripheral [3]. In turn, the low cost of these commercial devices makes them broadly accessible [4]. Due to these advantages, researchers have investigated mobile HMDs for treating amblyopia [3], self-managing pain [5], and geometry education [6].

However, mobile HMDs are severely limited for running VR applications compared to traditional VR systems. One of the biggest limitations for mobile HMDs is their limited graphics processing units (GPUs), which are over an order of magnitude slower than desktop-based GPUs [1]. These mobile GPUs cannot handle the same virtual environments and objects as desktop-based VR systems without rendering at lower framerates, which can induce simulator sickness [7]. Additionally, mobile HMDs are also constrained with little processing power and onboard memory, leading to upper bounds on data size. For example, the Samsung Gear VR has a limit of 100K polygons [8] and the Microsoft HoloLens has a limit of 900MB memory [9]. While the next generation of devices will be powered with better GPUs, the increasing complexity of the virtual environments created by artists will continue to pose similar performance issues in the future.

There are approaches for addressing the GPU limitations of mobile HMDs. Boos et al. [1] have presented the FlashBack system, which precomputes and caches all possible images that a VR user might encounter instead of relying on real-time graphics rendering. In an evaluation, Boos et al. [1] found that FlashBack delivered better frame rates than a desktop-based VR system for both static and dynamic virtual environments. However, they acknowledged that the FlashBack system cannot handle several currently visible dynamic objects or interactive lightning models, which are important for many VR applications.

Another approach to mitigating the GPU limitations of mobile HMDs is to reduce the overall number of polygons that the GPU must render in real time. One method to accomplish this is to artistically recreate 3D models by designing low-poly mesh versions. However, this method is extremely intensive in terms of an artist's time. Alternatively, mesh simplification algorithms can be used to rapidly create low-poly versions of pre-existing, high-poly 3D models. However, many existing mesh simplification algorithms are not able to adequately handle manifold meshes with boundaries and nonmanifold meshes. Many models, especially those made using computer-aided design (CAD) tools, are nonmanifold meshes with boundaries [10].

A manifold is a mesh in which all faces that share any given vertex form either a disc or a half-disc [11]. In a manifold without boundaries, every edge is shared by exactly two faces. However, a manifold with boundaries will contain edges that have exactly one adjacent face. A mesh that contains one or more vertices with adjacent faces that do not form a disc or half-disc is considered a nonmanifold. See Figure 1 for examples of manifold and nonmanifold meshes.

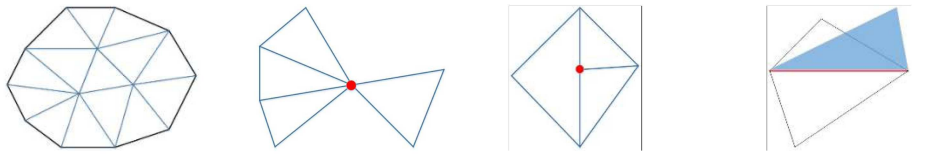



Fig. 1. Examples of 2D manifold and nonmanifold meshes.

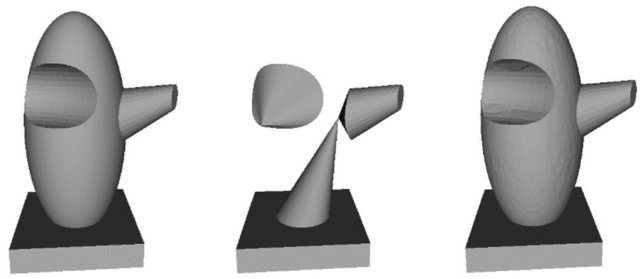



Fig. 2. Example of the $QEM_{BP}$ approach's inability to preserve important surface edges.

Most mesh simplification algorithms require a manifold mesh [11] as input [10]. As a result, only a few mesh simplification algorithms can handle most 3D models without failing. Of the simplification algorithms that can handle nonmanifold meshes, the quadric error metric (QEM) approach is one of the most popular due to its speed and accuracy [12]. However, because the original algorithm [13] allows for the collapse of boundary edges, it has the tendency to create large gaps at boundaries (see Figure 4(b)). To address this, Garland and Heckbert [14] provided a modification for boundary preservation ($QEM_{BP}$) by weighting the boundaries when deciding which edges to keep. Unfortunately, this algorithm tends to preserve unnecessary boundary edges and collapsing needed surface edges, resulting in degraded surface approximations (see Figure 2(b)).

To address the requirements of VR applications and mobile HMDs, we developed a mesh simplification algorithm utilizing the QEM infrastructure that eliminates the deficiencies of QEM and $QEM_{BP}$. This new algorithm, called $QEM_{4VR}$, uses a curvature-based boundary preservation approach to maintain key boundary edges while not sacrificing necessary surface edges [15]. This curvature-based approach avoids creating gaps at boundaries and holes within surfaces (see Figure 7(d)). In addition to yielding more accurate, low-poly meshes, it also preserves key surface properties, such as normals and materials (see Figure 9(f)).

In this article, we use six publicly available, high-poly models with and without textures to compare the accuracy and fidelity of our $QEM_{4VR}$ algorithm to previous QEM variations. Our results indicate that as meshes are simplified to approximately 10% of their original polygon totals, $QEM_{4VR}$ maintains more accuracy with fewer errors than the prior algorithms. Additionally, the low-poly meshes generated by $QEM_{4VR}$ have high-fidelity appearances compared to the original high-poly models, especially those with textures. Furthermore, the experimental results demonstrate that with $QEM_{4VR}$, we can achieve a frame rate of approximately 60 frames per second while retaining high visual quality. Hence, $QEM_{4VR}$ is a suitable algorithm for creating low-poly meshes for most VR applications, including mobile HMDs.

As VR applications are user centric, it is crucial to evaluate the perceptual quality of any mesh simplification method. In this article, we investigate the suitability of existing perceptual mesh quality metrics for evaluating mesh simplification algorithms. Our investigation indicates that existing perceptual quality measures are unable to incorporate the completeness of the model, gaps, and texture distortion, and fail to adequately represent the holistic, perceptual mesh quality. Moreover, this investigation further motivates the need for a user-based methodology to accurately evaluate the human perceptual qualities of simplified meshes. We present such a user-based methodology and briefly describe how future user studies will assist in determining quality boundaries that will enable adaptive streaming and rendering techniques.

## 2 RELATED WORK

Numerous mesh simplification algorithms have been previously proposed (see Cignoni et al. [16] and Luebke [10] for two excellent surveys). However, a key requirement for designing an algorithm for VR is that the mesh simplification approach must be capable of handling nonmanifold meshes, which are typical for models created by hand using CAD tools [10]. Hence, in this section, we distinguish between manifold-only algorithms and non-manifold-capable algorithms.

### 2.1 Manifold-Only Simplification Algorithms

One approach to simplifying manifold meshes is to optimize the original one. Hoppe et al. [17] presented a mesh optimization algorithm that produces a new mesh of the same topological type as the original mesh, but with a smaller number of vertices. To produce new meshes, Hoppe et al. [17] defined three types of mesh transformations to apply to the original mesh: edge collapse, edge split, and edge swap. Hoppe et al. [17] randomly use these transformations to produce new candidate manifold meshes and then pick the candidate that minimizes an energy function that represents the error of the mesh.

Hoppe [18] later defined the concept of a progressive mesh, which represents a manifold mesh as a sequence of edge collapses. The original mesh can be retained by applying a series of vertex splits to the simplified mesh. Hoppe [18] also defined a new energy function to use during simplification to better represent mesh complexity and maintain a higher degree of fidelity.

Another approach to simplifying a manifold mesh is surface retiling, which involves defining a new set of vertices projected onto the surface of the original mesh and then using those vertices to form new surface faces. The surface retiling algorithm presented by Turk [19] first projects a new set of vertices onto the surface of the mesh, usually in a uniform pattern and spacing, though the curvature of a surface could be taken into account. The fidelity of this approach is highly dependent on the method chosen to project the new set of vertices upon the surface.

The voxel-based simplification algorithm presented by He et al. [20] first segments a mesh into a voxel-based grid, and then applies a low-pass filter to eliminate voxels with the least overlapping volumes with the original mesh. The algorithm then uses the marching cubes algorithm [21] to generate a new mesh based on the remaining voxels. Due to its volume-based filtering approach, voxel-based simplification performs poorly on models with sharp edges and squared corners [10].

Cohen et al. [22] presented another approach to simplifying manifold meshes by using two offset copies of every mesh surface, called simplification envelopes. These offsets were used to ensure that a simplified surface would remain within the volume formed by the envelopes. While simplification envelopes can guarantee a high degree of fidelity given the offset distance used for the envelopes, the distance also serves as a lower bound for the algorithm and prevents drastic mesh simplification.

Yet another approach to simplifying manifold meshes is to use multiresolution analysis to decompose a function, representing the original high-poly mesh, into a simpler function,

representing a low-poly mesh [23]. Eck et al. [24] presented an adaptive subdivision algorithm using multiresolution analysis to simplify arbitrary manifold meshes. The key contribution of their work was the design of a continuous parametrization of an arbitrary mesh over a simple domain mesh. The fidelity of the algorithm presented by Eck et al. [24] is relatively high for smooth organic forms [10], but it fails to capture sharp features unless those features correlate to divisions in the base mesh [18]. Hosseini et al. [25] presented an adaptive 3D texture streaming approach for M3G-based mobile games. As this method aims at solely reducing transmission latency, it may not handle complex models for VR.

In summary, there have been many approaches to simplifying manifold meshes. Each approach has its strengths and weaknesses, in terms of the fidelity of the simplified meshes, the ability to define the degree of error, time performance, and the complexity of their implementations. However, these algorithms do not handle nonmanifolds and, therefore, are inadequate for most VR applications.

### 2.2 Nonmanifold Simplification Algorithms

One of the first mesh simplification algorithms capable of handling nonmanifold meshes was the decimation algorithm presented by Schroeder et al. [26]. The decimation algorithm operates by making multiple passes over all the vertices of the model. During each pass, for each vertex, the algorithm determines whether a vertex can be removed without violating the topology of the local neighborhood of faces. If the resulting surface would be within a user-defined distance of the original mesh, the algorithm removes the vertex and all of its associated faces. The resulting hole is then retriangulated. The decimation algorithm presented is capable of handling nonmanifold meshes, if it does not delete nonmanifold vertices during its removal passes [10]. However, this rule defines a lower bound for the decimation algorithm, as it cannot produce a low-poly mesh with fewer vertices than the number of nonmanifold vertices. Additionally, the user-defined distance has a major impact on both the lower bound and fidelity of the algorithm [13].

Another approach to simplifying nonmanifold meshes is vertex clustering. The vertex clustering algorithm presented by Rossignac and Borrel [27] assigns an importance value to each vertex based on the size of its adjacent faces and its curvature. The algorithm then uses a 3D grid to segment the mesh and collapses all vertices within any given grid cell to the single most important vertex within the cell. The resolution of the grid determines the fidelity of the resulting simplified mesh. However, because the approach does not guarantee the amount of error introduced by the simplification, vertex clustering algorithms are often visually less pleasing than other mesh simplification algorithms [10].

Another simplification approach that relies on vertex clustering is the hierarchical dynamic simplification (HDS) algorithm by Luebke and Erikson [28]. The HDS algorithm represents the entire mesh as a vertex tree, which is a hierarchy consisting of vertex clusters. The nodes of the tree can either be folded into their parent nodes to reduce the overall number of faces or unfolded with their children nodes to reobtain the original mesh. Because the vertex tree does not require vertex connectivity, the HDS algorithm supports nonmanifold meshes. However, its fidelity tends to be less than other simplification algorithms [10].

Perhaps the most common approach to simplifying nonmanifold meshes is to use a QEM, which was first presented by Garland and Heckbert [13]. The QEM of a vertex is a $4 \times 4$ matrix that represents the sum of the squared distances from the vertex to the planes of adjacent faces. When a vertex is merged with another vertex within a user-defined distance threshold, the error introduced can be computed as the sum of the QEMs of the vertices being merged, which becomes the QEM of the new vertex. When merging vertices, the QEM algorithm keeps a sorted priority queue of all candidate vertex pairs based on their merged QEM. The algorithm removes the vertex pair

with the lowest error from the top of the queue, merges the vertices, and then updates the errors of all vertex pairs involving the merged vertex. This approach yields simplified meshes that are relatively high fidelity, even at drastic levels of simplification [10].

One of the limitations of the original QEM algorithm is that it has the tendency to produce large deviations at boundary edges due to the decreased number of adjacent faces [13]. To address this issue and afford boundary preservation, Garland and Heckbert [13] also defined a boundary-constraint plane passing through each boundary edge. For each boundary-constraint plane, they computed a QEM that is multiplied by a high-constant-weight factor and added it to the initial QEMs of the edge vertices. Various researchers [29–31] have also suggested weighing the boundary-constraint plane by the square of the edge length. However, all of these approaches result in prioritizing all boundary edges more than surface edges, which results in simplified meshes with holes in their surfaces.

Another limitation of the original QEM algorithm is that it did not account for surface properties, such as normals, colors, and texture coordinates. Garland and Heckbert [14] generalized the original QEM to also handle surface properties. To handle normals or colors, the original $4 \times 4$ error matrix can be extended to a $6 \times 6$ matrix that contains the error of the vertex's normal (abc) or its color (rgb). Similarly, a $5 \times 5$ error matrix can be used to capture the error of the vertex's texture coordinates (st). Hoppe [12] also presented a generalized QEM that required less storage space than that of Garland and Heckbert [14] by using a wedge-based mesh data structure. Along with his generalized QEM, Hoppe [12] also proposed a memoryless simplification algorithm and a volume-preserving algorithm. More recently, Ovreiu [32] presented two quadric error metrics capable of handling surface attributes.

Since the introduction of QEM, many researchers have explored how to use it for various applications. For example, Pojar and Schmalstieg [33] developed a plugin tool for Autodesk Maya that allows the user to create multiresolution meshes, including from nonmanifold meshes. However, most variations of QEM are suitable for specific types of meshes (e.g., no boundaries, lots of boundaries, textured, etc.) but produce less-than-desirable results when used for other types of meshes. Because VR applications are so diverse and their 3D models can vary drastically (e.g., manifold with no boundaries, manifold with boundaries, nonmanifold), a single mesh simplification approach was not available. We addressed this issue with the presentation of our $\text{QEM}_{4\text{VR}}$ algorithm [15].

### 2.3 Quality-of-Experience Assessments

With the ever-increasing number of interactive multimedia systems, researchers have become more concerned about the Quality of Experience (QoE) afforded by those systems. The International Telecommunication Union has defined QoE as the "overall acceptability of an application or service, as perceived subjectively by the end-user" [34]. However, as Timmerer et al. [35] have pointed out, a user might "accept" a multimedia system "without necessarily being happy or satisfied with it." Hence, we consider the definition of QoE proposed by Raake and Egger [36] to be more appropriate: "[QoE] is the degree of delight or annoyance of a person whose experiencing involves an application, service, or system. It results from the person's evaluation of the fulfillment of his or her expectations and needs with respect to the utility and/or enjoyment in the light of the person's context, personality and current state."

Researchers have investigated various methods to assess overall QoE for immersive systems, such as VR. Many of these methods involve taking both objective and subjective measurements to quantify various aspects of the user's experience. Puig et al. [37] proposed a QoE method primarily focused on objective user performance metrics, particularly task completion times. They conducted a pilot study using the proposed method and found that users became proficient at an

object docking task much faster with a large projection screen than a computer screen. Within the VR community, there have been many studies that have investigated the effects of various interactive systems on user performance metrics. For example, McMahan et al. [38] evaluated the independent effects of and interactions between display fidelity and interaction fidelity on user performance in a first-person shooter VR game. A review of similar user-performance studies is provided by Bowman and McMahan [39].

In addition to task performance, QoE researchers have investigated the use of heart rate (HR) and electrodermal activity (EDA) as objective QoE measures. In an early study, Meehan et al. [40] found that HR significantly correlated to self-reported measures of presence (i.e., the sense of "being there") in an immersive VR system simulating walking above a pit, while EDA and skin temperature did not. More recently, Egan et al. [41] found that EDA was significantly higher for viewing a city landscape virtual environment with a conventional 2D monitor as opposed to an HMD, and that most subjective QoE responses were better for the HMD condition than the 2D monitor condition. Potential reasons for the different EDA effects found by Meehan et al. [40] compared to Egan et al. [41] are the improvement of EDA measurement devices and the contrasting nature of their virtual environments (i.e., acrophobia-oriented pit vs. calm city landscape). In other work, Keighrey et al. [42] used HR and EDA to compare a VR HMD to an Augmented Reality (AR) HMD for a speech and language assessment application. They found that the VR HMD elicited greater levels of HR and EDA, which indicate that the VR HMD yielded greater levels of arousal than the AR HMD.

Another approach to assessing QoE is to investigate the Quality of Sensory Experience (QoSE) [35]. According to Raake and Egger [36], perceiving sensory stimuli is the basis of quality and QoE. Hence, many researchers have sought to establish Just Noticeable Differences (JNDs) for various sensory stimuli. A JND is the minimum amount that a stimulus must be changed for the difference to be noticeable to humans [43]. Many JND studies have focused on the visual qualities of multimedia. Yang et al. [44] established a JND model for spatial masking factors in video encodings. Using a similar model, Zhang et al. [45] were able to estimate JNDs for viewing images by summing the effects of the visual thresholds into subbands. Beyond 2D multimedia, Zhao et al. [46] conducted an experiment to establish a binocular JND (BJND) model for viewing stereoscopic images. Using the BJND model, Hachicha et al. [47] created the objective Stereo Image Quality Assessment (SIQA). More recently, Wu et al. [48] conducted a JND study to identify a new critical quality factor for 3D tele-immersive video called Color-plus-Depth Level-of-Detail (CZLoD). In this article, we discuss how the JND methodology can be used to better assess the visual quality of simplified meshes.

## 3 NEW SIMPLIFICATION ALGORITHM

Considering the various properties of handmade 3D models used in VR applications, we have designed and developed a new mesh simplification algorithm that can effectively create a low-poly version of nearly any 3D model [15]. Our algorithm builds upon the original QEM approach presented by Garland and Heckbert [13]. Hence, we refer to it as $QEM_{4VR}$. Our algorithm overcomes the limitations of the original QEM algorithm and other prior variations by employing a curvature-based boundary preservation approach and considering a wide variety of surface properties, even within the same mesh, when assigning error metrics.

In Section 3.1, we present the basics of the original QEM algorithm for completeness and for those readers unfamiliar with the approach. In Section 3.2, we discuss the boundary-handling limitations of prior algorithms and how our curvature-based approach overcomes those limitations to yield higher-fidelity, low-poly meshes. In Section 3.3, we discuss the surface-handling limitations of previous variations and how our new simplification algorithm accounts for the wide range of

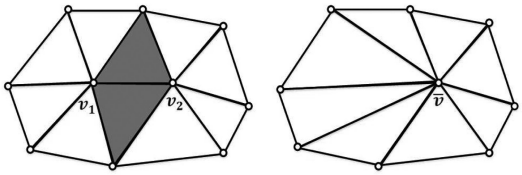



Fig. 3. Example of collapsing an edge to a single vertex through the contraction $(v_1,\ v_2) \rightarrow \bar{v}$.

surface properties that are common with handmade 3D models. Finally, in Section 3.4, we present a summary of the $\text{QEM}_{\text{4VR}}$ algorithm [15].

### 3.1 Basics of Quadric Error Metric

For completeness and those readers unfamiliar with the approach, we discuss the details of the original QEM algorithm [13] here.

*3.1.1 Precleaning of Mesh.* Handmade 3D models may consist of unreferenced, redundant vertices and redundant faces that are generated during the modeling process. Unreferenced vertices may be created due to inappropriate deletion of faces, while redundant vertices and faces are often created due to specific file formats. Unreferenced, redundant vertices add unnecessary computational overhead and can also limit the ability to achieve drastic simplifications. Hence, when using any QEM approach, such redundant vertices and faces should be removed to accelerate processing and improve efficiency. In practice, we use the "Remove Unreferenced Vertices" and "Remove Duplicated Vertices" filters provided by MeshLab [50] for this step.

*3.1.2 Mathematical Formulation.* The original QEM method proposed by Garland and Heckbert [13] is based on the iterative contraction of vertex pairs. A pair contraction represented as $(v_1,\ v_2) \rightarrow \bar{v}$ moves vertices $v_1$ and $v_2$ to new position $\bar{v}$, connects all incident edges on $v_1$ and $v_2$ to $\bar{v}$, and deletes both $v_1$ and $v_2$, along with any degenerate edges and faces (see Figure 3).

At each step of the iteration, a valid vertex pair is chosen for contraction based on the cost of contraction measured by the geometric error of approximation. It characterizes the geometric error of an approximation using the quadric error metric. The QEM associates a set of planes with every vertex of the model. The geometric error induced by the removal of a particular vertex is defined to be the sum of the squared distance of the given vertex to all the planes in the associated set. Each set is initialized with faces incident to the given vertex in the original mesh. When the edge is contracted into a single vertex, the resulting set is the union of the two sets associated with endpoints of the edge. Every face in the original model is defined by a plane represented by the equation $n^T v + d = 0$, where $n = [n_x n_y n_z]^T$ is a unit normal and $d$ is a constant. Given this representation, the squared distance of the vertex $v = [xyz]^T$ to the plane is given by [14]

$$D^2 = (n^T v + d)^2 = (v^T n + d)(n^T v + d) = \mathrm{v}^T (nn^T) v + 2dn^T v + d^2. \tag{1}$$

Garland and Heckbert [14] represented $D^2$ using a convenient representation in terms of a quadric $Q$:

$$Q = (A, b, c) = (nn^T, dn, d^2), \tag{2}$$

and the corresponding quadric error analogous to $D^2$ is

$$Q\,(v) = v^T A v + 2b^T v + c. \tag{3}$$

The addition of two quadrics can be computed component-wise: $Q_1(v) + Q_2(v) = (Q_1 + Q_2)(v)$, where $(Q_1 + Q_2) = (A_1 + A_2, b_1 + b_2, c_1 + c_2)$. Hence, the total quadric error at any vertex can be computed by summing up the quadrics for each plane in the set associated with that vertex. Furthermore, when the edge $(v_1, v_2)$ is collapsed, the resulting quadric is merely an addition of two quadrics given by $Q = Q_1 + Q_2$, and the cost of the contraction $(v_1, v_2) \rightarrow \bar{v}$ is given by $Q(\bar{v}) = Q_1(\bar{v}) + Q_2(\bar{v})$. Hence, the set of planes associated with each vertex is conceptually represented, with no need to maintain it explicitly.

The original quadric metric [13] is heavily influenced by the tessellation or structure of the mesh. As the shape of the surface is more important than the way it is tessellated, the quadric error should be independent of the structure of the mesh. To achieve this, Garland suggested an area-weighted quadric [29].

*3.1.3 Vertex Placement.* When contracting the edge $(v_1, v_2)$, a position for $\bar{v}$ must be made. Garland and Heckbert [14] suggested two options based on the required efficiency and approximation quality: (1) subset placement and (2) optimal placement.

With the subset placement strategy, one of the endpoints is selected as a target position. The endpoint with the smaller $Q(v)$ is contracted into the other endpoint. This strategy produces an approximation using a subset of the original set of vertices with their original positions. Another possibility is to use the midpoint of the edge $(v_1, v_2)$ as a target position.

With the optimal placement strategy, for a given quadric, the target position $\bar{v}$ is computed such that $Q(\bar{v})$ is minimal. As $Q(\bar{v})$is quadratic, its minimum occurs at $\frac{\partial Q}{\partial x} = \frac{\partial Q}{\partial y} = \frac{\partial Q}{\partial z} = 0$. By solving this linear system, we can find the optimal target position and corresponding error as $\bar{v} = -A^{-1}b$ and $Q(\bar{v}) = -b^T A^{-1} b + c$. If the matrix $A$ is not invertible, we can use a subset placement.

Optimal placement results in the better approximation closely fitting to the original mesh. Resultant meshes have more equilateral triangles with uniform areas. On the other hand, subset placement results in inferior approximation but needs significantly less storage space. With both optimal placement and midpoint placement, one needs to store all delta changes for every contraction. This overhead can be avoided in subset placement. Hence, we have chosen to use the subset placement policy in our $\text{QEM}_{\text{4VR}}$ method. However, $\text{QEM}_{\text{4VR}}$ can be easily adapted to any of the above-mentioned policies.

### 3.2 Curvature-Based Boundary Preservation

The key insight that our $\text{QEM}_{\text{4VR}}$ algorithm [15] leverages is the fact that boundary edges are important for the perception of the shape of a mesh, but that not every boundary edge needs to be kept to maintain that shape. In particular, we have used the curvature of a boundary edge vertex to judge the importance of its boundary edges. Thus, our algorithm better maintains both the boundaries and the surface edges of a mesh during simplification, which results in higher-fidelity simplified meshes.

*3.2.1 Limitations of Original QEM Approach.* Generally, sharp edges on the surface of a mesh and its boundaries characterize the object and are considered the most significant visual features [10]. Therefore, it becomes critical to preserve these features for obtaining a better sparse approximation. The original QEM algorithm [13] inherently handles surface shape discontinuities and preserves sharp edges. For example, consider a sharp edge of a cube. For the vertices on the edges of the cube, contributing faces come from the adjacent faces of the cube, which are perpendicular to each other. Hence, the cost of moving the vertex along the edge will be less than moving the vertex away from the edge. Therefore, the original QEM algorithm is strongly biased against altering the sharp features of a mesh's surface.

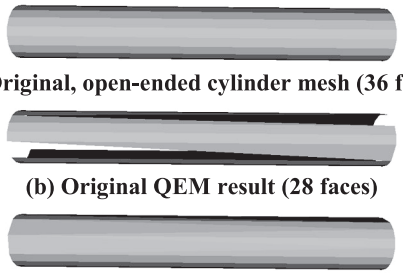



Fig. 4. Example of the original QEM algorithm's inability to preserve important boundary edges.

On the other hand, the original QEM algorithm does not account for actual mesh boundaries (i.e., edges with only one adjacent face). Instead, it has the tendency of prioritizing sharp surface edges over boundary edges. As a result, those boundary edges are removed, which in turn causes holes to form within the surfaces that the algorithm was trying to preserve (see Figure 4(b)).

*3.2.2 Limitations of Prior Boundary Constraints.* Recognizing the original algorithm's inability to preserve important boundary edges, Garland and Heckbert [13] suggested using a boundary preservation approach by marking the boundary edges during the initialization process. For each boundary edge, they computed a plane perpendicular to the edge's face that passed through the edge. Similar to the regular plane, a quadric is computed for these boundary-constraint planes. A constant and large weight factor is applied to the resultant quadric and added to the initial quadrics for both of the boundary edge's vertices. For the area-weighted quadric, they applied a squared length of the boundary edge as the weight factor. Figure 4 shows the results before and after applying the boundary constraints described.

However, this boundary preservation approach is not without its own limitations. The approach is highly biased toward not altering the boundary vertices and may result in inferior results. It will remove most surface edges and their vertices before removing any boundary vertices. For example, consider the mesh drastically simplified with boundary constraints in Figure 2.

*3.2.3 Curvature-Based Boundary Preservation Approach.* To achieve a better boundary constraint, it is important to consider the shape of the boundary curve. For example, the vertices on a linear boundary may be removed in order to keep some of the vertices on the inner surface to get a better approximation. Because the existing boundary preservation approach applies a constant weight factor, it removes the vertices from the inner surface instead of vertices on the linear boundary. Hence, it results in a degraded mesh. Here, we describe our curvature-based boundary preservation approach.

We first identify the boundary edges during initialization. For manifold and nonmanifold surfaces, if the edge is shared by two or more triangles, it is considered interior. If the edge is shared by only one triangle, it is considered as a boundary edge and the corresponding endpoints are considered as boundary vertices.

As the proposed approach is designed for both manifold and nonmanifold meshes, boundary preservation should also handle both types of meshes. First, we discuss the case of a manifold mesh with boundaries. Every boundary vertex will have exactly two neighboring boundary vertices. Consider the boundary vertex $v_1$ and its neighboring boundary vertices $v_2$ and $v_3$. We compute

the curvature of the boundary at vertex $v_1$ as [49]

$$\begin{aligned} x' &= v_3x - v_2x, \quad x'' = v_3x - 2v_1x + v_2x \\ y' &= v_3y - v_2y, \quad y'' = v_3y - 2v_1y + v_2y \\ z' &= v_3z - v_2z, \quad z'' = v_3z - 2v_1z + v_2z \end{aligned} \tag{4}$$

$$k = \frac{\sqrt{(z''y' - y''z')^2 + (x''z' - z''x')^2 + (y''x' - x''y')^2}}{(x'^2 + y'^2 + z'^2)^{\frac{3}{2}}}, \tag{5}$$

where $k$ is a curvature of the boundary curve at $v_1$; $x'$, $y'$, and $z'$ are first-order derivatives; and $x''$, $y''$, and $z''$ are second-order derivatives of the underlying surface in respective directions.

For nonmanifold surfaces, the assumption of exactly two neighboring boundary vertices may not hold true. Consequently, the curvature computed considering any random pair of neighboring boundary vertices will result in an incorrect judgment about the vertex. To handle such scenarios, we simply mark nonmanifold boundary vertices as complex vertices and do not simplify them by assigning them high weights.

Inspired by the original QEM mesh simplification method [13], we compute a boundary-constraint plane and corresponding quadric in a similar manner. Computation of a quadric for boundary-constraint planes allows incorporating boundary preservation without altering the functionality of the original algorithm. Next, we add the boundary-constraint plane's quadric to both endpoints of the boundary edge. Instead of multiplying it with the constant high-weight factor, we multiply the quadric with the curvature at each endpoint independently, and then add it to initial quadrics of the respective vertices. Mathematically,

$$\begin{aligned} Q(v_1) &= Q(v_1) + W_b * k(v_1) * Q_{bcp} \\ Q(v_2) &= Q(v_2) + W_b * k(v_2) * Q_{bcp} \end{aligned} \tag{6}$$

where $Q(v_1)$ and $Q(v_2)$ are the quadrics associated with the vertices $v_1$ and $v_2$, respectively; $Q_{bcp}$ is a quadric of the boundary-constraint plane at edge $(v_1, v_2)$; $W_b$ is the user-defined weight factor; and $k(v_1)$ and $k(v_2)$ are the curvature of the boundaries at the vertices $v_1$ and $v_2$, respectively.

This curvature-based weighing scheme for preserving the boundary assigns a higher weight to boundary vertices with high curvature. The boundary vertices having less curvature (e.g., a vertex on a linear boundary) are assigned smaller weight. It will avoid highly biasing the algorithm toward boundary vertices. Our algorithm removes some of the vertices on the linear boundary to achieve better approximation across the surface edges. See Figures 6 to 11 for results of our curvature-based approach.

### 3.3 Preserving Multiple Surface Properties

Another key contribution of our $QEM_{4VR}$ algorithm [15] is its ability to handle multiple surface properties. Prior QEM variations that preserve surface properties have been presented [12, 14]. However, these approaches make several assumptions regarding surface properties that are not typical for handmade 3D models. In turn, these prior preservation approaches often fail for VR applications. To overcome these limitations, we have identified key cases that are common to handmade 3D models and must be handled to preserve the surface properties of a mesh.

*3.3.1 Limitations of Prior Surface Property Approaches.* Nearly all 3D models used in VR applications possess surface properties, in addition to their geometric properties. The most common properties are normals, texture coordinates, colors, and materials. The associated texture information has a high impact on the perceptual quality. Hence, to obtain an approximation with high visual fidelity, it is crucial to maintain these surface properties.

In their later work, Garland and Heckbert [14] proposed a generalized error metric that is an extension of the basic quadric error metric. This generalized metric incorporated the surface properties as vertex attributes to provide surface-property-based simplification. They treated each vertex as a vector $v \in R^n$, where the first three components of $v$ are spatial coordinates and the remaining components represent the values of a surface property. Hence, for any vertex with three associated color components, we have $v = [x\ y\ z\ r\ g\ b]$ and $n = 6$. For $n$-dimensional vertices, the modified squared distance $D^2$ of $v$ from any triangle $T$ also has the structure of the original metric: $D^2 = v^T Av + 2b^T v + c$, where

$$\begin{aligned} A &= I - e_1 {e_1}^T - e_2 (e_2)^T \\ b &= (pe_1)\, e_1 + (pe_2)\, e_2 - p \\ c &= pp - (pe_1)^2 - (pe_2)^2 \\ e_1 &= \tfrac{q-p}{\|q-p\|} \\ e_2 &= \tfrac{r-p-(e_1(r-p))e_1}{\|r-p-(e_1(r-p))e_1\|} \end{aligned} \tag{7}$$

In this generalized quadric metric, $(p, q, r)$ are vertices of the triangle $T$, $A$ is the $n \times n$ matrix, and $b$ is an $n \times 1$ vector. One can easily replace color values with texture coordinates to create an $n$-dimensional vector for each vertex. This approach is suitable if the entire model is mapped to a single-texture material such that there is a one-to-one mapping between vertices and texture coordinates. But in practice, multiple-texture materials can be associated with a single 3D model. Furthermore, multiple values from a single texture map may also be assigned to a given vertex. Hence, a model may consist of vertices with multiple sets of texture coordinates. Hoppe [12] also suggested a QEM to handle surface attributes, but both surface property preservation methods assume that every vertex will have at least one surface property defined, in addition to its geometric property. However, that is not the case for many handmade 3D models.

*3.3.2 Multiple Surface Properties Preservation Approach.* To better handle and preserve the surface properties of handmade 3D meshes, we first must identify the common cases that should be expected by our simplification algorithm. We have identified the following common cases involving surface properties:

- A vertex may not have any surface properties.
- Multiple textures may be mapped to the mesh.
- A vertex may be associated with multiple coordinates within the same or different texture maps.

To handle these cases, we have created a new data structure for each vertex that maintains its neighboring faces and corresponding texture coordinates associated with the given vertex. We update these data structures during edge collapse with suboptimal vertex placement for adequate attribute transfer. We have also extended our boundary preservation approach to preserve the edges that form boundaries between multiple textures. The edges that are associated with multiple textures constitute material boundaries. This novel approach consists of

- defaulting texture coordinates and material indices to zero for faces with no surface properties,
- identifying vertices that form material boundaries,
- identifying critical vertices that are associated with multiple materials or textures,
- assigning the user-defined weight $W_t$ to the material boundary vertices and critical vertices, and
- transferring the attributes during each edge collapse.

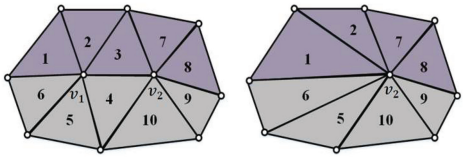



Fig. 5. Attribute transfer while collapsing the edge $(\boldsymbol{v}_1, \boldsymbol{v}_2)$ to vertex $\boldsymbol{v}_2$ where each vertex has multiple attribute values.

If the model has a subset of faces that do not have surface property information, we set corresponding properties such as texture coordinates and material indices to zero. Because the vertices of these faces do not have surface properties, setting the surface property values to zero will make sure that the order of associated edge collapses is based solely on the geometric error and not surface properties that do not exist.

Next, to identify the material boundaries, we utilize a structure that maintains the list of faces in the neighborhood of a vertex. If any two faces in this $v_1$ list have a different material index, we can consider that the given vertex has been assigned two different materials. Hence, it can be identified as a material boundary vertex. Further, if any vertex is associated with multiple texture values, then it is considered a critical vertex.

Next, we apply a user-defined weight factor $W_t$ to the quadric metrics associated with the vertices on the material boundary. This way, we can make sure that the material boundaries remain unaltered and material indices are properly preserved.

Because we have adopted a subset placement policy, handling the texture coordinates during the edge collapse is a trivial task for models with a one-to-one mapping between vertices and texture coordinates. But for models with multiple attributes per vertex, extra care is needed. Consider the edge collapse $(v_1, v_2)$, as shown in Figure 5. Neighboring faces with the same color indicate that a vertex has the same texture coordinate value for all of the faces. Hence, vertex $v_1$ has two texture coordinate values associated with it; one texture coordinate is common for faces 1, 2, and 3, while another texture coordinate is shared by faces 4, 5, and 6. As we use a suboptimal vertex placement, the vertex $v_1$ is moved to vertex $v_2$ during edge collapse. Faces 3 and 4 are removed, while faces 1, 2, 5, and 6 form a set of affected faces. For the affected faces, the vertex $v_1$ is replaced by vertex $v_2$. Because both the vertices $v_1$ and $v_2$ have multiple texture coordinates, when we replace $v_1$ with $v_2$, we must update the texture coordinate associated with the affected faces at vertex $v_1$ with the valid texture coordinate at the vertex $v_2$. For example, when face 5 with the gray color is updated, we should modify its texture coordinate associated with $v_1$ with that of $v_2$ corresponding to gray color. To achieve this, we find the closest match for the texture coordinate of $v_1$ for the affected faces in the set of texture coordinates associated with the vertex $v_2$, with the additional constraint of matching material indices. Hence, for faces 1 and 2, the texture coordinate of $v_1$ is replaced with the texture coordinate of $v_2$ related to purple, but it is replaced with the texture coordinate of $v_2$ represented by the gray color for faces 5 and 6. This strategy allows achieving an adequate attribute transfer during edge collapses, as illustrated in Figure 5.

### 3.4 Summary of QEM$_{4VR}$ Approach

To summarize, the QEM$_{4VR}$ algorithm employs the following steps to reduce any 3D model (manifold or nonmanifold) into a low-poly mesh with approximately $N$ (which is user defined) faces that approximate the surface properties of the original mesh.

**Step 1:** Remove any redundant vertices and faces that were created during the construction of the model.

**Step 2:** For each vertex $v$, create the data structures $T_v$ and $T_{x_v}$ representing the neighboring faces of vertex $v$ and its texture coordinates, respectively.

**Step 3:** For each vertex, determine if it is a complex vertex lying on the mesh boundary or not. If it lies on the boundary, determine the boundary curvature at the vertex using Equation (5). If it is a nonmanifold vertex on the boundary, mark it as a complex vertex. Also, determine if the vertex is on a material boundary or not utilizing its data structures $T_v$ and $T_{x_v}$.

**Step 4:** Compute the quadric metric for each vertex. For vertices on the mesh boundary, weigh the quadric metric by its corresponding curvature value and the user-defined weight factor $W_b$. Further, if the vertex lies on a material boundary, weigh the corresponding quadric with a user-defined weight factor $W_t$. Note that edges involving nonmanifold boundary vertices (marked as complex vertex) are not collapsed.

**Step 5:** For each edge, compute a quadric cost that is the sum of the cost associated with its two endpoints.

**Step 6:** Select the edge with the minimum quadric error. To collapse the selected edge with vertices $v_1$ and $v_2$, we select the subset placement policy for determining the target position. If $v_2$ is selected as a target position, we merge the set of neighboring faces of $v_1$ with that of $v_2$ and update the attributes for the merged faces using the described attribute transfer strategy. Next, we update the quadric error associated with $v_2$ and the list of edges.

**Step 7:** Repeat steps 2 to 6 until the poly count is less than $\boldsymbol{N}$.

### 3.5 Typical Workflow

In typical VR applications, the virtual scene may consist of a large number of 3D models. Hence, it is important to understand the workflow of the proposed $\text{QEM}_{\text{4VR}}$ system and how it can be incorporated into the workflow of developing a typical VR application. The description of the workflow of the proposed $\text{QEM}_{\text{4VR}}$ will be helpful in planning and resource allocation while designing the virtual scene.

$\text{QEM}_{\text{4VR}}$ is used to preprocess the high-poly model to generate lower-poly versions of various acceptable poly counts. $\text{QEM}_{\text{4VR}}$ consists of four major phases:

- Quadric computation per vertex
- Cost computation per edge
- Building the edge-collapse list based on quadric cost
- Performing the edge-collapse operation to generate the low-poly model.

Different low-poly models generated by $\text{QEM}_{\text{4VR}}$ form a level-of-detail set of models for a given high-poly 3D model. These level-of-detail models can be added to the virtual scene to enable view-based level-of-detail rendering. The selection of a certain level of detail is done in a real-time manner based on the current position of the virtual camera and the corresponding orientation.

## 4 EXPERIMENTS

We have performed five sets of experiments to demonstrate the effectiveness and quality of $\text{QEM}_{\text{4VR}}$:

1. Compare geometric approximation errors when high-poly models are drastically simplified.
2. Compare texture errors when high-poly models are drastically simplified.
3. Compare approximation errors induced during progressive simplification.
4. Compare how multiple simplified models impact frame rate for a VR application.
5. Compare perceptual quality measurements of the simplified meshes.

Table 1. Attributes of Dataset Used for Experiments

| | Mesh Type | Faces | Vertices | Boundary Edges |
|---|---|---|---|---|
| **Nontextured Models** | | | | |
| RockerArm | Manifold | 80,354 | 40,177 | 0 |
| Car | Manifold | 268,630 | 202,890 | 120,227 |
| Dragon | Nonmanifold | 37,986 | 22,126 | 6,214 |
| **Textured Models** | | | | |
| Head | Manifold | 17,684 | 8,844 | 0 |
| Jet | Manifold | 35,095 | 18,849 | 2,341 |
| Airplane | Nonmanifold | 66,496 | 36,937 | 6,686 |

For the nontextured models, we compared our $QEM_{4VR}$ algorithm to the original QEM algorithm ($QEM_{ORIG}$), which does not preserve boundaries, and the QEM algorithm with boundary constraints ($QEM_{BP}$). For the textured models, we compared our $QEM_{4VR}$ algorithm to a texture preservation variation ($QEM_{TP}$) and a boundary-and-texture preservation variation ($QEM_{BTP}$), in addition to $QEM_{ORIG}$ and $QEM_{BP}$.

**Setup.** The $QEM_{4VR}$ algorithm was implemented in C++, and the experiments were conducted on a computer with Intel® Core™ i7-5820K with 3.30GHz speed and 32GB internal RAM. For other methods that are considered for comparison, we used their MeshLab [50] implementations. We used OpenGL for rendering the resultant low-poly meshes.

**Dataset.** We evaluated the performance of our new $QEM_{4VR}$ algorithm with a dataset consisting of a variety of 3D models. The dataset included manifold meshes without boundaries, manifold meshes with boundaries, and nonmanifold meshes, all with and without textures. See Table 1 for a summary.

**Parameters.** User-defined weight factors $W_b$ and $W_t$ help to prioritize surface and material boundaries. Along with boundary curvature, the above-mentioned weight factors have a significant impact on the order of edge collapse. Hence, these weights must be selected carefully. We experimented with various values of $W_b$ and $W_t$ ranging from 1 to 1,000. For brevity, we have not included results with varying values of $W_b$ and $W_t$. However, we found that $W_b$ being 5 and $W_t$ being 1,000 are most suitable for all the models in the experimental dataset. For other variations of QEM, we set the quality threshold to the default 0.3 in MeshLab. For the boundary preservation variations, we set the boundary weight to 1 for all models, because lowering the boundary-preserving weight does not have a drastic effect on the final output.

### 4.1 Geometric Approximation Errors for Drastic Simplification

In the first experiment, we simplified all the models to 10% of their original size using $QEM_{4VR}$ and the other variations of QEM. To quantify how well the low-poly meshes generated by the various QEM approaches approximate the original meshes, we used an approach similar to prior researchers [12, 13]. We computed the distance between two meshes by sampling a set of points from the original high-poly mesh and then measuring their distance to their closest point on each low-poly mesh. This provides a one-sided approximation error. The average computed errors, called the mean approximation error or Metro [51], is normalized with respect to the diagonal length of the original model's bounding box and is used as the measurement for error.

Tables 2 and 3 provide the quantitative results by the different methods for errors introduced during simplification of the nontextured and textured models, respectively. It can be clearly seen that our new $QEM_{4VR}$ algorithm results in the least amount of error and well approximates the

Table 2. Mean Approximation Errors (Lower Is Better) for Simplifying Nontextured Models to Approximately 10% of Original Size

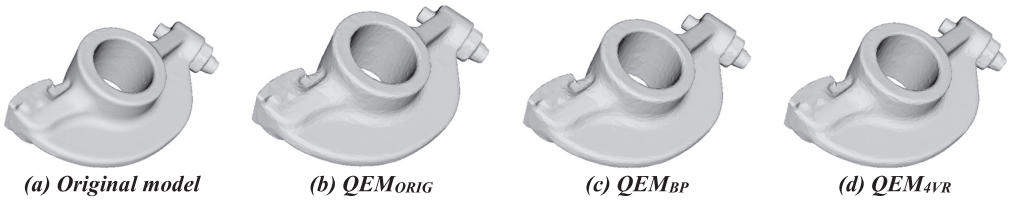
| Non-textured Model | Number of Faces | | | | Metro ($10^{-3}$) | | |
|---|---|---|---|---|---|---|---|
| | Original | $QEM_{ORIG}$ | $QEM_{BP}$ | $QEM_{4VR}$ | $QEM_{ORIG}$ | $QEM_{BP}$ | $QEM_{4VR}$ |
| RockerArm | 80,354 | 8,174 | 8,174 | 8,174 | 0.181 | 0.181 | **0.180** |
| Car | 268,630 | 24,176 | 104,461* | 24,258 | 1.245 | 1.018 | **0.473** |
| Dragon | 37,986 | 3,566 | 6,156* | 3,567 | 1.397 | 1.657 | **0.341** |

*This is the lowest bound for $QEM_{BP}$ due to its inability to reduce boundary edges.

Table 3. Mean Approximation Errors (Lower Is Better) for Simplifying Textured Models to Approximately 10% of Original Size

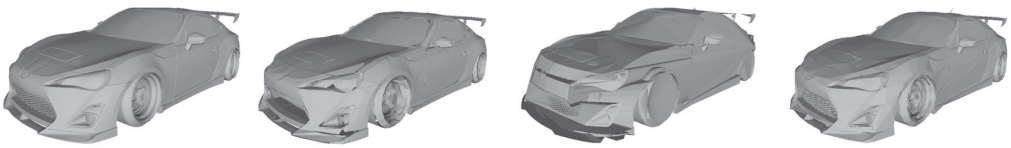
| Textured Model | Number of Faces | | | | | | Metro ($10^{-3}$) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Original | $QEM_{ORIG}$ | $QEM_{BP}$ | $QEM_{TP}$ | $QEM_{BTP}$ | $QEM_{4VR}$ | $QEM_{ORIG}$ | $QEM_{BP}$ | $QEM_{TP}$ | $QEM_{BTP}$ | $QEM_{4VR}$ |
| Head | 17,684 | 1,844 | 1,844 | 1,844 | 1,844 | 1,844 | 2.553 | 2.553 | 1.391 | 1.391 | **0.911** |
| Jet | 35,095 | 3,104 | 3,455 | 3,504 | 3,505 | 3,505 | 1.071 | 3.267 | 1.318 | 2.316 | **0.340** |
| Airplane | 66,496 | 6,796 | 6,796 | FAIL* | FAIL* | 6,797 | 0.941 | 3.415 | FAIL* | FAIL* | **0.237** |

*$QEM_{TP}$ and $QEM_{BTP}$ fail in these cases due to vertices not having any texture coordinates.



*(a) Original model* *(b) $QEM_{ORIG}$* *(c) $QEM_{BP}$* *(d) $QEM_{4VR}$*

Fig. 6. Simplification of RockerArm (manifold with no boundaries) to approximately 10% of its original number of faces.



*(a) Original model* *(b) $QEM_{ORIG}$* *(c) $QEM_{BP}$* *(d) $QEM_{4VR}$*

Fig. 7. Simplification of Car (manifold with boundaries) to approximately 10% of its original number of faces.

original models. Figures 6 to 8 and 9 to 11 show the simplified nontextured and textured models, respectively.

### 4.2 Texture Errors for Drastic Simplification

To compare the quality of our $QEM_{4VR}$ algorithm to the other QEM variations, we computed a texture error between the original model and each low-poly mesh as described in [12]. The texture error is estimated by measuring the deviation of the texture coordinates of sampled points from values linearly interpolated at their projection on the closest face of the low-poly model.

Various results, as shown in Figures 9 to 11, demonstrate the ability of our $QEM_{4VR}$ algorithm to preserve surface properties, including multiple textures. $QEM_{4VR}$ efficiently handles the attribute transfer during simplification such that the resultant low-poly mesh well approximates the original model without introducing any visible degradation in the texture mapping. Table 4 provides the

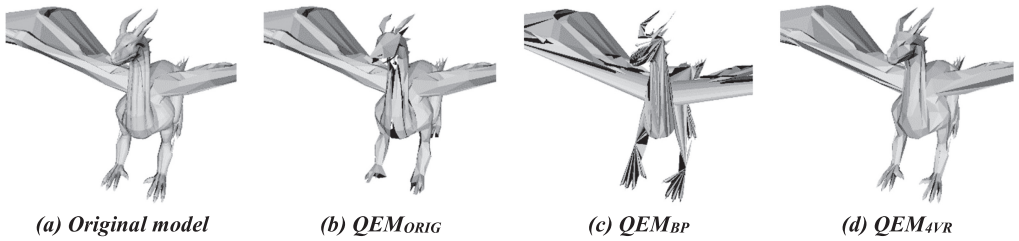



Fig. 8. Simplification of Dragon (nonmanifold) to approximately 10% of its original size.

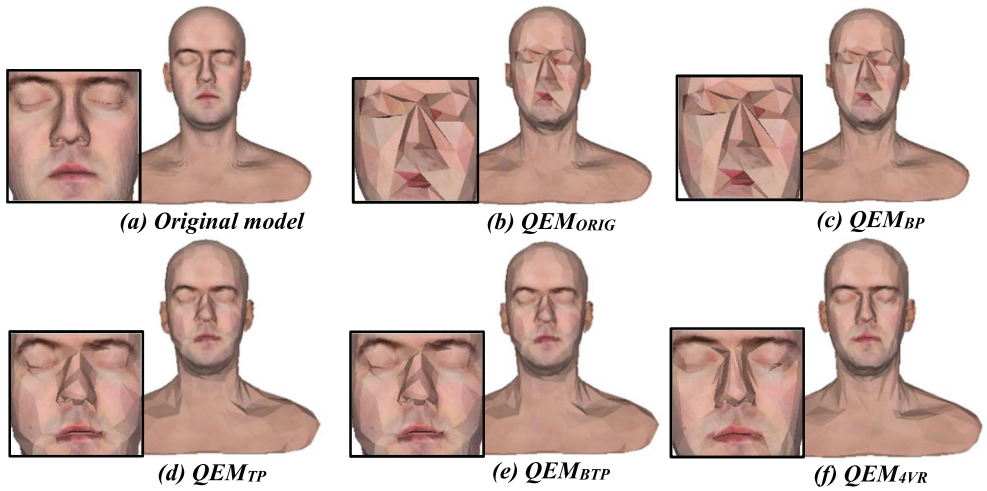



Fig. 9. Simplification of Head (textured manifold with no boundaries) to approximately 10% of its original size with close-up inspection of texture qualities.

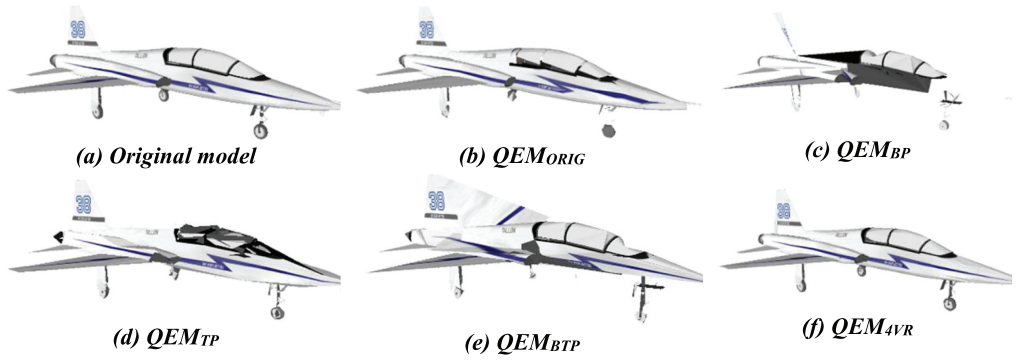



Fig. 10. Simplification of Jet (textured manifold with boundaries) down to approximately 10% of its original size.

quantitative texture error measures for the different approaches. Our $QEM_{4VR}$ method induces less texture error than the other QEM variations. Also, it is important to point out that both $QEM_{TP}$ and $QEM_{BTP}$ failed on the nonmanifold Airplane mesh due to it containing vertices without associated texture coordinates.

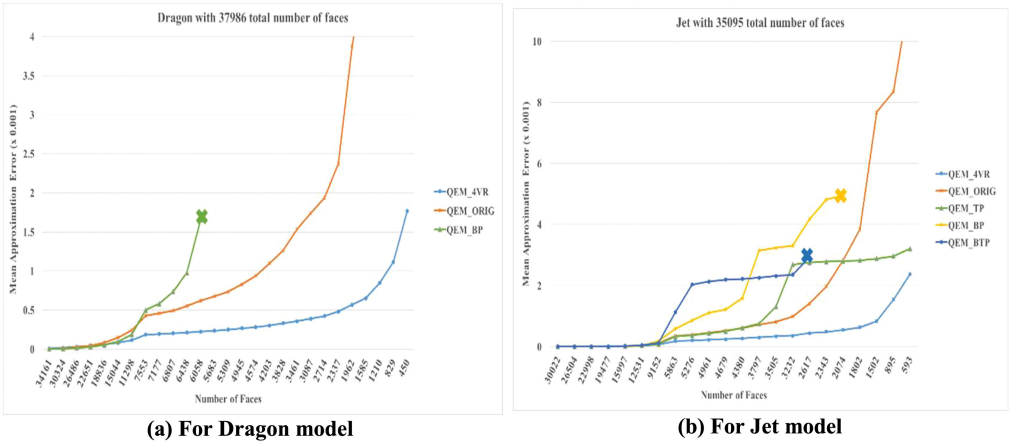
*(a) Original model* *(b) $QEM_{ORIG}$* *(c) $QEM_{BP}$* *(d) $QEM_{4VR}$*

Fig. 11. Simplification of Airplane (textured nonmanifold with boundaries) down to approximately 10% of its original size. Note that $QEM_{TP}$ and $QEM_{BTP}$ failed to produce results due to their inability to handle vertices with multiple texture coordinates.

Table 4. Texture Errors (Lower Is Better) for Simplifying Textured Models to Approximately 10%

| Textured Model | Number of Faces | | | | | | Texture Error ($10^{-2}$) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Original | $QEM_{ORIG}$ | $QEM_{BP}$ | $QEM_{TP}$ | $QEM_{BTP}$ | $QEM_{4VR}$ | $QEM_{ORIG}$ | $QEM_{BP}$ | $QEM_{TP}$ | $QEM_{BTP}$ | $QEM_{4VR}$ |
| Head | 17,684 | 1,844 | 1,844 | 1,844 | 1,844 | 1,844 | 1.02 | 1.02 | 0.87 | 0.87 | **0.34** |
| Jet | 35,095 | 3,104 | 3,455 | 3,504 | 3,505 | 3,505 | 3.03 | 4.89 | 3.07 | 3.45 | **2.01** |
| Airplane | 66,496 | 6,796 | 6,796 | FAIL* | FAIL* | 6,797 | 7.19 | 8.13 | FAIL* | FAIL* | **2.26** |

*$QEM_{TP}$ and $QEM_{BTP}$ fail in these cases due to vertices not having any texture coordinates.




**(a) For Dragon model** **(b) For Jet model**

Fig. 12. Mean approximation error (Metro) with respect to the number of faces for progressive simplification. Termination of error curve indicates that no further poly count reduction can be achieved with the respective mesh simplification algorithm.

### 4.3 Progressive Simplification

To further evaluate the efficacy of the proposed method to generate the high-fidelity sparse models, we obtained various low-poly models with different levels of sparseness (i.e., with a different number of faces) for the Dragon and Jet models. For these low-poly models, we computed the mean approximation error or Metro. Figure 12 illustrates the behavior of Metro while reducing the number of faces for the Dragon and Jet models. It can be seen that $QEM_{4VR}$ introduces minimal error and consequently maintains high fidelity even at a very high level of sparseness. Further, $QEM_{4VR}$ causes minimum approximation error across different levels of sparseness compared to the other QEM variations. It is also important to note that for both $QEM_{BP}$ and $QEM_{BTP}$, the error curve terminates midway, indicating that no further poly count reduction can be achieved due to the lower bound on simplification enforced by corresponding boundary constraints.

Table 5. Frame Rate Analysis

| Number of Models | Original | | $QEM_{ORIG}$ | | $QEM_{BP}$ | | $QEM_{4VR}$ | |
|---|---|---|---|---|---|---|---|---|
| | Polygon Count | Frame Rate | Polygon Count | Frame Rate | Polygon Count | Frame Rate | Polygon Count | Frame Rate |
| 1 | 66,496 | 57.61 | 6,796 | 57.68 | 6,796 | 57.76 | 6,797 | 57.78 |
| 2 | 146,850 | 57.39 | 14,970 | 57.62 | 14,970 | 57.62 | 14,971 | 57.65 |
| 3 | 181,945 | 52.18 | 18,074 | 57.59 | 18,425 | 57.35 | 18,476 | 57.65 |
| 4 | 450,575 | 37.71 | 42,250 | 57.50 | 122,886 | 50.26 | 42,734 | 57.63 |
| 5 | 468,259 | 37.36 | 44,094 | 57.49 | 124,730 | 50.25 | 44,578 | 57.55 |
| 6 | 506,245 | 35.33 | 47,660 | 57.49 | 130,886 | 47.92 | 48,145 | 57.49 |

### 4.4 Frame Rate Analysis

To analyze the frame rate, we designed an experimental setup where four different scenes were generated using the Unity3D game engine. The first scene was created utilizing the original six dense models from our dataset. Those models were added to the scene and rendered in a sequential manner. We recorded the average frame per second (fps) every time a new model was added to the scene. Similarly, the other three scenes were created by using sparse models generated by $QEM_{ORIG}$, $QEM_{BP}$, and $QEM_{4VR}$. Experiments were performed on Samsung Gear VR with the Samsung S6 edge as a mobile device. Table 5 shows the frame rate performance while rendering the scenes. As the polygon count increases, the frame rate drops significantly in the case of the scene generated using dense models and sparse models obtained by $QEM_{BP}$. On the contrary, the scenes generated using sparse models obtained by $QEM_{ORIG}$ and $QEM_{4VR}$ maintain the average frame rate of 58fps.

### 4.5 Perceptual Quality Assessment

In VR applications, QoE is a crucial factor [41]. Hence, it becomes essential to quantitatively measure the visual quality of our simplified meshes in correlation with the human perception. Various perceptual quality measures have been investigated. A mesh structural distortion measure (MSDM) proposed by Lavoué et al. [52] is one of the earlier works on objective perceptual mesh quality. Lavoué et al. [52] extended the well-known structural similarity index for 2D images [53] to 3D triangular meshes. MSDM utilizes the changes in the local statistics (i.e., mean, variance, and covariance) of a surface curvature to quantify a visual degradation in the mesh quality. Later, Lavoué et al. [54] proposed an improved version (MSDM2) that performs a vertex-matching preprocessing step to allow the comparison of two meshes with different topologies.

Recently, Wang et al. [55] proposed the fast mesh perceptual distance (FMPD) metric. FMPD measures the local surface roughness derived from the Gaussian curvature of the mesh surface. It estimates the perceptual distance as the deviation of a global roughness of the modified mesh from a global roughness of the reference mesh. Hence, it does not require a mesh registration preprocessing step. Consequently, it can be applied to compare two meshes with different connectivity. Furthermore, FMPD does not need the full information about the original reference mesh, and can be considered as a reduced-reference metric.

Torkhani et al. [56] proposed a curvature tensor-based perceptual distance measure (TPDM) that utilizes tensor eigenvalues and eigenvectors to derive a perceptually oriented distance metric. TPDM incorporates roughness-based weighting of the local tensor distance in order to account for the visual masking effect of the human visual system. TPDM also performs a vertex matching as a preprocessing similar to MSDM2.

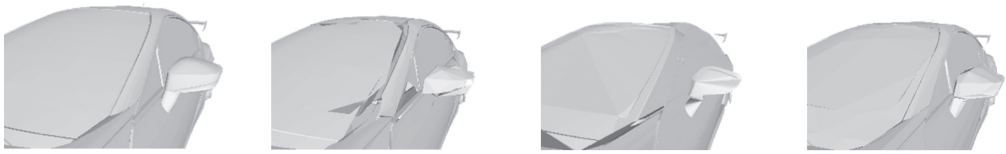
Table 6. FMPD Metric (Lower Is Better) for Simplifying Nontextured Models to Roughly 10%

| Nontextured Model | FMPD | | |
|---|---|---|---|
| | $QEM_{ORIG}$ | $QEM_{BP}$ | $QEM_{4VR}$ |
| RockerArm | **0.408** | **0.408** | 0.413 |
| Car | **1.039** | 1.921 | 1.744 |
| Dragon | **0.185** | 0.909 | 0.220 |

Table 7. FMPD Metric (Lower Is Better) for Simplifying Textured Models to Roughly 10%

| Textured Model | FMPD | | | | |
|---|---|---|---|---|---|
| | $QEM_{ORIG}$ | $QEM_{BP}$ | $QEM_{TP}$ | $QEM_{BTP}$ | $QEM_{4VR}$ |
| Head | **0.312** | **0.312** | 0.516 | 0.516 | 0.523 |
| Jet | 1.470 | 2.946 | 0.122 | **0.096** | 1.999 |
| Airplane | **0.338** | 0.990 | FAIL* | FAIL* | 1.423 |

*$QEM_{TP}$ and $QEM_{BTP}$ fail in these cases due to vertices not having any texture coordinates.



*(a) Original model* | *(b) $QEM_{ORIG}$* **(FMPD = 1.039)** | *(c) $QEM_{BP}$* **(FMPD = 1.921)** | *(d) $QEM_{4VR}$* **(FMPD = 1.744)**

Fig. 13. Simplification of Car (manifold with boundaries) to approximately 10% of its original number of faces, along with corresponding FMPD measures.

Among the available perceptual quality metrics, MSDM requires that the meshes to be compared share the same connectivity. Hence, it cannot be applied in simplification evaluation [54]. Although MSDM2 and TPDM perform vertex matching as a preprocessing step to avoid the same connectivity constraint, these methods do not handle nonmanifold meshes. Hence, we utilized FMPD as a quantitative metric for assessing the perceptual quality of our simplified meshes.

**Experimental setup:** To perform a perceptual quality assessment of $QEM_{4VR}$, we utilized the open-source implementation of FMPD available at [57]. We simplified all the models in Table 1 to 10% of their original size using the proposed $QEM_{4VR}$ and other variations of QEM. For each low-poly mesh, we computed its FMPD (see Tables 6 and 7). Note, FMPD values are not clipped to the range (0, 1).

For RockerArm (manifold model), the reduced meshes obtained with $QEM_{4VR}$ and other QEM variants are perceptually similar (see Figure 6), and consequently, corresponding FMPD measures are also similar (see Table 6). However, in the case of the Car model (manifold with boundaries), even though the reduced mesh obtained using $QEM_{ORIG}$ has a lot of gaps near the windshield and side mirrors (see Figure 13), the corresponding FMPD metric is 1.039 (lower than $QEM_{4VR}$). Similarly, in case of the Dragon model (nonmanifold), the distorted and incomplete reduced mesh obtained with $QEM_{ORIG}$ has a lower FMPD metric (See Figure 14).

Furthermore, when the FMPD metric is applied to textured models, similar results are observed. For the textured Head model, the FMPD measure for $QEM_{4VR}$ is higher than for other QEM

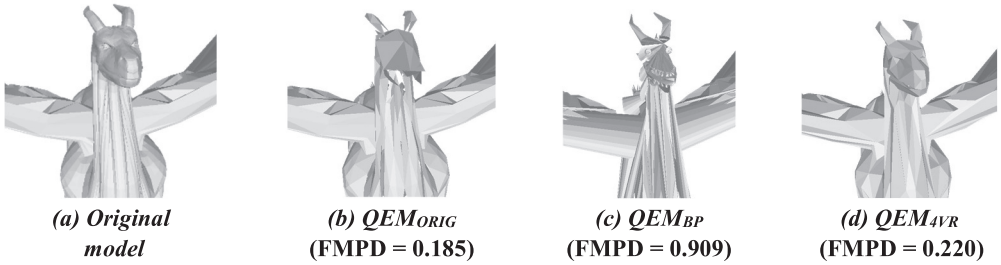



Fig. 14. Simplification of Dragon (nonmanifold) to approximately 10% of its original number of faces, along with corresponding FMPD measures.

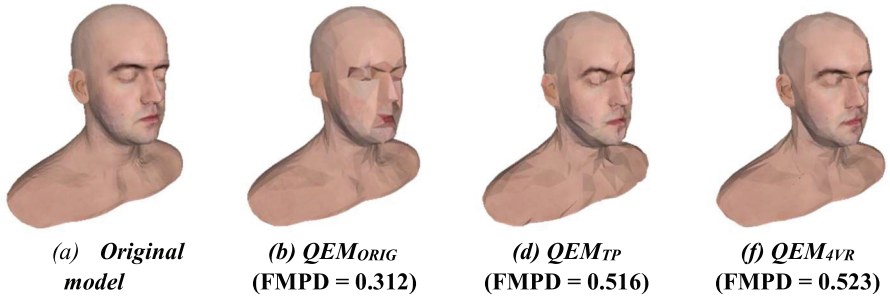



Fig. 15. Simplification of Head (textured manifold with no boundaries) to approximately 10% of its original number of faces, along with corresponding FMPD measures. As $QEM_{BP}$ and $QEM_{BTP}$ results are similar to $QEM_{ORIG}$ and $QEM_{TP}$, respectively, they are excluded for succinctness.

variants even when the reduced mesh obtained using $QEM_{4VR}$ has a better perceptual quality (see Figure 15—various sharp features such as the nose and ears are well maintained using $QEM_{4VR}$). Moreover, the FMPD measures for the low-poly models obtained using $QEM_{ORIG}$ and $QEM_{BTP}$ are lower than $QEM_{4VR}$ despite the presence of a large number of gaps at boundaries and the corresponding visual quality not being satisfactory (see Figure 16).

**Progressive Simplification:** To further evaluate the effectiveness of the FMPD, we obtained various low-poly models with different levels of sparseness (i.e., with a different number of faces) for the Dragon and Jet models. For these low-poly models, we computed the FMPD. Figure 17 illustrates the behavior of the FMPD while reducing the number of faces for the Dragon and Jet models. Logically, as the number of faces reduces, the perceptual distance from the original model should monotonically increase. However, FMPD has a contrasting behavior in different cases. As shown in Figure 17, the FMPD curve for various mesh simplification methods has anomalous characteristics with both nontextured and textured models.

To summarize, because the FMPD metric measures the difference in the global roughness as a perceptual distance, it does not take boundary gaps into account. Due to the inability of FMPD to incorporate boundary gaps and the completeness of the model while determining perceptual quality, it does not accurately represent the subjective visual quality. Therefore, for the low-poly mesh generated using $QEM_{ORIG}$, despite the presence of large boundary gaps, the FMPD metric is lower than compared to $QEM_{4VR}$. Although existing perceptual quality metrics can accurately measure degradation due to noise addition (e.g., watermarking), these methods are unable to accurately measure the subjective perceptual quality of simplified meshes.

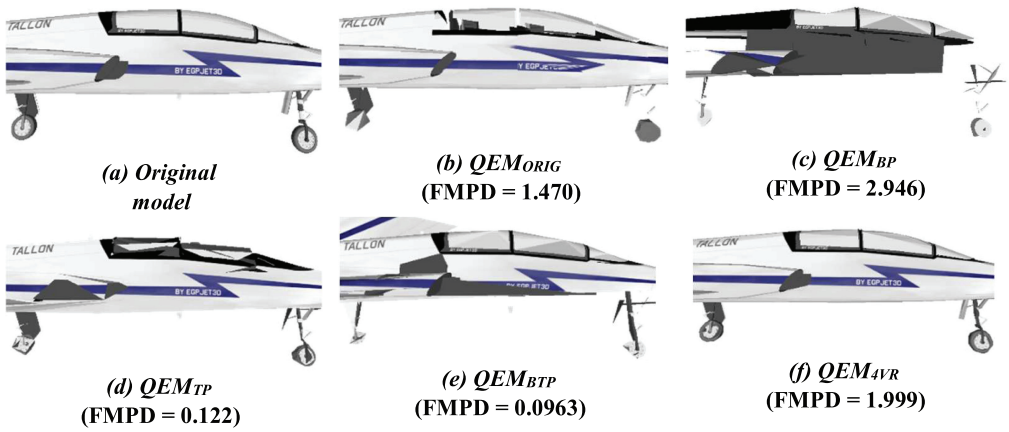
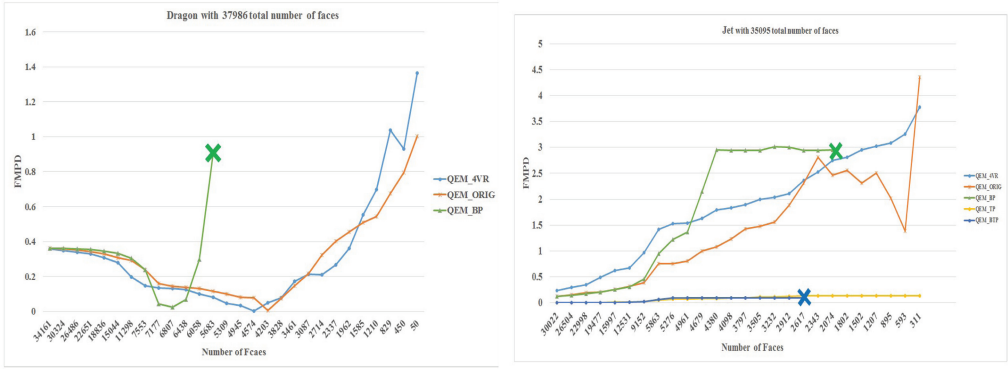



Fig. 16. Simplification of Jet down to approximately 10% of its original number of faces, along with corresponding FMPD measures.




(a) For Dragon model (b) For Jet model

Fig. 17. FMPD with respect to the number of faces in simplified models. Termination of error curve indicates that no further poly count reduction can be achieved with the respective method.

## 5 FUTURE JND STUDIES EVALUATING SUBJECTIVE PERCEPTUAL QUALITY

In Section 4, we demonstrated that the objectively computed FMPD metric does not accurately reflect the subjective perceptual quality of our simplified meshes. This is due to the fact that the FMPD metric is based on the deviation in global surface roughness while subjective perceptual quality is dependent on the Human Visual System (HVS) [44]. Most objective perceptual quality metrics assume that visual quality increases as the number of vertices increases, but user studies have shown that not every set of vertices has a significant impact on visual quality [58]. More recently, Wang et al. [59] clearly demonstrated that traditional quality measures, which are linear in nature, do not account for the nonlinear aspects of human perception. Hence, in order to truly evaluate the perceptual quality of the $QEM_{4VR}$ algorithm, we must rely on the HVS and conduct user studies on subjective perceptual quality.

As discussed in Section 2.3, a JND methodology can be used to evaluate the visual qualities of multimedia (e.g., [44–48]). However, there are three JND methodologies commonly used in the field of psychophysics: (1) the method of constant stimuli, (2) the method of limits, and (3) the

method of adjustment [60]. With the method of constant stimuli, a predetermined series of repeating stimuli is shown to the user, and the user is asked to respond if a difference is noticed [61]. While highly accurate due to keeping subjects from anticipating changes in the stimulus, the method is considered inefficient [60, 61]. With the method of limits, a stimulus is continuously increased or decreased until it is perceivable by the user [60]. A common implementation of the method of limits is the staircase or up-down method, in which the stimulus is increased when it is not perceived and decreased when it is perceived [61]. Finally, with the method of adjustment, the user can freely adjust the difference until it is not perceived [60].

Recent research by Qin et al. [60] has demonstrated that a one-up-two-down staircase method yields smaller JNDs than an adapted method of adjustment. Hence, we plan to employ the same method for evaluating the subjective perceptual quality of our $QEM_{4VR}$ algorithm. In our future studies, participants will not be asked whether they see a difference between two meshes, but instead will be asked to indicate which mesh is visually higher quality. For each side-by-side comparison, both meshes will be simultaneously rotated one full cycle in a view-independent manner to allow all silhouettes to be examined, similar to the JND study conducted by Cheng and Boulanger [62] on the effects of viewing distance on level of detail.

For each comparison, if the participant selects the incorrect mesh (i.e., the lower-poly mesh), the simplification difference (i.e., difference in polygon counts) will be increased by 5% of the original model's polygon count. This is the "one up" aspect of the staircase method. When the participant selects the correct mesh (i.e., the higher-poly mesh), both meshes will be redisplayed in a random order. If the participant again correctly selects the higher-poly mesh, the simplification difference will be decreased by 5% for the next pair of meshes. This is the "two down" aspect.

We have planned for conducting two JND studies to evaluate the subjective perceptual quality of our $QEM_{4VR}$ algorithm. The first JND study will focus on nontextured models and will have participants complete the one-up-two-down staircase method for $QEM_{ORIG}$, $QEM_{BP}$, and $QEM_{4VR}$. The second JND study will focus on textured models and will have participants complete the method for $QEM_{TP}$, $QEM_{BTP}$, and $QEM_{4VR}$. For both studies, we have prepared simplified meshes for six models (two manifolds with no boundaries, two manifolds with boundaries, and two nonmanifolds). We hypothesize that the results of these studies will demonstrate that $QEM_{4VR}$ yields a larger JND than the other algorithms for both nontextured and textured models, which would indicate that $QEM_{4VR}$ affords better subjective perceptual quality. We are currently awaiting Institutional Review Board (IRB) approval to begin conducting our JND studies.

## 6 CONCLUSION

We have described a new high-fidelity mesh simplification method, $QEM_{4VR}$, tailored for VR applications to yield low-poly meshes for any 3D model (both manifolds and nonmanifolds). The $QEM_{4VR}$ method has two significant advantages: (1) its resulting meshes have less geometric error than previous variations of the QEM simplification algorithm due to its curvature-based boundary preservation approach, and (2) it is more capable of preserving surface properties, such as multiple texture coordinates for a single vertex. We validated both key contributions through experiments that compared our method to prior QEM variations.

To help VR developers create high-fidelity, low-poly virtual environments, we have made the implementation of our $QEM_{4VR}$ algorithm, and the 3D models used in our experiments, available at [63].

Based on the perceptual quality assessment performed in this work, we have identified the following directions for future work: (1) Our perceptual quality assessment experiment shows that state-of-the-art perceptual quality metrics are incapable of adequately measuring the visual quality of simplified nonmanifold meshes. It motivates the need to further explore and design a quality

metric correlated with the human perception to evaluate mesh simplification methods. An adequate perceptual measure should incorporate boundary gaps, completeness of the model, and associated texture information while computing the error. (2) We plan to perform future studies involving human responses based on the concept of Just Noticeable Differences (JNDs) to assess and compare the subjective perceptual quality of our new $QEM_{4VR}$ algorithm to prior QEM algorithms, for both nontextured and textured meshes.

## REFERENCES


[1] K. Boos, D. Chu, and E. Cuervo. 2016. Flashback: Immersive virtual reality on mobile devices via rendering memoization. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 291–304.

[2] D. Pohl and C. F. de Tejada Quemada. 2016. See what I see: Concepts to improve the social acceptance of HMDs. In *2016 IEEE Virtual Reality (VR'16)*. IEEE, 267–268.

[3] A. Gargantini, F. Terzi, M. Zambelli, and S. Bonfanti. 2015. A low-cost virtual reality game for amblyopia rehabilitation. In *Proceedings of the 3rd 2015 Workshop on ICTs for Improving Patients Rehabilitation Research Techniques*. ACM, 81–84.

[4] A. Steed, S. Frlston, M. M. Lopez, J. Drummond, Y. Pan, and D. Swapp. 2016. An'in the wild' experiment on presence and embodiment using consumer virtual reality equipment. *IEEE Transactions on Visualization and Computer Graphics* 22, 4, 1406–1414.

[5] X. Tong, D. Gromala, A. Amin, and A. Choo. 2015. The design of an immersive mobile virtual reality serious game in cardboard head-mounted display for pain management. In *International Symposium on Pervasive Computing Paradigms for Mental Health*. Springer International Publishing, 284–293.

[6] C. Lai, R. P. McMahan, M. Kitagawa, and I. Connolly. 2016. Geometry explorer: Facilitating geometry education with virtual reality. In *International Conference on Virtual, Augmented and Mixed Reality*. Springer International Publishing, 702–713.

[7] D. J. Zielinski, H. M. Rao, N. D. Potter, M. A. Sommer, L. G. Appelbaum, and R. Kopper. 2016. Evaluating the effects of image persistence on dynamic target acquisition in low frame rate virtual environments. In *2016 IEEE Symposium on 3D User Interfaces (3DUI'16)*. IEEE, 133–140.

[8] C. Pruett. Squeezing performance out of your unity gear VR game. Retrieved from https://developer.oculus.com/blog/squeezing-performance-out-of-yourunity-gear-vr-game.

[9] Microsoft Developers Resources. Performance recommendations for Microsoft Hololens. Retrieved from https://developer.microsoft.com/en-us/windows/holographic/performance-recommendations.

[10] D. P. Luebke. 2001. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications* 21, 3, 24–35.

[11] A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. 1998. Converting sets of polygons to manifold surfaces by cutting and stitching. In *Proceedings of Visualization*. IEEE, 383–390.

[12] H. Hoppe. 1999. New quadric metric for simplifying meshes with appearance attributes. In *Proceedings of the Conference on Visualization: Celebrating Ten Years*. IEEE Computer Society Press, 59–66.

[13] M. Garland and P. S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., 209–216.

[14] M. Garland and P. S. Heckbert. 1998. Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings of Visualization*. IEEE, 263–269.

[15] K. Bahirat, C. Lai, R. P. McMahan, and B. Prabhakaran. 2017. A boundary and texture preserving mesh simplification algorithm for virtual reality. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 50–61.

[16] P. Cignoni, C. Montani, and R. Scopigno. 1998. A comparison of mesh simplification algorithms. *Computers & Graphics* 22, 1, 37–54.

[17] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. 1993. Mesh optimization. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, 19–26.

[18] H. Hoppe. 1996. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. ACM, 99–108.

[19] G. Turk. 1992. Re-tiling polygonal surfaces. *ACM SIGGRAPH Computer Graphics* 26, 2, 55–64.

[20] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang. 1995. Voxel based object simplification. In *Proceedings of the 6th Conference on Visualization*. IEEE Computer Society, 296.

[21] W. E. Lorensen and H. E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Siggraph Computer Graphics* 21, 4, 163–169.

[22] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, and W. Wright. 1996. Simplification envelopes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. ACM, 119–128.
[23] M. Lounsbery, T. D. DeRose, and J. Warren. 1997. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics (TOG)* 16, 1, 34–73.
[24] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. 1995. Multiresolution analysis of arbitrary meshes. In *Proceedings of the 22nd Annual Conference on Computer GRAPHICS and Interactive Techniques*. ACM, 173–182.
[25] M. Hosseini, D. T. Ahmed, and S. Shirmohammadi. 2012. Adaptive 3D texture streaming in M3G-based mobile games. In *Proceedings of the 3rd Multimedia Systems Conference*. ACM, 143–148.
[26] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. 1992. Decimation of triangle meshes. *ACM Siggraph Computer Graphics* 26, 2, 65–70.
[27] J. Rossignac and P. Borrel. 1993. Multi-resolution 3D approximations for rendering complex scenes. *Modeling in Computer Graphics: Methods and Applications*. Springer, Berlin, Heidelberg, 455–465.
[28] D. Luebke and C. Erikson. 1997. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., 199–208.
[29] P. Lindstrom and G. Turk. 1998. Fast and memory efficient polygonal simplification. In *Proceedings of Visualization*. IEEE, 279–286.
[30] M. Garland. 1999. Quadric-based polygonal surface simplification (No. CMU-CS-99-105). *Carnegie-Mellon University of Pittsburgh*, School of Computer Science.
[31] J. H. Wu, S. M. Hu, C. L. Tai, and J. G. Sun. 2001. An effective feature-preserving mesh simplification scheme based on face constriction. In *Proceedings of the 9th Pacific Conference on Computer Graphics and Applications, 2001*. IEEE, 12–21.
[32] E. Ovreiu. 2012. Accurate 3D mesh simplification. Doctoral Dissertation, INSA de Lyon.
[33] E. Pojar and D. Schmalstieg. 2003. User-controlled creation of multiresolution meshes. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics*. ACM, 127–130.
[34] T. Ebrahimi. 2009. Quality of multimedia experience: Past, present and future. In *Proceedings of the 17th ACM International Conference on Multimedia* (No. MMSPL-CONF-2010-003) (*MM'09)*. ACM, 3–4.
[35] C. Timmerer, T. Ebrahimi, and F. Pereira. 2015. Toward a new assessment of quality. *Computer* 48, 3, 108–110.
[36] A. Raake and S. Egger. 2014. Quality and quality of experience. In *Quality of Experience*. Springer, Cham, 11–33.
[37] J. Puig, A. Perkis, F. Lindseth, and T. Ebrahimi. 2012. Towards an efficient methodology for evaluation of quality of experience in Augmented Reality. In *2012 4th International Workshop on Quality of Multimedia Experience (QoMEX'12)*. IEEE, 188–193.
[38] R. P. McMahan, D. A. Bowman, D. J. Zielinski, and R. B. Brady. 2012. Evaluating display fidelity and interaction fidelity in a virtual reality game. *IEEE Transactions on Visualization and Computer Graphics* 18, 4, 626–633.
[39] D. A. Bowman and R. P. McMahan. 2007. Virtual reality: How much immersion is enough? *Computer* 40, 7.
[40] M. Meehan, B. Insko, M. Whitton, and F. P. Brooks Jr. 2002. Physiological measures of presence in stressful virtual environments. *ACM Transactions on Graphics (TOG)* 21, 3, 645–652.
[41] D. Egan, S. Brennan, J. Barrett, Y. Qiao, C. Timmerer, and N. Murray. 2016. An evaluation of heart rate and electrodermal activity as an objective QoE evaluation method for immersive virtual reality environments. In *2016 8th International Conference on Quality of Multimedia Experience (QoMEX'16)*. IEEE, 1–6.
[42] C. Keighrey, R. Flynn, S. Murray, and N. Murray. 2017. A QoE evaluation of immersive augmented and virtual reality speech & language assessment applications. In *2017 9th International Conference on Quality of Multimedia Experience (QoMEX'17)*. IEEE, 1–6.
[43] A. Lecuyer, S. Coquillart, A. Kheddar, P. Richard, and P. Coiffet. 2000. Pseudo-haptic feedback: Can isometric input devices simulate force feedback? In *IEEE Proceedings of Virtual Reality, 2000*. IEEE, 83–90.
[44] X. K. Yang, W. S. Ling, Z. K. Lu, E. P. Ong, and S. S. Yao. 2005. Just noticeable distortion model and its applications in video coding. *Signal Processing: Image Communication* 20, 7, 662–680.
[45] X. Zhang, W. Lin, and P. Xue. 2008. Just-noticeable difference estimation with pixels in images. *Journal of Visual Communication and Image Representation* 19, 1, 30–41.
[46] Y. Zhao, Z. Chen, C. Zhu, Y. P. Tan, and L. Yu. 2011. Binocular just-noticeable-difference model for stereoscopic images. *IEEE Signal Processing Letters* 18, 1, 19–22.
[47] W. Hachicha, A. Beghdadi, and F. A. Cheikh. 2013. Stereo image quality assessment using a binocular just noticeable difference model. In *2013 20th IEEE International Conference on Image Processing (ICIP'13)*. IEEE, 113–117.
[48] W. Wu, A. Arefin, G. Kurillo, P. Agarwal, K. Nahrstedt, and R. Bajcsy. 2011. Color-plus-depth level-of-detail in 3D tele-immersive video: A psychophysical approach. In *Proceedings of the 19th ACM International Conference on Multimedia*. ACM, 13–22.

[49] R. Goldman. 2005. Curvature formulas for implicit curves and surfaces. *Computer Aided Geometric Design* 22, 7, 632–658.
[50] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. 2008. Meshlab: An open-source mesh processing tool. In *Eurographics Italian Chapter Conference*. 129–136.
[51] P. Cignoni, C. Rocchini, and R. Scopigno. 1998. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum* 17, 2, 167–174.
[52] G. Lavoué, E. D. Gelasca, F. Dupont, A. Baskurt, and T. Ebrahimi. 2006. Perceptually driven 3D distance metrics with application to watermarking. In *SPIE Optics+ Photonics (63120L-63120L)*. International Society for Optics and Photonics.
[53] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4, 600–612.
[54] G. Lavoué. 2011. A multiscale metric for 3D mesh visual quality assessment. *Computer Graphics Forum* 30, 5, 1427–1437.
[55] K. Wang, F. Torkhani, and A. Montanvert. 2012. A fast roughness-based approach to the assessment of 3D mesh visual quality. *Computers & Graphics* 36, 7, 808–818.
[56] F. Torkhani, K. Wang, and J. M. Chassery. 2014. A curvature-tensor-based perceptual quality metric for 3D triangular meshes. *Machine Graphics & Vision* 23, 1.
[57] K. Wang. 2012. Source code for fast mesh perceptual distance (FMPD). Retrieved from http://www.gipsa-lab.grenoble-inp.fr/~kai.wang/publications_en.html.
[58] I. Cheng, R. Shen, X. D. Yang, and P. Boulanger. 2006. Perceptual analysis of level-of-detail: The JND approach. In *8th IEEE International Symposium on Multimedia (ISM'06)*. IEEE, 533–540.
[59] H. Wang, I. Katsavounidis, J. Zhou, J. Park, S. Lei, X. Zhou, M. O. Pun, X. Jin, R. Wang, X. Wang, and Y. Zhang. 2017. VideoSet: A large-scale compressed video quality dataset based on JND measurement. *Journal of Visual Communication and Image Representation* 46, 292–302.
[60] S. Qin, S. Ge, H. Yin, L. Liu, and I. Heynderickx. 2009. Effect of experimental methodology on the JND of the black level for natural images. *Journal of the Society for Information Display* 17, 8, 687–694.
[61] A. B. Watson and A. Fitzhugh. 1990. The method of constant stimuli is inefficient. *Perception & Psychophysics* 47, 1, 87–91.
[62] I. Cheng and P. Boulanger. 2005. Feature extraction on 3-D TexMesh using scale-space analysis and perceptual evaluation. *IEEE Transactions on Circuits and Systems for Video Technology* 15, 10, 1234–1244.
[63] Link for the code: http://cs.utdallas.edu/multimedialab/qem4vr/.

Received September 2017; revised March 2018; accepted April 2018