A Linear Time Algorithm for High Quality Mesh Simplification

Hung-Kuang Chen[1,2,5], Chin-Shyurng Fahn[3], Jeffrey J. P. Tsai[4], Rong-Ming Chen[2], and Ming-Bo Lin'

'Dept. of Electronic Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, R. O. C.

[2]Dept. of Information and Design, Taichung Health-care and Management University, Taichung, Taiwan, R. O. C.

[3]Dept. of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, R. O. C.

[4]Dept. of Computer Science, University of Illinois at Chicago, U. S. A.

[5]hank@thmu.edu.tw

Abstract

High resolution 3D range scanning as well as iso-surface extraction have introduced densely and uniformly sampled models that are difficult to render at an interactive rate. To remove excessive details and produce meshes of various resolutions for different kinds of applications, the study of fast and high quality polygonal mesh simplification algorithms has become important. In this paper, we propose a new linear time algorithm that can achieve fast and high quality mesh simplification. In the new algorithm, we pipeline the cost computation, optimization, and edge collapse, and use a small constant-sized Replacement Selection min-heap instead of a large greedy queue to effectively reduce the runtime complexity to linear complexity. Compared to previous works, our new algorithm has at least three advantages. First, the new algorithm is runtime efficient. Second, the new algorithm is memory efficient. Third, the algorithm is capable of generating competitive high quality outputs.

1 Introduction

Polygonal meshes have been widely accepted as a boundary representation of 3D objects [1]. However, complex objects in fine details often require a high-resolution representation that contains a huge amount of polygons. Examples like the Digital Michelangelo Project [2] and the Visible Human Project [3] create models of size from tens of millions to billions triangles. Processing or rendering such models usually requires simplification. A polygonal mesh simplification algorithm is useful in at least two aspects. First, it can be used to remove excessive details that can be recovered by proper texture mapping techniques. Second, it can be used to construct multiresolution representations of the input mesh to satisfy the needs of various applications [4]. Among the previous works, four types of methods are significant: wavelet methods [5][6], vertex clustering methods [7]-[9], iterative contraction based methods [10]-[16], and reverse simplification (R-Simp) methods [17].

Hoppe et al. [11] first cast mesh simplification as an optimization problem. Three topological operators — the *edge collapse,* the *vertex split,* and the *edge swap* are applied iteratively to minimize an energy function that estimates the deviation of the current mesh from the original surface. The result has very good quality; however, the runtime cost for such optimization is unacceptably long. In his later work [12], a greedy-based structure together with the energy function was adopted to optimize the selection of candidate edges. Furthermore, a progressive representation of polygonal meshes called *progressive mesh* (PM) provides a continuous multiresolution representation by reversing the edge collapses according to previously stored history records of the collapsed edges. Inspired by the plane-based error metrics [13], Garland et al. proposed another metrics named quadric error metrics (QEM) [14] that measure the sum of squared distances from a relocated vertex to the set of planes spanned by all faces in its ring neighborhood. In [14], the face quadrics as well as all the vertex quadrics are computed and saved before simplification. Lindstrom and Turk suggested computing vertex quadrics on the fly [16]. This scheme eliminates the need of storage space for storing the vertex quadrics and generates better quality outputs at the cost of lower runtime efficiency. A more recent work used multiple-choice techniques [18]. The method seems very fast. However, the algorithm requires the input mesh to be a sequence of spatially ordered

0-7695-2217-3/04 $20.00 © 2004 IEEE

polygon soup. If the input mesh does not satisfy this requirement, an external sorting process must be executed. Moreover, the algorithm is incapable of differentiating boundary edges from non-boundary edges of the input mesh. If the input mesh contains many holes, a large portion of the in-core buffer is needed to hold the boundary edges until the whole input mesh is scanned. Moreover, such an approach adopted the half edge collapse operator rather than the full edge collapse operator; hence, it usually generates lower quality output than those using the full edge collapse operator with the optimal placement.

In this paper, we propose a novel polygonal mesh simplification algorithm that employs an iterative edge collapse procedure to generate high quality approximations of the input mesh with low main memory and runtime costs. The new algorithm has at least three benefits.

First, our algorithm is runtime efficient and has only $\Theta(n)$ time complexity. In comparison with [10]-[16], our method is significantly faster for two following reasons: first, it adopts two-level optimization that defers the calculation of the optimal placement of the new vertex, which saves a lot of calculation for solving a linear system; second, our method uses a pipeline scheme and a small constant-sized Replacement Selection (RS) min-heap rather than a large greedy queue. This approach successfully reduces the time complexity of the algorithm to be $\Theta(n)$ that is significantly lower than those of the methods proposed in [10]-[16].

Second, our algorithm is highly memory efficient. The new algorithm adopts the memoryless scheme suggested by Lindstrom et al., which computes the vertex quadrics on the fly [16]. Additionally, we apply a small constant-sized RS min-heap instead of a large greedy queue. These two schemes eliminate the main memory cost of the vertex quadrics and the large greedy queue.

Third, our algorithm produces very good quality outputs. Our method adopts the quadric error metrics and uses the full edge collapse operator with the optimal placement strategy. From experimental results, our method obviously generates higher quality outputs than the famous QSlim V2.0 does [19].

2 Preliminaries

Before introducing our work, we will briefly review a number of related works and terminologies in this section.

2.1 Ring

Figure 1 shows the ring of a vertex v, denoted as $r(v)$. For a ring, we only store the indices of the incident faces into an array. Hence, we define the ring of a vertex v to be the set of faces in the ring, i.e., $r(v) = \{f_0, f_1, f_2, f_3, f_4, f_5\}$. For better performance, we use an array of unsigned integers to store the ring of a vertex.

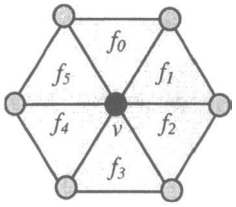


Figure 1. The ring of a vertex v.

2.2 The Edge Collapse

An edge $\overline{pq}$ is represented by two directed half edges denoted as $\overrightarrow{pq}$ and $\overrightarrow{qp}$ [20]. The full edge collapse operator on edge $\overline{v_s v_t}$ removes the edge $\overline{v_s v_t}$ and its incident faces. After the edge collapse operation, the end points of $\overline{v_s v_t}$ are merged to a new vertex v. By contrast, the half edge collapse operator $v_s \rightarrow v_t$ is equivalent to the collapse of $\overline{v_s v_t}$ that restricts the placement of the new vertex to v, or realizes the decimation of vertex v_s that fills the resulting hole by connecting v_t to the other non-neighboring boundary vertices [15]. An example of the above two edge collapse operations is illustrated in Fig. 2.

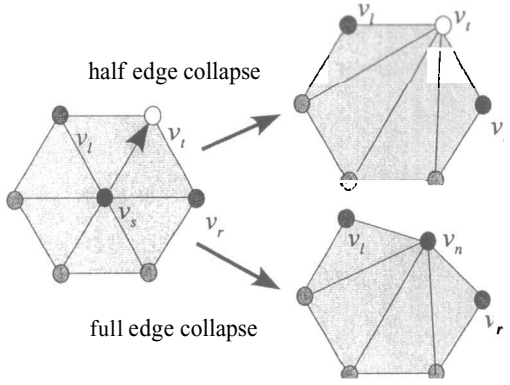


Figure 2. An example of the half and full edge collapses.

Considering the ring of a vertex v_s as shown in Fig. 2, it may be simplified by removing one of the non-boundary edges. We may treat the simplification of a mesh as a series of distinct vertex ring simplifications.

2.3 The Quadric Error Metrics

Garland and Heckbert proposed the use of the quadric error metrics, or simply denoted as QEM. The QEM originates from the plane-based error metrics proposed by Ronfard and Rossignac [13], which measures the maximum distances from a set of planes to a given vertex. Rather than measuring the maximum vertex-to-plane distances, the quadric error metrics calculates the sum of squared distances from a set of planes to the vertex. The

computation of such a measurement can be efficiently speeded up by matrix techniques; hence, it can lead to a very fast simplification algorithm. Since our method is based on this error metrics, the following is the brief review of its mathematic reasoning.

Each triangle in the original surface mesh spans a plane. Given a plane whose equation is $ax+by+cz+d=0$ or $\mathbf{n}^{\mathrm{T}}\mathbf{v}+d=0$ where $\boldsymbol{n}=[a \quad b \quad c]^{\mathrm{T}}$ is the unit face normal and d is a scalar constant, the squared distance from a vertex $\boldsymbol{v}=[x \quad y \quad z]^{\mathrm{T}}$ to the plane is

$$D(\mathbf{v})=(\boldsymbol{n}'\boldsymbol{v}+d)^2=\mathbf{v}^{\mathrm{T}}\mathbf{n}\mathbf{n}^{\mathrm{T}}\mathbf{v}+2d\mathbf{n}^{\mathrm{T}}\mathbf{v}+d^2. \quad (1)$$

For a given plane $\mathbf{n}^{\mathrm{T}}\mathbf{v}+d=0$, the ***fundamental quadric*** $\mathbf{Q}$ of the plane is defined as a 4x4 symmetric matrix

$$\mathbf{Q}=\begin{bmatrix}\mathbf{A} & \mathbf{b}\\ \mathbf{b}^{\mathrm{T}} & d^2\end{bmatrix}=\begin{bmatrix}\mathbf{n}\mathbf{n}^{\mathrm{T}} & d\mathbf{n}\\ d\mathbf{n}^{\mathrm{T}} & d^2\end{bmatrix}=\begin{bmatrix}a^2 & ab & ac & ad\\ ab & b^2 & bc & bd\\ ac & bc & c^2 & cd\\ ad & bd & cd & d^2\end{bmatrix}. \quad (2)$$

Therefore, the squared distance from a vertex $\boldsymbol{v}$ to the plane is

$Q(\bar{\mathbf{v}})=\bar{\mathbf{v}}^{\mathrm{T}}\mathbf{Q}\bar{\mathbf{v}}$, where $\bar{\mathbf{v}}=[\mathbf{v}^{\mathrm{T}} \quad 1]^{\mathrm{T}}=[x \quad y \quad z \quad 1]^{\mathrm{T}}$. (3)

Given a vertex $\boldsymbol{v}$, the vertex quadric of $\boldsymbol{v}$ is defined as the sum of the fundamental quadrics of the planes spanned by the faces incident to $\boldsymbol{v}$. Let $\boldsymbol{r(v)}$ be the set of faces incident to $\boldsymbol{v}$ and $\mathbf{Q}_i$ be the face quadric of face f_i, for $f_i \in \boldsymbol{r(v)}$. Thus, the vertex quadric of $\boldsymbol{v}$ is

$$\mathbf{Q}_{\boldsymbol{v}}=\sum_{\forall f_i \in r(v)}\mathbf{Q}_i. \quad (4)$$

Considering the full edge collapse of $\overline{\boldsymbol{v_s}\boldsymbol{v_t}}$, it introduces a new vertex $\boldsymbol{v_n}$. The error cost of this edge collapse can be estimated by the sum of squared distances from v_n to the set of planes spanned by the surfaces incident to $\boldsymbol{v_s}$ or $\boldsymbol{v_t}$. Let $\boldsymbol{r(v_s)}$ and $\boldsymbol{r(v_t)}$ be the rings of $\boldsymbol{v_s}$ and $\boldsymbol{v_t}$, respectively. The sum of squared distances from $\boldsymbol{v_n}$ to the set of original surfaces is

$$Q_s(\bar{\mathbf{v}}_n)+Q_t(\bar{\mathbf{v}}_n)=\sum_{\forall f_i \in r(v_s)}Q_i(\bar{\mathbf{v}}_n)+\sum_{\forall f_i \in r(v_t)}Q_i(\bar{\mathbf{v}}_n)$$

$$=\sum_{\forall f_i \in r(v_s)\cup r(v_t)}Q_i(\bar{\mathbf{v}}_n)=\bar{\mathbf{v}}_n^{\mathrm{T}}\left(\sum_{\forall f_i \in r(\quad)}\mathbf{Q}_i\right)(\bar{\mathbf{v}}_n), \quad (5)$$

where $\bar{\mathbf{v}}_n=[\mathbf{v}_n^{\mathrm{T}} \quad \mathbf{1}]=[x_n \quad y_n \quad z_n \quad 1]^{\mathrm{T}}$.

The error cost may be further minimized by solving an optimal placement of the new vertex $\boldsymbol{v_n}$ from $\nabla(Q_s+Q_t)(\bar{\mathbf{v}}_n)=2\mathbf{A}\mathbf{v}_n+\mathbf{2b}=0$, or $\mathbf{A}\mathbf{v}_n=-\mathbf{b}$. Note that the linear system has a unique solution only when the matrix $\mathbf{A}$ is non-singular or invertible.

3 The Simplification Algorithm

In the main, the algorithm is composed of three stages: the preprocessing stage, the simplification stage, and the output stage. In the preprocessing stage, the vertex rings are constructed through a single pass over the faces of the input mesh. Upon processing each face, the face index is inserted to the incident face lists of the three vertices. Following the preprocessing stage are the simplification and the output stages. The three stages are repeatedly performed until the termination condition is satisfied. Every pass of the loop outputs a new level of approximation. Thus, if k iterations are executed, the algorithm may produce k levels of details (LOD).

The algorithm accepts an input mesh $M(V, F)$ encoded in index face format comprising vertex geometries $\boldsymbol{v_i}=(x_i, y_i, z_i)\in V, i=1\sim|V|$ and triangle faces of three vertex indices $\boldsymbol{f_i}=(\boldsymbol{v_a}, \boldsymbol{v_b}, \boldsymbol{v_c})\in \boldsymbol{F}, \boldsymbol{i}=1\sim|F|$ as well as a termination condition variable ***goal*** that specifies the number of faces of the coarsest LOD. The algorithm is roughly outlined as follows:

Algorithm: High Quality Mesh Simplification (HQMS)

1. Calculate the set of rings, $\boldsymbol{R}=\{\boldsymbol{r(v)}|\boldsymbol{v}\in V\}$, as follows:
   For each triangle face $f_i(v_a, v_b, v_c)$ of $\boldsymbol{F}$, $i=1\sim|F|$,
   insert i to $\boldsymbol{r(v_a)}$, $\boldsymbol{r(v_a)}$, and $\boldsymbol{r(v_a)}$.

2. Simplify the input mesh on the basis of R **as** follows:
   1) Build **up** a minimum RS-Heap $\boldsymbol{H}$ of k records (k **is** a constant) **as** follows:
      (1) For each $r(v_i)$ of R, $i=1\sim k$,
      Compute $\boldsymbol{C(r(v_i))}=(e_{ij}, Q_i(\bar{\mathbf{v}}_j), \mathbf{Q_i})$ such that $Q_i(\bar{\mathbf{v}}_j)$ is the smallest for all internal edges of $\boldsymbol{r(v_i)}$.
      Insert $\boldsymbol{C(r(v_i))}$ to $\boldsymbol{H}$ with $Q_i(\bar{\mathbf{v}}_j)$ as the key.
   2) For each $r(v_i)\in \boldsymbol{R}, i=k+1\sim|V|$,
      (1) If $f_c(v_i)=1$, perform the following calculations:
      Compute $\boldsymbol{C(r(v_i))}=(e_{ij}, Q_i(\bar{\mathbf{v}}_j), \mathbf{Q_i})$ such that $Q_i(\bar{\mathbf{v}}_j)$ is the smallest for all internal edges of $\boldsymbol{r(v_i)}$.
      Delete the root of $\boldsymbol{H}, \boldsymbol{C(r(v_s))}=(e_{st}, Q_s(\bar{\mathbf{v}}_t), Q_s)$.
      Insert $\boldsymbol{C(r(v_i))}$ to H with $Q_i(\bar{\mathbf{v}}_j)$ as the key.
      (2) If $f_c(v_t)=1$, perform the edge collapse on $\boldsymbol{e_{st}}$ as follows:
      Find $I(e_{ij})=\{f \mid f=(v_s, \boldsymbol{v_t}, v_k)\vee(v_t, v_k, v_s)\vee(\boldsymbol{v_k}, \boldsymbol{v_s}, v_t)\vee(v_t, v_s, v_k)\vee(v_s, \boldsymbol{v_k}, v_t)\vee(\boldsymbol{v_k}, \boldsymbol{v_t}, \boldsymbol{v_s})\}$.
      Let $F=F-I(e_{ij})$ and $V=V-\{\boldsymbol{v_s}\}$.
      Set $f_c(v_s)$ and $f_c(v_t)$ to 2.
      Calculate $\mathbf{Q_t}$ and let $\mathbf{Q_t}=\mathbf{Q_s}+\mathbf{Q_t}$.
      Find the optimal placement of $\boldsymbol{v_t}$.
      (3) If $|V|<\boldsymbol{goal}$, proceed to Stage **3**.
   3) Repeat the following operations until $|H|=0$ or $|V|<\boldsymbol{goal}$:
      (1) Delete the root of $\boldsymbol{H}$, $\boldsymbol{C(r(v_s))}=(e_{st}, Q_s(\bar{\mathbf{v}}_t), \mathbf{Q_s})$.
      (2) If $f_c(v_t)=1$, perform the edge collapse on $\boldsymbol{e_{st}}$ **as** follows:
      Find $I(e_{ij})=\{f \mid f=(\boldsymbol{v_s}, \boldsymbol{v_t}, v_k)\vee(\boldsymbol{v_t}, \boldsymbol{v_k}, v_s)\vee(\boldsymbol{v_k}, \boldsymbol{v_s}, v_t)\vee(v_t, \boldsymbol{v_s}, v_k)\vee(v_s, v_k, v_t)\vee(v_k, v_t, \boldsymbol{v_s})\}$.
      Let $F=F-I(e_{ij})$ and $V=V-\{\boldsymbol{v_s}\}$.

Set $f_c(v_s)$ and $f_c(v_t)$ to 2.
Calculate Q_t and let $Q_t=Q_s+Q_t$.
Find the optimal placement of v_t.
(3)If $|V| < goal$, proceed to Stage 3.

3. If $|V| > goal$, go back to Stage 1; otherwise, output the remaining vertices and faces of $M(V, F)$.

3.1 Data Structures

As shown in Table 1, there are eight data structures used by our algorithm. The vertex and face tables, respectively, store the vertices and faces of the input mesh. The ring file stores the rings of the vertices, which provides local connectivity information to each vertex.

Table 1. The eight data structures and their sizes.

Data	Element Type	Element Size (bytes)	Number of Elements
Vertex table	VertexT	12	|V|
Face table	FaceT	12	|F|
Ring	RingT	4×MAXDEG[1]	|V|
Vertex flag	byte	1	|V|
Face flag	byte	1	|F|
Contraction flag	byte	1	|V|
Vertex index table	unsigned integer	4	|V|
Face index table	unsigned integer	4	|F|

1. *MAXDEG* is the maximum degree of the vertex connectivity-1.

Among these data structures, three of them serve as flag variables, i.e., the vertex flag, the face flag, and the contraction flag. The former two flags indicate the presence of the corresponding vertex and face, which are primarily used at the output stage for exporting the resulting mesh. The contraction flag is used to represent the status of the ring of the corresponding vertex. Let $r(v)$ be the ring of a vertex v and $R=\{r(v)|v\in V\}$. For each vertex v of V, we associate its ring, or $r(v)$, with a flag $f_c(v)$ whose value is given in Table 2.

Table 2. Values of the contraction flag.

Flag value	Meaning
0	$\|r(v)\|=0$
1	Unmodified
2	Modified
3	$\|r(v)\| \geq$ MAXDEG

simplified mesh.

3.2 The Preprocessing Stage

At this stage, the algorithm constructs the ring of each vertex from the face set, which gathers topological information about a vertex. To obtain better runtime efficiency, we implement the ring as an array of face indices. Upon visiting a face, the face index is inserted to the array of each vertex of the face. The first element of the array keeps the number of faces of the ring, or $|r(v_i)|$, where $v_i \in V, i=1\sim|V|$. An upper bound, ***MAXDEG***, to $|r(v)|$ of a vertex v has to be specified, so that the rings can be retrieved and cached efficiently through block transfer. However, a large upper bound may cause too much waste of memory space, while a small upper bound leads to too inferior quality. The upper bound should be carefully selected. The operations of this stage is summarized as follows:

For each $f_i(v_a, v_b, v_c)\in F$, $i=1\sim|F|$:
1. If $f_c(v_a)=0$, then initialize $r(v_a)$, insert f_i to $r(v_a)$, and set $f_c(v_a)=1$.
2. If $f_c(v_a)=1$ and $|r(v_a)| <$ ***MAXDEG***, then insert f_i to $r(v_a)$; otherwise, if $|r(v_a)|=$***MAXDEG***, then set $f_c(v_a)=3$.

3.3 The Simplification Stage

The simplification stage of our algorithm integrates the cost evaluation, optimization, and contraction into a single pipeline as illustrated in Fig. 3.

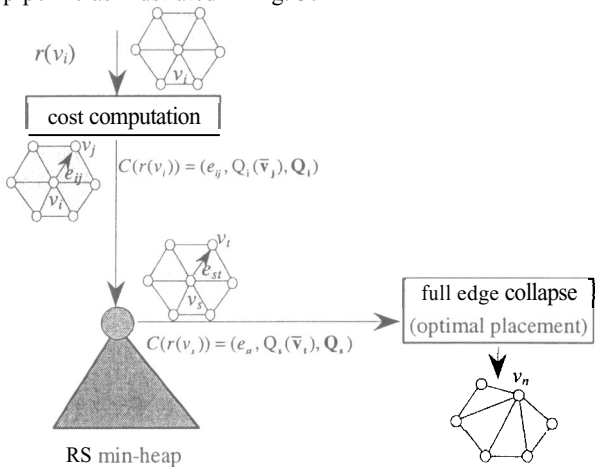


Figure 3. The simplification pipeline.

The simplification process has three substages corresponding to the three steps of sorted runs generation using the RS min-heap, i.e., the initialization, operation, and finale. [21]. In primary, the procedure computes the optimal contraction for a ring $r(v_i)$ as follows:

Let *cost* be a large number.
1. Compute the vertex quadric of v_i, $\mathbf{Q_i}$, from $r(v_i)$.
2. Compute the set of boundary vertices of $r(v_i)$, $h(r(v_i))=\{v_j | v_j \in r(v_i) \wedge v_j \neq v_i\}$.
3. $\forall v_j \in h(r(v_i))$, if $|r(v_i) \cup r(v_j)| < MAXDEG$ and $cost > \mathbf{Q}_i(\bar{\mathbf{v}}_j)$, then $C(r(v_i))=(e_{ij}, \mathbf{Q}_i(\bar{\mathbf{v}}_j), \mathbf{Q}_i)$.

In the finale step, the computation of the new contraction record is no longer needed. Instead **of** inserting new records, null records with large costs are inserted to the next-run part of the heap, so that valid records are squeezed out from the heap. The deletion and execution of the root record are repeatedly performed until the heap is full of null records or the number of the remaining faces is less than or equal to the required resolution.

Following an edge collapse, our previous work [22] suggested a set of dependency control strategies. This strategy is based on the definition of a set of dependent vertices called *dependent set.* According to such a set, all contractions are categorized into two classes: *strict dependent contractions* and *loose dependent contractions.* Let the dependent set be V_d, the set of dependent contractions be D, the set of strict dependent contractions be D_s, and the set of loose dependent contractions be D_l. Therefore, we have $D = D_s \cup D_l$, where $D_s = \{v_i \rightarrow v_j | v_i \in V_d\}$ and $D_l = \{v_i \rightarrow v_j | v_j \in V_d\}$. These two types of dependent contractions should be avoided.

To keep track of the vertices in the dependent set, we associate each vertex with a flag variable called *contraction flag*. The contraction flag values are assigned according to Table 2. In our implementation, given an edge collapse as depicted in Fig. 2, we denote $V_d = \{v_s, v_t\}$. By checking the corresponding contraction flag of v_t, we may easily identify the strict dependent contractions that should be excluded from execution. Prior to carry out a contraction record, for example, $C(r(v_i))=(e_{ij}, \mathbf{Q}_i(\bar{\mathbf{v}}_j), \mathbf{Q}_i)$, the contraction flags of both v_s and v_t are verified. If $f_c(v_s) \neq 1 \wedge f_c(v_t) \neq 1$, the contraction is not executed.

3.4 The Output Stage

The output stage exports the remaining vertices and faces in the face table. Let an iteration of the outer-most while-loop of the new algorithm be a step. For each step, the output stage generates a new level of approximation. If k steps are required to achieve the coarsest resolution, k levels of approximation are generated. These different levels of approximation naturally form a set of static LOD representations of the input mesh in which the original mesh is just the level-0 mesh, or M^0, and the output from the i-th step is the level-i mesh, or M^i.

3.5 Time Complexity Analysis

In the main, the algorithm is composed of three stages, namely, the preprocessing, the simplification, and the output stage. The time complexity of the algorithm is derived as follows.

The preprocessing stage requires a single pass over the entire face table; hence, the runtime cost of this stage is about $T_1=\Theta(|F_i|)=\Theta(|V_i|)$ for the i-th pass of simplification.

The simplification stage comprises three substages, i.e., the initialization, operation, and finale. The three substages totally require $|V_i|+h$ steps if the input mesh contains $|V_i|$ vertices and the heap consists of h records. Since h is a constant, the cost for insertion to or deletion from the RS min-heap in each step is also constant. Thus, the runtime cost, T_2, of this stage for the i-th pass of simplification is also $\Theta(|V_i|)$.

The output stage involves two loops to export the remaining vertices and faces. Hence, the time complexity of this stage, T_3, is also $\Theta(|V_i|)$ for the i-th pass of simplification.

The simplification proceeds iteratively until the input mesh is simplified to the desired resolution. Since $V_d=\{v_s, v_t\}$, i.e., $|V_d|=2$, about $|V/2|$ contractions are executed during pass-i simplification. Let $M_1(V_1, F_1)$, $M_2(V_2, F_2)$, …, and $M_i(V_i, F_i)$ be the output from pass-0, pass-1, …, and pass-(i-1) simplification, respectively. Then,

$$|V_1| = n-(n/2) = n/2,$$
$$|V_2| = |V_1| - (|V_1|/2) = |V_1|/2 = n\cdot(1/2)^2,$$
$$|V_i| = |V_{i-1}| - (|V_{i-1}|/2) = |V_{i-1}|/2 = n\cdot(1/2)^i,$$

If the desired resolution has n_0 vertices and the simplification successfully stops at pass-m, then $|V_m| \leq n\cdot(1/2)^m \leq n_0$. Therefore, $(1/2)^m \leq (n_0/n)$. By taking logarithms on the both sides, we have

$$m\log\left(\frac{1}{2}\right) \leq \log\frac{n_0}{n} \Rightarrow m \geq \frac{\log n - \log n_0}{\log 2} = O(\log n).$$

As a result, the algorithm requires about $O(\log n)$ passes to complete the simplification stage. Thus, the overall time complexity of the algorithm is

$$\sum_{i=0}^{m}(T_1+T_2+T_3) = \sum_{i=0}^{m}(\Theta(|V_i|)+\Theta(|V_i|)+\Theta(|V_i|))$$
$$= \sum_{i=0}^{m}(\Theta(|V_i|)) = \Theta\left(\sum_{i=0}^{m}\left(n\cdot\left(\frac{1}{2}\right)^i\right)\right)$$
$$= \Theta\left(n\cdot\sum_{i=0}^{m}\left(\frac{1}{2}\right)^i\right) = \Theta(n).$$

4 Experimental Results

In this section, we perform the experiments on a personal computer equipped with an Intel Pentium IV 2.2GHz processor, 1 GB DDR DRAM, and a disk system with three IDE 7,200 RPM hard disks. Three graphical models are employed in the experiments, including the Dragon mesh, Happy Buddha mesh, and David mesh. The test results are compared with those created by the famous

public domain software QSlim V2.0 [19]. The first two models are downloaded from the Stanford 3D Scanning Repository (http://www-graphics.stanford.edu/data/3Dscanrep/). The David mesh is downloaded from the Digital Michelangelo Project Archive of 3D Models (http://www-graphics.stanford.edu/dmich-archive/) [2]. The three models are all with the indexed-face representation encoded in PLY file format. Their statistics are listed in Table 3.

Table 3. Test models and their statistics.

Model	Dragon	Happy Buddha	David (2mm)
No. of vertices	437,645	543,652	3,614,098
No. of faces	871,414	1,087,716	7,227,031
File size (Mbytes)	33.8	42.6	124.07

In the experiments, the test models are simplified with our new algorithm and the public domain in-core simplification package QSlim V2.0. The results are then analyzed with the public domain software Metro V3.1 [23] to measure the geometric errors of all output models. The memory requirements, the program execution times, and the geometric errors of the outputs of our new algorithm (HQMS) as well as those of the QSlim V2.0 will be given in following subsections. To provide a basis for comparison, we prefer to use a public domain package rather than another version implemented by us for the sake of fairness. Since the QSlim V2.0 is well known for its good quality output and its fast runtime among in-core simplification methods, we therefore adopt it as the object of comparison.

4.1 Memory Requirements

The memory requirement includes the disk space and main memory space. According to Table 1, our new algorithm requires $(18+4\times MAXDEG)\times|V|+17\times|F|$ bytes disk space for intermediate data. In all experiments, we let MAXDEG be 20; hence, the new algorithm requires $98\times|V|+17\times|F| \approx 132\times|V|$ bytes disk space to accommodate the intermediate data. In comparison with the QSlim that roughly requires $268\times|V|$ bytes in-core memory, the main memory cost of our new algorithm is significantly lower.

Table 4. Main memory costs.

Model	Algorithm	RAM (MB)
Dragon	QSlim'	111.86
	HQMS	46.70
Happy Buddha	QSlim[1]	138.95
	HQMS	58.07
David (2mm)	QSlim[1]	923.71
	HQMS	427.37

1. Estimated by $268\times|V|$.

The main memory costs of the new algorithm and the QSlim for the simplification of the three test models are summarized in Table 4. From the results presented in Table 4, it is clear that the main memory cost of our algorithm is less than a half of that required by the QSlim. Thus, with the same amount of main memory resource, our algorithm is capable of simplifying meshes far larger than the QSlim does.

4.2 Runtime efficiency

In this subsection, the runtime efficiency is evaluated in terms of program execution times and triangle reduction rates. The experimental results are recorded in Table 5.

Table 5. The execution times and reduction rates (output=1,000 triangles).

Model	Algorithm	Execution Time (Sec.)	Reduction Rate (Triangles /Sec.)
Dragon	QSlim[1]	27.90	31,198
	HQMS[2]	8.22	103,706
Happy Buddha	QSlim[1]	35.24	30,837
	HQMS[2]	10.27	104,785
David (2mm)	QSlim[1]	N/A	N/A
	HQMS[2]	65.28	110,550

1. Using the optimal placement and area weights.
2. Heap size=15.

From Table 5, it is easy to see that our new algorithm is apparently faster than the QSlim V2.0. The triangle reduction rate of the HQMS is more than triple of the QSlim.

4.3 Output quality

The quality measurement is difficult and depends on the application in use. In this paper, the evaluation is judged by means of the geometric errors and the rendered images of the outputs. To provide a fair measurement over the geometric errors, we choose the Metro V3.1 provided by Cignoni et al. [23]. However, due to the limitation of the Metro, this measurement is not available for large meshes. On the other hand, the comparisons on the visual appearances are obtained from visually examining the images rendered by smooth shading of the OpenGL V1.2. The geometric errors of the simplified Dragon and Happy Buddha meshes are shown in Figs. 4 and 5, respectively.

From Figs. 4 and 5, it is clear to see that the geometric errors of the outputs from our algorithm is significantly lower than those produced by the QSlim V2.0. Moreover, the rendered images of Dragon and Happy Buddha meshes with 10,000 triangles are shown in Figs. 6 and 7, respectively. From these figures, we also found that the visual quality of the resulting simplified meshes from our algorithm is amazingly better than those created by the QSlim V2.0 using the optimal placement with an area weight policy. For the David mesh, both the Metro and QSlim are no longer applicable. Thus, we only illustrate the rendered images of two resolutions of the mesh in Fig. 8. From the experimental results shown above, we are confident to conclude that our algorithm is a very

successful algorithm in producing high quality outputs.

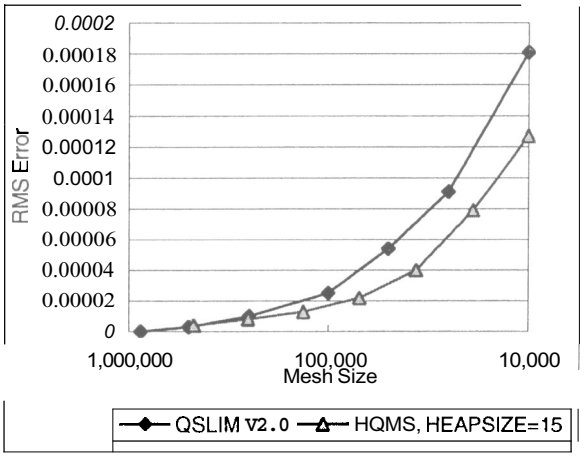


Figure 4. RMS errors of the simplified Dragon meshes resulting from the QSlim and the HQMS.

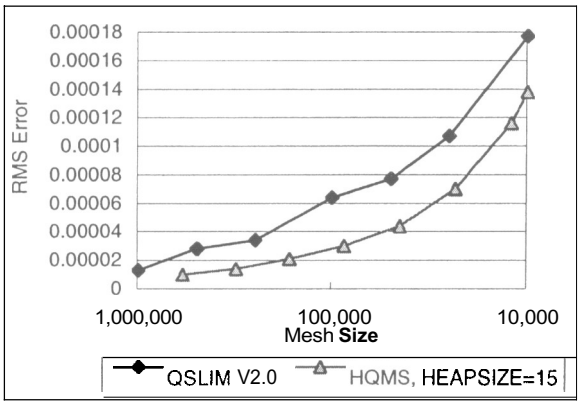


Figure 5. RMS errors of the simplified Happy Buddha meshes resulting from the QSlim and the HQMS.

5 Conclusions

In this paper, we have presented a new polygonal mesh simplification algorithm with $\Theta(n)$ complexity using iterative edge collapses and the quadric error metrics. The new algorithm integrates the computation, optimization, and full edge collapses into a single pipeline. This new design eliminates not only the storage cost of the vertex quadrics using the memoryless quadric computation suggested by [16] but also the cost of the contraction queue using a small constant-sized RS min-heap. The resulting algorithm is extremely runtime and memory efficient. On the comparison of time complexity, our new algorithm with linear time complexity obviously outdoes the QSlim V2.0 and most other iterative contraction based algorithms [10]-[16]. In addition, our algorithm requires significantly less memory space than the QSlim and most other similar algorithms do [10]-[16]. Furthermore, the new algorithm is proved to have the capability of producing very high quality approximations that are even superior to those obtained from the QSlim.

Acknowledgements

We would like to thank M. Levoy et al. from the Stanford Graphics Laboratory and P. Cignoni et al. from the Visual Computing Research Laboratory of ISTI of the Italian National Research Council for providing us the test meshes and Metro V3.1, respectively. We also thank M. Garland from the Department of Computer Science of the University of Illinois at Urbana-Champaign to make his QSlim V2.0 available.

References

[1] A. Watt, *3D Computer Graphics,* 3rd Ed., Addison-Wesley, Reading, MA, 2000.

[2] M. Levoy et al., "The digital Michelangelo project: 3D scanning of large statues," in *Proc. of SIGGRAPH* 2000, vol. 34, pp. 131-144, 2000.

[3] M. J. Ackerman, "The visible human project," in *Proc. of IEEE,* vol. 86, no. 3, pp. 504-511, 1998.

[4] D. P. Luebke et al., *Level-of-Detail for 3D Graphics*, Morgan Kaufman, San Mateo, **CA,** 2003.

[5] J. M. Lounsbery, "Multiresolution analysis for surfaces of arbitrary topological types," Ph.D. Dissertation, Univ. of Washington, Seattle, WA, 1994.

[6] M. Eck et **al.,** "Multiresolution analysis of arbitrary meshes," in *Proc. of SIGGRAPH '95,* vol. 29, pp. 173-181, 1995.

[7] J. Rossignac and P. Borrel, "Multiresolution 3D approximation for rendering complex scenes," in *Proc. of Geometric Modeling in Computer Graphics '93,* pp. 455-465, 1993.

[8] K. L. Low and T. **S.** Tan, "Model simplification using vertex clustering," in *Proc. of I997 Symp. on Interactive 3D Graphics,* **pp.** 75-82, 1997.

[9] P. Lindstrom, "Out-of-core simplification of large polygonal models," in *Proc. of SIGGRAPH* 2000, vol. 34, pp. 259-270, 2000.

[10] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of triangle meshes," in *Proc. of SIGGRAPH* '92, vol. 26, pp. 65-70, 1992.

[11] H. Hoppe et al., "Mesh optimization," in *Proc. of SIGGRAPH '93,* vol. 27, pp. 19-26, 1993.

[12] H. Hoppe, "Progressive meshes," in *Proc. of SIGGRAPH '96,* vol. 30, pp. 99-108, 1996.

[13] R. Ronfard and J. Rossignac, "Full-range approximation of triangulated polyhedra," in *Proc. of Eurographics* '96, **vol.** 15, no. 3, pp. 67-76. 1996.

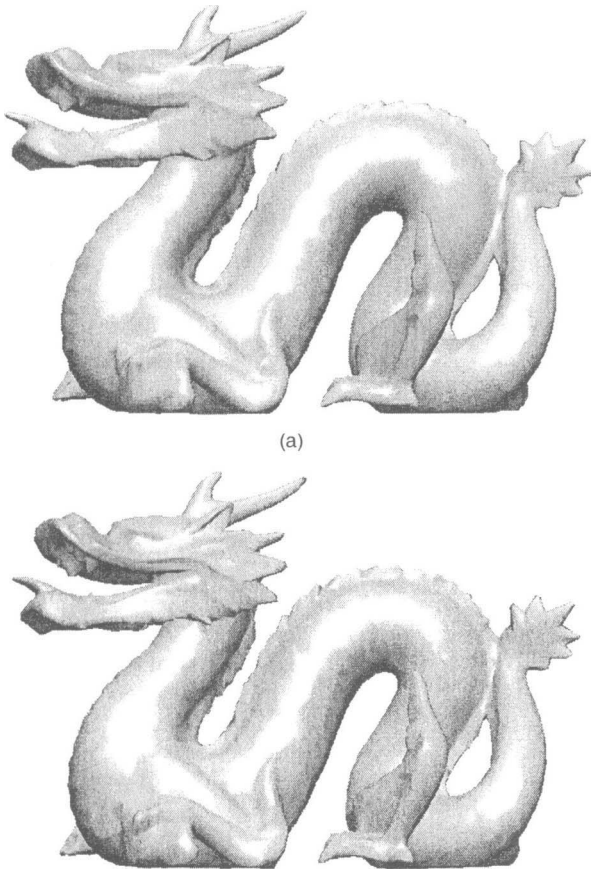
[14] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proc. of SIGGRAPH* '96, vol. 30, pp. 209-216, 1997.

[15] L. Kobbelt, **S.** Campagna, and H. P. Seidel, "A general framework **for** mesh decimation," in *Proc. of Graphics Interface'98*, pp.43-50, 1998.

[16] P. Lindstrom and G. Turk, "Evaluation of memoryless simplification," *IEEE Trans. on Visualization and Computer Graphics,* vol. 5, no. 2, pp. 98-115, 1999.

[17] D. Brodsky and B. Watson, "Model simplification through refinement," in *Proc. of Graphics Interface* 2000, pp. 221-228, 2000.

[18] J. Wu and Leif Kobbelt, "A stream algorithm for the decimation of massive meshes," in *Proc. of Graphics Interface '03*, pp. 185-192, 2003.
[19] M. Garland and P. S. Heckbert, *QSlim v.2.0 Simplification Software*, Dept. of Computer Science, Univ. of Illinois, http://graphics.cs.uiuc.edu/-garland/software/QSlim.html, 1999.
[20] K. Weiler, "Edge-based data structures for solid modeling in curved surface environments," *IEEE Computer Graphics and Applications*, vol. 5, no. 1, pp. 21-40, 1995.
[21] D. E. Knuth, *The Art of Computer Programming,* vol. 3, Addison-Wesley, Reading, MA, 1973.
[22] H. K. Chen et al., "A novel cache-based polygonal mesh simplification algorithm," submitted to *Journal of Information Science and Engineering,* 2004.
[23] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: measuring error on simplified surfaces," in *Proc. of Eurographics '98,* vol. 17, no. 2, pp. 167-174, 1998.



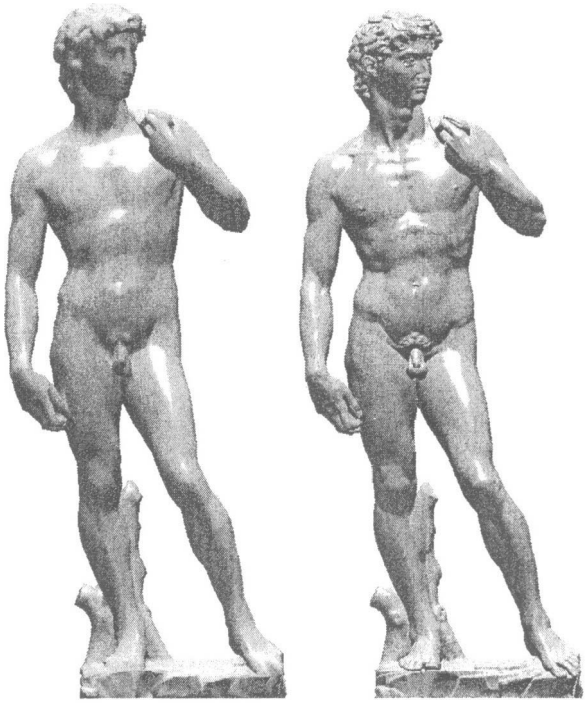
(a)

(b)

Figure 6. The simplified Dragon meshes with 10,000 triangles resulting from : (a) QSlim; **(b)** HQMS.

(a) (b)

Figure 7. The simplified Happy Buddha meshes with 10,000 triangles resulting from: (a) QSlim; **(b)** HQMS.



(a) (b)

Figure 8. The simplified outputs of the David mesh resulting from HQMS: (a) 5,416 vertices/10,872 triangle faces; **(b)** 139,011 vertices /277,874 triangle faces.