GPU-based multiple-choice scheme for mesh simplification

Koh, Naimin; Zhang, Wenjing; Zheng, Jianmin; Cai, Yiyu

2018

Koh, N., Zhang, W., Zheng, J., & Cai, Y. (2018). GPU-based multiple-choice scheme for mesh simplification. Proceedings of Computer Graphics International 2018, 195-200. doi:10.1145/3208159.3208195

https://hdl.handle.net/10356/144946

https://doi.org/10.1145/3208159.3208195

GPU-based Multiple-Choice Scheme for Mesh Simplification

Naimin Koh
Nanyang Technological University
Autodesk Inc.
Singapore
koh059@e.ntu.edu.sg

Wenjing Zhang
Nanyang Technological University
Singapore
zhangwenjing@ntu.edu.sg

Jianmin Zheng
Nanyang Technological University
Singapore
asjmzheng@ntu.edu.sg

Yiyu Cai
Nanyang Technological University
Singapore
myycai@ntu.edu.sg

ABSTRACT

This paper presents a novel GPU-based parallel algorithm to simplify triangular meshes. Existing GPU based methods usually produce simplified meshes with lower quality. This is generally because they put more emphasis on parallelism than mesh quality. After a thorough analysis of the existing methods, we propose a GPU based multiple-choice mechanism, which combines multiple-choice decimation and GPU-based parallel simplification, to balance the mesh quality and computational speed. As a result, our algorithm improves the quality of the simplified meshes and also achieves high speed-up provided by GPU. The experiments and the comparison with the prior art confirm the performance of the proposed algorithm.

CCS CONCEPTS

• **Computing methodologies** → **Graphics processors**; **Mesh models**;

KEYWORDS

GPU, parallel computation, mesh simplification

ACM Reference Format:
Naimin Koh, Wenjing Zhang, Jianmin Zheng, and Yiyu Cai. 2018. GPU-based Multiple-Choice Scheme for Mesh Simplification. In *CGI 2018: Computer Graphics International 2018, June 11–14, 2018, Bintan Island, Indonesia.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3208159.3208195

1 INTRODUCTION

Mesh simplification is a fundamental process in many applications like 3D reconstruction, rendering and Virtual Reality. The purpose of simplification is to reduce the number of triangles while keeping the simplified mesh close to the original mesh as much as possible. This process, however, is usually quite slow for meshes with a large number of vertices. Many parallel algorithms have been proposed to speed up the simplification by exploiting the parallelism provided by GPU. To do parallel simplification, the algorithms need to deal with conflict carefully: different threads cannot modify the overlapped part of the mesh. Such a racing problem is the main focus in the existing methods [7, 12]. As a result, the simplified mesh may have low quality compared to a sequential greedy algorithm running on CPU. While the speed is important for the simplification, the quality of simplified mesh also plays much importance.

In this paper, we present a novel GPU-based mesh simplification algorithm, which takes both the speed and mesh quality into account. The method runs entirely on GPU and collapses multiple edges in parallel. Data transfer between CPU and GPU is minimized. Thus our algorithm is much faster than a CPU based method. Moreover, the quadric error metric (QEM) is adopted to evaluate the cost of simplification and edge collapse is used as basic simplification operator. To improve the mesh quality, we employ a multiple-choice mechanism: for each simplified edge, we first randomly find several candidates and use the one with the least QEM-based error, and then the independent edges are identified for parallel processing. A probabilistic analysis shows that this mechanism can improve the chance to collapse the correct edges in general. In this way, both the racing condition and the mesh quality are considered. This gives us a better trade-off between speed and quality. We demonstrate the method with several models and compare it with the state of the art.

2 RELATED WORK

The problem of simplifying triangular meshes has been extensively studied. Our focus is on the algorithms based on edge collapse with QEM, which is first proposed by [5] and known as QSlim. Various extensions have also been proposed to modify the metrics to incorporate color and UV coordinates to preserve attributes besides geometry [2, 6, 8].

However, for a mesh model with a large number of vertices the initial QSlim algorithm is time and memory consuming. All the edges are ranked with their cost and each time the one with least cost is chosen for collapse. An additional heap is maintained for this ranking. A memoryless version is later proposed [9]. Out of core algorithms are proposed to simplify very large meshes [11]. Multiple-choice algorithms [15, 16] address these problems with a different point of view. Instead of definitely choosing the best edge, it proposed a probabilistic optimization to increase the possibility

CGI 2018, June 11–14, 2018, Bintan Island, Indonesia

ACM ISBN 978-1-4503-6401-0/18/06…$15.00
https://doi.org/10.1145/3208159.3208195

to choose a good edge. There is no need to evaluate the cost for all edges or maintain a heap for ranking. The memory usage is largely reduced and the speed is accelerated.

Recently, GPGPU is more and more used to for parallel computation. Some researchers are devoted to speed up the mesh simplification with GPU [4, 7, 12, 14]. Their focus is on how to choose an appropriate set of edges for collapse without incurring conflicts. [4] adopts an octree to partitions the 3D space. The simplification is thus dependent on how the mesh is partitioned. [7] choose to collapse edges with local minimum cost, which guarantees the selected edges are independent. However, a local minimum edge is not guaranteed to incur a small cost and the efficiency is highly dependent of the number of local minimum edges. Shontz and Nistor [14] uses a CPU-GPU algorithm whose major part is executed on CPU and part of the algorithm is executed on GPU. Transferring data between CPU and GPU is very inefficient and should be avoided as indicated in many GPU-based algorithms. Recently, super independent vertices [12] are adopted to ensure selected edges are independent of each other. The quality of the simplified mesh generated with these algorithms, however, could be worse than that generated by a greedy-based sequential algorithm. While most of the existing methods adopt a triangle soup mesh representation, a recent work [10] adopts half-edge data structure for parallel simplification. Even though our method adopts triangle soup as [12], the idea of adopting multiple-choice scheme for better quality can also be incorporated into other data structure like half-edge.

3 THE PROPOSED METHOD

The input to our method is a triangular mesh $T = (V, F)$. $V \in R^{3n}$ is the vertex position. $F \in R^{3m}$ is the triangle indices, i.e. each triangle defines a triple (i, j, k) indicating the indices of vertices in V. The output if a simplified triangular mesh $T' = (V', F')$ with fewer triangles, the number of which is specified by the user. We do not require the mesh to be manifold. After these data is read from disk to CPU RAM, we allocate the required memory on GPU and stream the data in GPU memory. After that, all the computation is done on GPU.

Our method is composed of three components: QEM based edge collapse, multiple-choice decimation, and GPU parallelism. QEM is adopted to rate the edge collapse cost. Multiple-choice decimation is used to choose the "best" edge in a probabilistic view. GPU is exploited for parallel acceleration. We first give a brief introduction and analysis of these three components and their usage in existing methods from Section 3.1 to Section 3.3. We then give a detailed description of our algorithm in Section 3.4.

3.1 QEM based edge collapse

QEM [5] is a popular method to measure the error incurred for an edge collapse based on vertex-plane distance. An implementation based on QEM, QSlim, is quite successful. For a vertex $\mathbf{v}_i$, each of its adjacent triangles defines a plane $\mathbf{q} = (a, b, c, d)$, where (a, b, c) is the normalized normal and d is the distance to the origin. Thus the squared distance from a point v to the plane can be calculated as $D^2(\mathbf{v}) = (\mathbf{v}, 1)(\mathbf{q}^T\mathbf{q})(\mathbf{v}, 1)^T = (\mathbf{v}, 1)\mathbf{Q}(\mathbf{v}, 1)^T$. For a vertex $\mathbf{v}_i$, its quadric is the sum of those of its adjacent triangles: $\mathbf{Q}(\mathbf{v}_i) = \sum_{k \in N_i} \mathbf{Q}_k$, where N_i is the indices of triangles adjacent to $\mathbf{v}_i$.

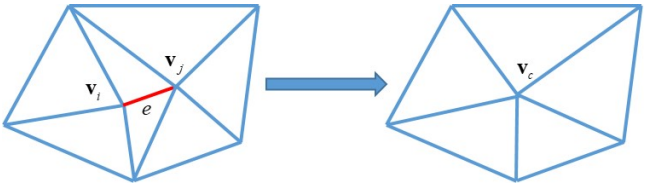


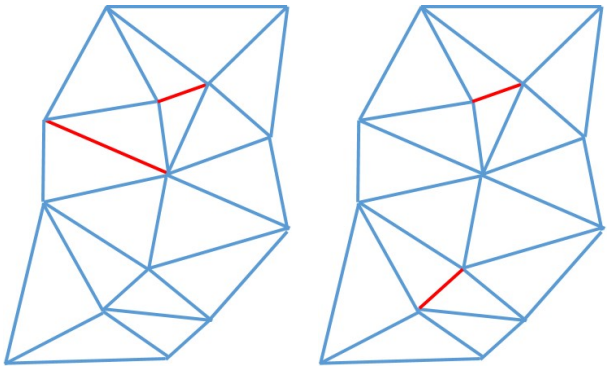
Figure 1: A demonstration of edge collapse. Edge $e(\mathbf{v}_i, \mathbf{v}_j)$ (red) is collapsed into one vertex $\mathbf{v}_c$.

For an edge collapse, we can use QEM to approximate the introduced geometric error. For an edge $e = (\mathbf{v}_i, \mathbf{v}_j)$, the resulting vertex after collapsed is denoted as $\mathbf{v}_c$, as show in Fig. 1. The error is defined as $D^2(\mathbf{v}_c) = (\mathbf{v}_c, 1)(\mathbf{Q}(\mathbf{v}_i) + \mathbf{Q}(\mathbf{v}_j))(\mathbf{v}_c, 1)^T$. The position of $\mathbf{v}_c$ is chosen as the one minimizing $D^2(\mathbf{v}_c)$. The $\mathbf{Q}(\mathbf{v}_c)$ for the new vertex is updated as $\mathbf{Q}(\mathbf{v}_c) = \mathbf{Q}(\mathbf{v}_i) + \mathbf{Q}(\mathbf{v}_j)$.

In QSlim, a greedy strategy is used to select edges for collapse. All the edges are ranked according to their incurred error D^2. Each time the edge with minimum cost is chosen for collapse and the edge costs for adjacent edges are updated and iterate again. A heap data structure is used to hold all the edges and choose the best edge.

QSlim works quite well. The simplified mesh is usually of good quality. A lot of derivatives are based on it [3]. However, two main problems exist for this greedy algorithm: slow computation and high memory consumption. Since each time we can only collapse one edge and the subsequent collapse depends on the previous ones, the algorithm is quite slow and cannot be parallelized. In addition, we need to compute the cost for all the edges and construct a heap data structure. This will inevitably consume a large portion of memory and be computationally intensive.

3.2 Multiple-choice decimation



(a) Conflicting edges (b) Independent edges

Figure 2: An illustration of conflicting edges (a) and independent edges (b).

Multiple-choice decimation (MCD) [16] is proposed to address the problem faced by the original QSlim method. It is based on a novel probabilistic analysis. If the mesh will be largely simplified, it is less stringent on our choice of the best edges: even if our current choice is not the best choice, it is OK if it will be eventually collapsed in the future. For each step, the multiple-choice algorithm selects multiple edges, among which the one with minimum edge cost is

finally collapsed. The more edges we choose as candidates, the less likely we choose a wrong edge. For example, assume the mesh is targeted to simplified to 5% of the original complexity. Each time we pick 8 possible edges randomly and choose the one with minimum cost. The probability for a wrong decision is:

$$(\frac{5}{100})^8 \approx 10^{-11}$$

By wrong decision, we mean all the 8 picked edges do not belong to the 95% majority of edges that are supposed to be removed. We refer the reader to [16] for more discussion on this probability analysis.

One main advantage of the multiple-choice method is that there is no need to store a heap data structure as QSlim. This reduces a lot of memory consumption. In addition, the edge cost is only computed on demand (when the edge is chosen as candidates). A lot of computation is saved. However, the original MCD is based on CPU. It can still be slow for very large meshes.

3.3 Parallel collapse based on super independent vertices

GPU based parallel methods [12] are proposed to collapse many edges at once. The main difficulty for parallel collapse is thread racing: edges chosen by multiple threads have overlapped influence area (See Fig. 1). In [12], a super independent vertices mechanism is used to address this issue. Super independent vertices are vertices at least 3 rings away from each other. Thus, each super independent vertex defines a separate region, in which only one edge is collapsed. This guarantees the independence of the selected edges.

To identify super independent vertices, multiple threads are used to scan the whole vertices and try to lock the surroundings (3-rings) of a certain vertex. If such a lock for a certain vertex succeeds, the vertex is called a super independent vertex. For all the super independent vertices, their surrounding regions are disjoint. For each super independent vertex, the adjacent edge with minimum cost is identified as candidate for collapse. Then all these edges are sorted with cost and a subset of these edges are collapsed at once.

The advantage of the GPU implementation in [12] is speed. It can be 5 times faster than QSlim. However, the quality of the simplified mesh is of lower quality. Unlike QSlim, we can not select the single edge with the minimum cost. We must collapse multiple edges at once even if the edge cost is actually very large. In [12], the author proposes to sort all the found edges and choose only a subset of these edges with smaller edge costs for collapse. However, this will incur more computation since we need to do the sorting. In addition, it is hard to specify the size of the subset. If the subset is too large, the quality will decrease and otherwise, the parallelism is decreased since the number of collapsed edge each time is reduced.

3.4 Algorithm

After analyzing previous algorithms, we propose to combine multiple-choice decimation and GPU-based parallel simplification. Our method utilizes GPU to do parallel simplification, i.e. we collapse multiple edges at the same time. However, different from [12] which chooses a single edge in an area defined by super independent vertex, we adopt the multiple-choice method to improve the quality in a probabilistic sense.

We use a similar data structure used in [12] to assist simplification. We build a Vertex-Triangle adjacency list. For each vertex, we store reference to its adjacent triangles. After the initialization of additional parameters like vertex quadrics, we can proceed to iterative decimation. We refer the reader to [12] for a detailed description of the data structure and initialization. For each iteration, we do parallel collapse. We first use GPU to randomly select a set of candidate edges. Then, independent edges are identified by locking the edge surroundings. All the independent edges are then collapsed together. After that, we update the mesh topology. We iterate the process until the number of the remaining triangles reaches the target.

Candidate edges identification. We divide our GPU threads into subgroups (e.g. 8 threads per group), each of which is responsible for choosing one candidate edge. Within each subgroup, all the threads are used to choose multiple candidate edges randomly and compute the cost for each edge. The edge with minimum cost is chosen as a candidate. To be specific, each thread is used to find an edge randomly. Since our data structure has no edge information, we do this through the Vertex-Triangle adjacency list. Each thread first randomly select a vertex, then an edge adjacent to this vertex is randomly selected and its corresponding edge cost is evaluated as indicated in Section 3.1. Among the edges selected by the subgroup, we choose the one with minimum edge cost as a candidate for collapse.

Independent edges identification. We then try to identify independent edges. This is done in a similar way in [12] to identify super independent vertex. For each candidate edge, we try to lock its surroundings by marking the one ring triangles adjacent to it. An additional buffer is used for the marking. Each thread tries to write into the buffer for its own candidate edge. After the marking, we check again the marker for each candidate edge's adjacent triangles. If the marker is consistent with the candidate edge, the edge is independent of others and collapsed. Otherwise, this edge is dropped without collapse.

Topology Update. After the edge collapse, the topology of the mesh is updated. The triangles deleted are marked as removed. The triangle indices are updated accordingly. The Vertex-Triangle adjacency list is updated according to [12].

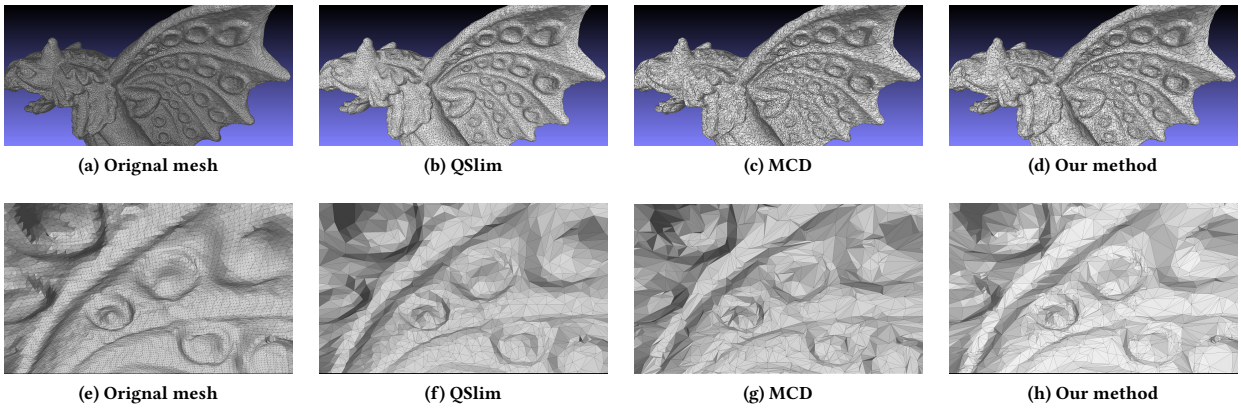
After that, we check the remaining number of triangles. We iterate again if it does not reach the target.

The algorithm is summarized as follows:

(1) Stream (V, F) into GPU and initialize quadrics for vertices.
(2) Build Vertex-Triangle adjacency list.
(3) For each thread, randomly select a vertex and then an edge adjacent to the vertex.
(4) For each selected edge, compute the edge cost.
(5) For each thread subgroup, select the edge with minimum cost as a candidate for collapse.
(6) For each candidate edge, try to lock its surroundings by marking its one-ring adjacent triangles. We adopt a similar strategy used in [12] to lock the surroundings: Each thread just try to write into the mark for its surrounding vertices without considering conflicts. After that, each thread just check whether the marks of its surrounding vertices are consistent with itself. If two threads are conflicting and trying

Table 1: Statistics of the simplification result on different models. Timing is in seconds. Quality is measured with geometric deviation [12].

Model	Triangles	Target	QSlim		MCD		Our method	
			time	quality	time	quality	time	quality
Gargoyle	1,726,420	86,320	22.86	0.01966	6.79	0.0478	1.30	0.03542
Dragon	7,218,906	360,944	109.02	0.0816	31.78	0.084	6.39	0.0836
Lucy	28,055,742	1,402,787	469.089	0.0187	124.2	0.0318	28.03	0.032



(a) Orignal mesh (b) QSlim (c) MCD (d) Our method

(e) Orignal mesh (f) QSlim (g) MCD (h) Our method

Figure 3: Comparison of the simplification results for the gargoyle model.

to mark the same vertices, the marks cannot be consistent for both threads at the same time.

(7) For all the edges whose surroundings are consistently locked, do the edge collapse.
(8) Update the mesh topology.
(9) If the number of the remaining triangles is larger than the target, go to step (3).

Compared with existing methods (QSlim, MCD and GPU-based simplification with super independent vertices), our method has the following advantages:

- our method can collapse multiple edges at the same time and adopts GPU for parallel computation. The time cost is largely reduced compared to a CPU based MCD and QSlim.
- at each time, our method selects the best edge within a subgroup. If the mesh is largely simplified, the probability that we will choose a wrong edge is exponentially decreased just as the CPU based MCD. As a result, the quality of the simplified mesh can be improved.
- each time, we only need to lock the one-ring triangles adjacent to the edge. [12], on the other hand, needs to lock three-ring of vertices adjacent to the super vertex. The locked area for our method is smaller. Thread conflicts are fewer in our method, which increases parallelism.
- Unlike [12], we do not need to sort the resulting edges according to the edge costs. This can simplify the implementation and reduce the computational cost.

Table 2: Comparison of geometric deviation between [12] and our method at different simplification levels for the Gargoyle model.

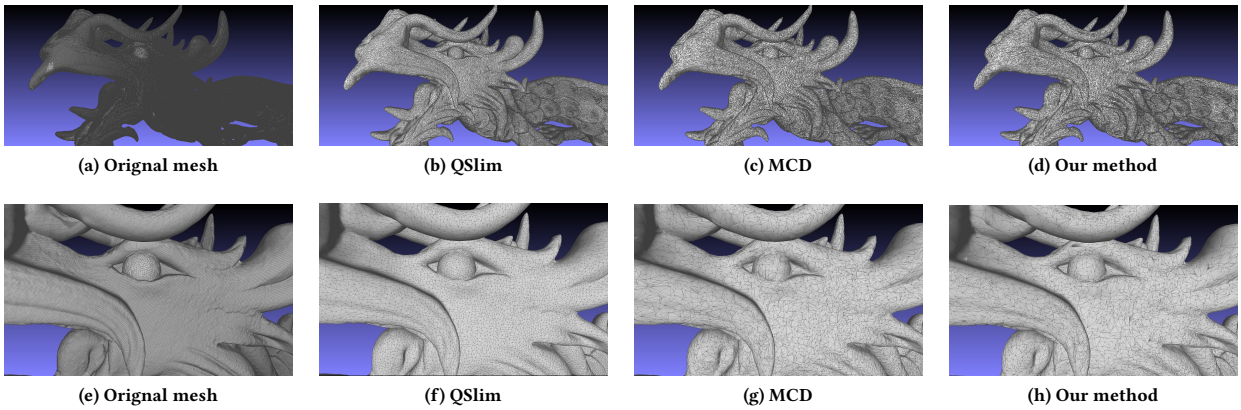
Target number of triangles	Our method	[12]
25%	0.009	0.01524
10%	0.021	0.03357
5%	0.03542	0.05961

4 EXPERIMENTAL RESULTS

We test our method on several models. Our method is implemented in C++ and runs on a computer with CPU i7-6700 3.4GHz and GPU GTX970. The GPU is programmed with CUDA Toolkit 8. We compare our method with QSlim, MCD and super independent vertices [12]. For simplification quality, we measure the geometric deviation used in [12]. The numerical results are summarized in Table 1.

We first compare our method with QSlim. We use the QSlim implemented by MeshLab 2016 [1]. Compared with QSlim, our method can run much faster but has a slight loss in mesh quality.

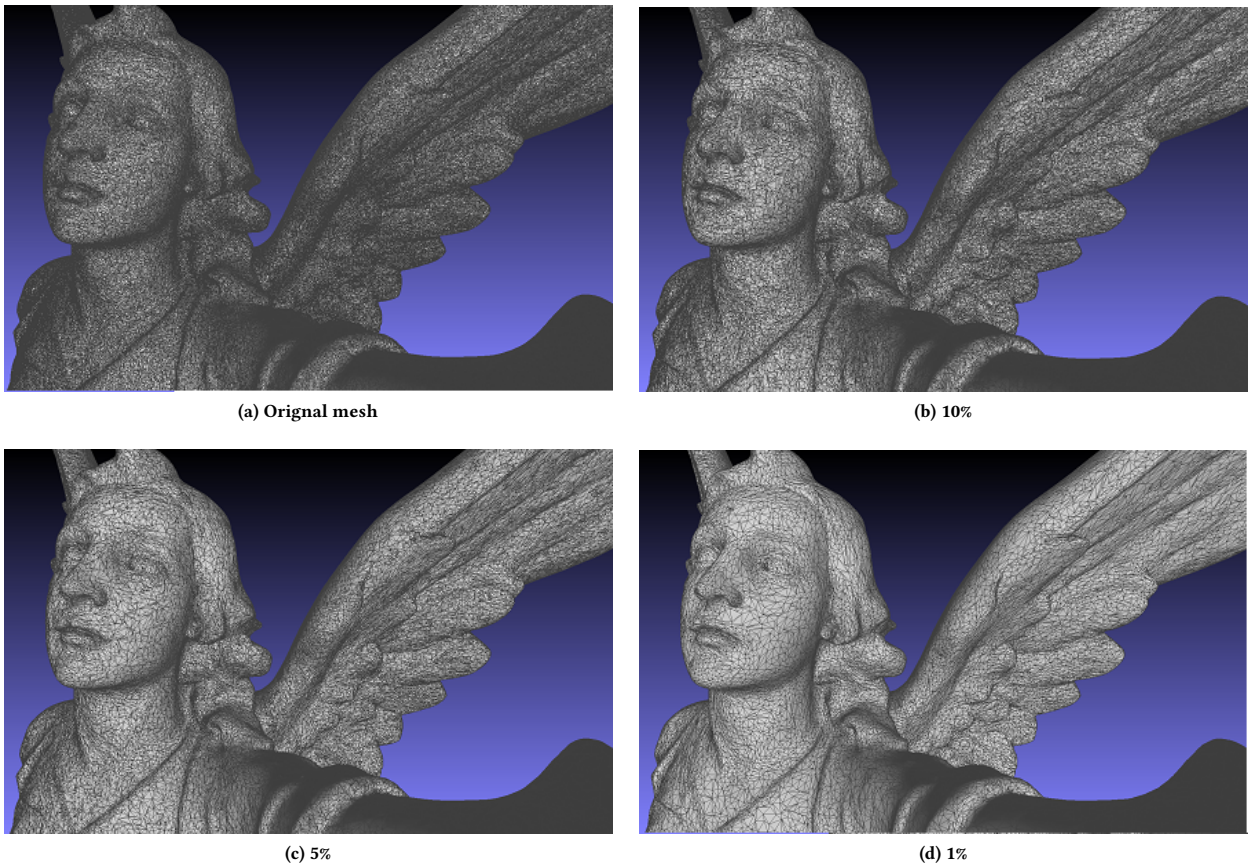
Compared to CPU based MCD, our method can still be faster thanks to the parallelism provided by GPU. The quality of our method is comparable with MCD. In some instances, our method is even better.



(a) Orignal mesh (b) QSlim (c) MCD (d) Our method

(e) Orignal mesh (f) QSlim (g) MCD (h) Our method

Figure 4: Comparison of the simplification results for the dragon model.



(a) Orignal mesh (b) 10%

(c) 5% (d) 1%

Figure 5: Simplification of the Lucy model at different levels. The number indicates the percent of the remaining triangles.

Since we cannot find an implementation in [12], we do not compare the time cost. However, it is reported that their method observe a speed up of 1.5-4.2 compared to QSlim. Our method, on the other hand, sees a speed up of 10-17. In addition, our method can generate higher quality simplified meshes. We compare the geometric derivation in Table 2. The data for [12] is got from their paper. Even though [12] suggests using only a subset of the selected edges for collapse to improve the quality, such a mechanism is very hard to control. If the subset is too small, many independent edges are not allowed to collapse, which largely reduces the parallelism provided by GPU. On the other hand, if the subset is too large, the quality of the simplified meshes is compromised. Our method, on the other

hand, only needs to select the size of each thread subgroup. We find that the subgroup with size 8 works well for all our models.

5 CONCLUSIONS

In this paper, we present a novel parallel simplification algorithm for triangular meshes. Our method is based on quadric error metric and edge collapse. GPU is employed for fast parallel processing. The multiple choice method is adopted to improve the quality. Our method can run fast and preserve good mesh quality. We also give a thorough analysis of the existing methods, which may guide future improvement.

Currently, we only consider the geometry of meshes. A lot of meshes in practical usage are actually equipped with texture [13]. These textures, however, usually enforce additional constraints. Extending our method to textured meshes sees more promising applications. The multiple choice method is based on probability. As the size of the mesh reduces, its efficiency is progressively compromised. Further improving the mesh quality using this mechanism without introducing too much computation burden needs further investigation.

ACKNOWLEDGEMENT

This work is supported by the National Research Foundation under Virtual Singapore Award No. NRF2015VSG-AA3DCM001-018.

REFERENCES

[1] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. 2008. MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference*, Vittorio Scarano, Rosario De Chiara, and Ugo Erra (Eds.). The Eurographics Association.

[2] Jonathan Cohen, Marc Olano, and Dinesh Manocha. 1998. Appearance-preserving simplification. In *SIGGRAPH'98*. 115–122.

[3] N. Coll and T. Paradinas. 2010. Accurate simplification of multi-chart textured models. *Computer Graphics Forum* 29, 6 (2010), 1842–1853.

[4] Christopher DeCoro and Natalya Tatarchuk. 2007. Real-time Mesh Simplification Using the GPU. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (I3D '07)*. ACM, New York, NY, USA, 161–166. DOI: http://dx.doi.org/10.1145/1230100.1230128

[5] Michael Garland and Paul S. Heckbert. 1997. Surface simplification using quadric error metrics. In *SIGGRAPH'97*. 209–216.

[6] Michael Garland and Paul S. Heckbert. 1998. Simplifying Surfaces with Color and Texture Using Quadric Error Metrics. In *Proceedings of the Conference on Visualization'98*. IEEE Computer Society Press, Los Alamitos, CA, USA, 263–269.

[7] N Grund, E Derzapf, and M Guthe. 2011. Instant Level-of-Detail. In *Vision Modeling and Visualization 2011*.

[8] Hugues Hoppe. 1999. New quadric metric for simplifying meshes with appearance attributes. *Proceedings Visualization '99* (1999).

[9] Peter Lindstrom and Greg Turk. 1998. Fast and memory efficient polygonal simplification. *Proceedings Visualization '98* (1998), 279–286.

[10] Thomas Odaker, Dieter Kranzlmueller, and Jens Volkert. 2016. GPU-Accelerated Real-Time Mesh Simplification Using Parallel Half Edge Collapses. In *Mathematical and Engineering Methods in Computer Science*, Jan Kofroň and Tomáš Vojnar (Eds.). Springer International Publishing, Cham, 107–118.

[11] Hiromu Ozaki, Fumihito Kyota, and Takashi Kanai. 2015. Out-of-core Framework for QEM-based Mesh Simplification. In *Proceedings of the 15th Eurographics Symposium on Parallel Graphics and Visualization (PGV '15)*. 87–96.

[12] Alexandros Papageorgiou and Nikos Platis. 2015. Triangular Mesh Simplification on the GPU. *The Visual Computer* 31, 2 (Feb. 2015), 235–244.

[13] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. 2001. Texture mapping progressive meshes. In *SIGGRAPH'01*. 409–416.

[14] Suzanne M. Shontz and Dragos M. Nistor. 2013. CPU-GPU Algorithms for Triangular Surface Mesh Simplification. In *Proceedings of the 21st International Meshing Roundtable*, Xiangmin Jiao and Jean-Christophe Weill (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 475–492.

[15] AntÃťnio W. Vieira, Thomas Lewiner, Luiz Velho, HÃĺlio Lopes, and Geovan Tavares. 2004. Stellar Mesh Simplification Using Probabilistic Optimization. *Computer Graphics Forum* 23, 4 (Sep. 2004), 825–838.

[16] Jianhua Wu and Leif Kobbelt. 2002. Fast Mesh Decimation by Multiple-Choice Techniques. *Vision,Modeling and Visualization* (Sep. 2002), 241–248.