A Novel Cache-based Approach to Large Polygonal Mesh Simplification

HUNG-KUANG CHEN[1,3], CHIN-SHYURNG FAHN[2], JEFFREY J. P. TSAI[4] AND MING-BO LIN[1]

[1]*Department of Electronic Engineering*
[2]*Department of Computer Science and Information Engineering*
National Taiwan University of Science and Technology
Taipei, 106 Taiwan
[3]*Department of Information and Design*
Asia University
Taichung, 413 Taiwan
[4]*Department of Computer Science*
University of Illinois at Chicago
Chicago, IL 60637, U.S.A.

Traditional iterative contraction based polygonal mesh simplification (PMS) algorithms usually require enormous amounts of main memory cost in processing large meshes. On the other hand, fast out-of-core algorithms based on the grid re-sampling scheme usually produce low quality output. In this paper, we propose a novel cache-based approach to large polygonal mesh simplification. The new approach introduces the use of a cache layer to accelerate external memory accesses and to reduce the main memory cost to constant. Through the analysis on the impact of heap size to the locality of references, a constant sized heap is suggested instead of a large greedy heap. From our experimental results, we find that the new approach is able to generate very good quality approximations efficiently with very low main memory cost.

Keywords: cache-based polygonal mesh simplification, large mesh, iterative half-edge collapse, quadric error metrics, independent queuing

1. INTRODUCTION

The rapid advancements in 3D scanning technology have introduced a great challenge: the processing and rendering of large meshes. Recent examples such as the Digital Michelangelo Project [1], the Forma Urbis Romae Project [2] at Stanford University on culture heritages preservation and the Visible Human Project [3] of the National Library of Medicine for medical applications, have created large, unsegmented 3D surface meshes over tens of million faces. Rendering and processing such meshes usually exceeds the capability of current computer systems. Therefore, mesh simplification algorithms capable of converting an input mesh into various level-of-detail (LOD) representations are becoming extremely important.

Polygonal mesh simplification (PMS) has been studied intensively. A host of works have been proposed. Among these works, three types of PMS algorithms are most widely

Received April 7, 2004; revised February 1 & August 2, 2005; accepted August 29, 2005.
Communicated by Pau-Choo Chung.

used, i.e., the iterative edge/half-edge collapse based simplification algorithms [4-8], the vertex decimation based algorithms [9], and the vertex clustering based algorithms [10-12]. The former two types produce good quality approximations, but their runtime efficiency is low. While sampling with low resolution grids, the third type of PMS algorithms appears to be more runtime efficient, but its output is much inferior. Ironically, most of these works are designed only for in-core execution; thus, they are not capable of simplifying large meshes [4-11]. To address this issue, a number of works have been proposed [13-22]. Before introducing our own work, we will first review some earlier significant works on large mesh simplification.

Hoppe proposed a method to simplify large terrain meshes by hierarchically segmenting a terrain mesh into disjoint blocks followed by a series of edge collapses of the blocks [15]. This method requires that special care be taken with the boundary edges and involves an expensive process for rejoining the blocks. Prince extended this work to deal with general meshes [13]. However, the main memory space requirement reported in [13] is not constant and may exceed the available space.

Elsana and Chiang proposed an external memory method for view-dependent simplification using iterative edge collapses [16]. Instead of spatially partitioning the input mesh, they divide the input mesh into disjoint spanned sub-meshes using the span relationship. Edge collapses are applied to each independent sub-mesh. To enable external memory execution, they keep the priority queue in the external memory space. Their approach requires intensive non-local external memory accesses; hence, the runtime efficiency is low. Furthermore, they did not report any results for large meshes.

Lindstrom proposed an out-of-core PMS algorithm capable of simplifying large complex meshes that are too large to fit into the main memory space [12]. This algorithm adopts the uniform grid vertex clustering method proposed by Rossignac and Borrel [10]. Different from [10], Lindstrom's method begins by accumulating face quadrics in each grid and then determines the representative vertex of each grid by solving the accumulated quadric matrix, similar to the optimum placement of an edge collapse, as proposed in [5]. Their method achieves excellent runtime efficiency. Nevertheless, it suffers from at least two drawbacks. First, it requires a considerable amount of main memory to hold the output meshes as well as the grid quadrics. Second, the quality of the output mesh is low in comparison with the quality achieved with iterative edge collapses based methods. The work proposed in [18] addressed the former issue and provided a solution by preprocessing the input mesh via an external sort.

Another stream of works is based on the Reverse Simplification (R-Simp) method [14, 17, 23]. Inspired by [23], Shaffer and Garland developed a method that begins with quadric quantization based on a uniform grid, followed by BSP-Tree construction to cluster vertices adaptively [14]. In comparison with the previous methods, this approach yields better quality output at the cost of lower runtime efficiency. Choudhury and Watson proposed a virtual memory enhancement version of the R-Simp methods, enabling better virtual memory utilization based on a depth-first simplification approach that increases the reuse of working sets [17]. However, this method was found to be slower than Shaffer and Garlands' approach and had an inherent limitation imposed by the maximum virtual memory space.

Recently, a number of external memory iterative edge collapse-based PMS algorithms have been proposed in [19-22]. Cignoni *et al.* proposed an external memory data

structure, called the Octree based External Memory Mesh (OEMM), for general management of large meshes, it includes mesh manipulation, editing, filtering, simplification, and selective inspection [19]. Their method provides a hierarchical spatial partitioning scheme supporting global indexing and partial retrieval operations. However, the external memory data structures must be constructed offline in advance.

Wu and Kobbelt proposed a stream-lined approach based on a multiple choice technique [20]. The method was found to be fast; however, its input mesh has to be a sequence of spatially ordered polygons, or an external spatial sort over the input mesh such as the one used in [18] is required. In addition, the algorithm is incapable of differentiating the boundary edges of an input mesh. If the input mesh contains a large number of holes, a great portion of the in-core buffers will be needed to hold these boundary edges until the whole input mesh has been scanned. Isenburg et al. suggested adapting the computation according to the processing sequence paradigm [24] over the input dataset to solve this problem [21].

Shaffer and Garland proposed a new external memory multiresolution surface representation based on an external memory octree [22]. They claimed their method is insensitive of input meshes and is capable of dealing with very large meshes. Nevertheless, it still requires a sophisticated offline construction process before the generation of a simplified mesh. Furthermore, the simplification algorithm is output sensitive. $O(|V_{out}|)$ memory space is required to hold an output mesh, and $O(|V_{out}|\log(|V_{out}|))$ time is required to generate a specific-sized mesh, where $|V_{out}|$ is the number of vertices in the specified output mesh.

2. CACHE-BASED POLYGONAL MESH SIMPLIFICATION

Our new approach, called cache-based polygonal mesh simplification (CBPMS), comprises two layers, i.e., simplification and cache layers. A block diagram of the new approach is shown in Fig. 1.

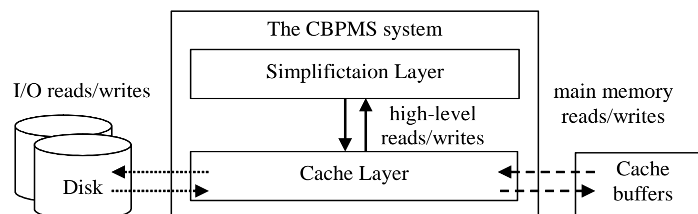


Fig. 1. A block diagram of the CBPMS system.

Without loss of generality, the input mesh of our algorithm is assumed to be a binary index faced triangular mesh. The input mesh consists of a set of triplets of floating point numbers representing the *X*, *Y*, and *Z* coordinates of the vertices in the 3D Euclidean space together with a set of triplets of indices to the vertices of the triangular faces. Furthermore, the input mesh may be either manifold or non-manifold.

The simplification layer adopts a framework that is similar to the independent queu-

ing [25] but with three differences. First, we propose a new contraction control strategy that provides a more flexible definition over the dependent contractions. Second, instead of a large external priority queue, a fix-sized RS-heap is built to improve the quality of output meshes; it not only requires less memory but also is more runtime efficient. Third, the contractions are considered on each vertex ring rather than on each edge ring; hence, many fewer data references and computations are required. To allow the simplification layer to work in a main memory insensitive way, all the data structures as well as read or write access are managed by the cache layer. The two layers will be discussed in the following two subsections.

2.1 The Simplification Layer

The simplification layer comprises two phases. In the first phase, called the setup phase, the vertex quadrics of all the vertices are computed. In the second phase, called the simplification phase, four stages are run iteratively until the input mesh is simplified as the goal parameter requires or until no further simplification is possible.

2.1.1 The setup phase

In the setup phase, the vertex quadric of each vertex is computed by accumulating the fundamental quadrics of the faces in the neighborhood of the vertex according to [5]. The computation of the fundamental quadric $\boldsymbol{Q}_i^f$ associated with a triangular face f_i can be done as follows:

1. Get the coordinates of the three vertices of f_i.
2. Find the plane equation of the plane spanned by the triangle: $ax + by + cz + d = 0$.
3. Compute $\boldsymbol{Q}_i^f$ by letting

$$\boldsymbol{Q}_i^f = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} [a \quad b \quad c \quad d] = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}. \tag{1}$$

Let r_i be the set of faces in the vertex ring of a vertex v_i. The vertex quadric $\boldsymbol{Q}_i^v$ of the vertex v_i can be given by $\boldsymbol{Q}_i^v = \sum_{\forall f_j \in r_i} \boldsymbol{Q}_j^f$.

2.1.2 The simplification phase

As we have mentioned earlier, four stages are run iteratively until the simplification is done in the simplification phase. The first stage is called the vertex ring construction stage. In this stage, the algorithm constructs the vertex rings of all the vertices from the input mesh. The next stage is called the computation stage. In that stage, the optimal contraction of each vertex ring is computed and inserted into a constant-sized Replace Selection (RS) min-heap [26] according to their error costs. In the third stage, called the contraction stage, the contractions from the sorted runs of contractions generated in the sec-

ond stage to the vertex rings are performed according to the contraction control strategy (DCS). In the last stage, named the output stage, the simplified mesh and associated data structures are exported as the inputs to the next iteration, and the flag variables are reset as described later.

2.1.3 The vertex ring construction stage

The vertex ring r_i of a vertex v_i can be represented as the set of faces adjacent to vertex v_i. In practice, this set of faces may be implemented by means of a list of face indices. A tradeoff between disk space and disk access efficiency should be considered prior to the determination of an array or a cluster chain implementation of the list. Since disk space is cheap and disk access efficiency is important, we implement the list by means of a fixed length array. When a ring contains more faces than the array can hold, it is not considered in the later stages. The pseudo code for the construction of vertex rings is shown below. Note that when the number of faces exceeds a prescribed threshold, MAXDEG, the ring is labeled as *heavy* state and is not further processed:

For each vertex v_j of f_i, $i = 1 \sim |F|$, perform the following operations:
If $|r_j| = 0$, initialize r_j, insert f_i to r_j, and set the contraction flag of ring r_j to *valid* state.
Otherwise, insert f_i to r_j if $|r_j| \leq$ MAXDEG or set the contraction flag of ring r_j to *heavy* state if $|r_j| >$ MAXDEG;

2.1.4 The computation stage

In the computation stage, the algorithm first computes the best contraction for each ring by finding the smallest error cost half-edge collapse from valid half-edge collapses. Afterwards, an RS min-heap is used to generate sorted runs of the contraction records. A schematic view of this stage is shown in Fig. 2.

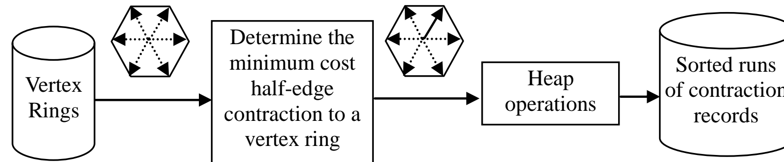


Fig. 2. A schematic view of the computation stage.

(1) Contraction records and error cost computations

Given a vertex ring r_s of v_s, the *internal outward half-edges* are those whose source is v_s and whose target is one of the boundary vertices. The best contraction to r_s is, therefore, the valid half-edge collapse with the smallest error cost among all internal outward half-edges. Let l_s be the list of boundary vertices of r_s. Then, the best contraction to ring r_s is computed by

$$\varepsilon_s = \min\left\{\varepsilon_i \mid \varepsilon_i = \hat{v}_i^T Q_s^v \hat{v}_i, \text{ where } \hat{v}_i = \begin{bmatrix} v_i \\ 1 \end{bmatrix} \text{ and } v_i \in l_s\right\}. \quad (2)$$

Let the half-edge $\overrightarrow{v_s v_t}$ be the best contraction to r_s with the smallest error cost ε_s. A contraction record c_s of ring r_s, consisting of a triplet of the source vertex v_s, the target vertex v_t, and the error cost ε_s is then inserted into the RS min-heap.

(2) The generation of sorted runs of contraction records

The order of the contractions may be arbitrary, yet it has an impact on the quality of the resulting mesh and the locality of references. At one extreme, the non-optimizing queuing approach does not sort contractions; at another extreme, the greedy queuing approach enforces a strictly increasing order of contractions according to their error costs [25]. Generally, the latter gives higher quality output but has much lower locality of references and higher frequency updates; thus, it is believed to be unsuitable for external memory implementation [14]. In this paper, we propose a compromise approach that adopts a fix-sized RS-heap to partially sort contractions.

The heap size has a great impact on the runtime performance. According to [26], the average run length output from an RS-heap is roughly two times the heap size. Let the number of vertices of the input mesh be n, let the heap size be h, and let the number of elements in a cache be CMS. We can set the heap size to be

$$h = \begin{cases} n, & CMS / c \geq n \\ CMS / c, & CMS / c < n \end{cases}, \tag{3}$$

where c is a constant. The impact of the heap size on the runtime efficiency can be evaluated from two perspectives: the contribution to the runtime complexity and the locality of the references in the resulting sequence. Since CMS and c are supposed to be constants for a given set of parameters, the heap size can be regarded as a small constant value if $CMS/c << n$. Notice that CMS is related to the size of the cache memory. If the cache buffer in use is small relative to the mesh size, the variation of the heap size will not have a significant effect on the runtime complexity of the simplification. Therefore, we will concentrate on the influence of the change of the heap size on the locality of references. The theory behind our approach and the experimental results that we have obtained are discussed in the following.

Let $H = \{\hat{r}_k \mid \hat{r}_k \in \hat{R}\}$ be an arbitrary ordered sequence of references to an ordered set $R = \{r_1, r_2, ..., r_{|V|}\}$ where $\hat{r}_i$ represents a reference in H to r_i in R. Given any two consecutive references $\hat{r}_i$ and $\hat{r}_j$ to R, we define the distance of the references as

$$d(\hat{r}_i, \hat{r}_j) = |j - i|. \tag{4}$$

Let $(\hat{r}_m, \hat{r}_n)$ denotes a pair of successive references in H to R in which the reference $\hat{r}_n$ is a successor to the reference $\hat{r}_m$ in H. We can further define the maximal distance of successive references from H to R as

$$D(H \mid R) = \max\{d(\hat{r}_m, \hat{r}_n\} \mid \forall(\hat{r}_m, \hat{r}_n) \text{ of } H\}. \tag{5}$$

In the four stages of the simplification layer, the reference sequence of the vertex quadric calculation, vertex ring construction, computation, and output stages are mainly

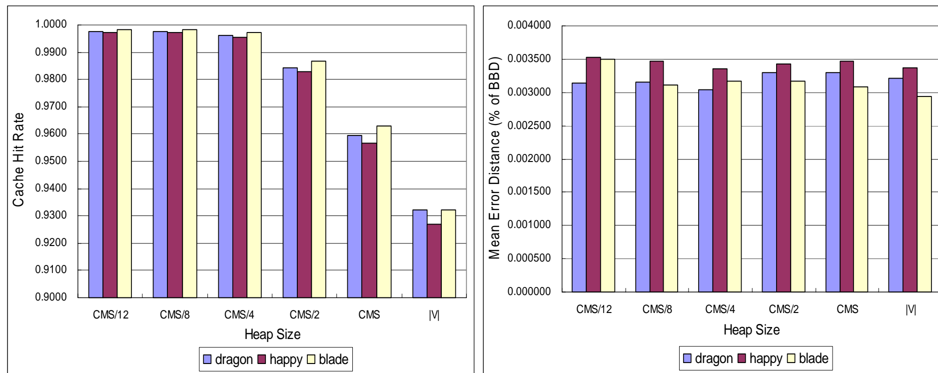
sequential and not affected by the heap size. Hence, there is no need to analyze the impact of the heap size on the reference locality of these stages. In consequence, we only concentrate on the reference locality of the output sequence of the RS-heap.

Assume that the sorted run sequence $H = \{h_i \mid h_i$ contracts vertex $v_i\}$ generated by the RS-heap is used to determine the order of contractions for the vertex rings. In theory, we can approximate the locality of references of H by finding the maximal distance of successive references issued by successive contraction records of the sequence.

Considering two successive input items, they may be output from the heap to the same run or two successive runs, which implies

$$1 \leq d(h_{i-1}, h_i) \leq 2 \times \text{run length} - 1. \quad (6)$$

Note that the average run length for random inputs is roughly $2h$. In a specific case, if we let $h = CMS/4$, then $d(h_{i-1}, h_i) \leq CMS - 1$, i.e., successive references are restricted to a local region that is no larger than CMS records. To verify this proposition, we performed a number of experiments on the Dragon, Happy Buddha, and Blade meshes. The results are presented in Fig. 3.



(a) The cache hit-ratios. (b) The approximation errors.

Fig. 3. The impact of the heap size.

From Fig. 3 (a), it is clear that the cache hit-ratios decrease radically when $c \geq 4$ for all three meshes and that the cache hit-ratios are higher than 99% when $c \geq 4$. As for the output quality, the results shown in Fig. 3 (b) reveal that the mean error distance of the output meshes (1,000 triangles) does not decrease much as c increases. Hence, a choice of $c \geq 4$ is suggested, i.e., $h \leq CMS/4$.

2.1.5 The contraction stage and the dependency control strategies

In the contraction stage, the algorithm accepts a sequence of sorted runs of contractions generated by the Replacement Selection Sort [26]. The contractions are executed according to this partially sorted sequence and a dependency control strategy (DCS) that is employed to find dependent contractions prohibited from executing. Before discussing

of the half-edge collapse procedure in the contraction stage, we will elaborate on the dependent contractions and our dependency control strategies.

Let h_s be a contraction to the ring of v_s by collapsing the half-edge $\overrightarrow{v_s v_t}$. A contraction h_a to the ring of v_a is a *dependent contraction* of another contraction h_b to the ring of v_b if the ring of v_a is modified by h_b. Given a vertex v_s, the ring of v_s, denoted as r_s, and a contraction h_s to r_s, the conventional definition of the set of dependent contractions of h_s is $H_s = \{h_i \mid v_i \in r_s\}$.

On the basis of the Dobkin-Kirkpatrick (DK) hierarchy [27], the independent queuing approach suggests disabling all these dependent contractions [25]. Thus, if there are k vertices on the boundary of the ring of v_s, k rings are not allowed for contraction. For a manifold surface, the average value of k is six, which implies that only 1/7 of rings are contracted on an average. This policy takes creates a deeper hierarchy of contractions and leads to low runtime efficiency. Instead of the above conventional definition, we have developed a new set of rules as follows.

Initially, we let the vertices of the input mesh be in state S_0, where no restriction is placed on the contraction of the vertex rings. Given a ring of a vertex v_s, a possible contraction to the ring is performed by h_s: $v_s \rightarrow v_t$. Following the execution of h_s, the vertices of the ring are transformed into one of four states, namely, s_1, s_2, s_3, and s_4: the source and target vertices of the contracted half-edge, respectively, are changed into states s_1 and s_2, the vertices opposite to the contracted half-edge $\overrightarrow{v_s v_t}$ are turned into state s_3, and all other boundary vertices of the vertex ring are transformed into state s_4. According to their states, the vertices can be classified into four disjoint sets: S_1, S_2, S_3, and S_4, where S_i represents the set of vertices in state s_i for i = 1, 2, 3, and 4. Fig. 4 shows an example ring and the states of its vertices after the contraction is performed.

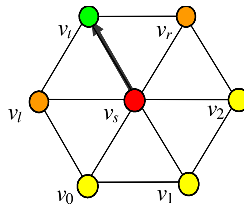


Fig. 4. A ring and its four sets of vertices: $S_1 = \{v_s\}$, $S_2 = \{v_t\}$, $S_3 = \{v_l, v_r\}$, and $S_4 = \{v_0, v_1, v_2\}$.

On the basis of the previously described vertex classification scheme, we have developed four strategies, i.e., Strategies I-IV, to provide four levels of dependency control. By arranging the range of the *dependent set* V_d, four strategies can be defined as follows:

Strategy I: $V_d = \{v \mid v \in S_1\}$; (7)

Strategy II: $V_d = \{v \mid v \in S_1 \cup S_2\}$; (8)

Strategy III: $V_d = \{v \mid v \in S_1 \cup S_2 \cup S_3\}$; (9)

Strategy IV: $V_d = \{v \mid v \in S_1 \cup S_2 \cup S_3 \cup S_4\}$. (10)

According to the contents of V_d, we define two types of dependent contractions —

tight dependent contractions, denoted as H_t, and *loose dependent contractions*, denoted as H_l, where

$$H_t = \{h_i \mid h_i: v_i \rightarrow v_j, \text{ for } v_i \in V_d\} \text{ and } H_l = \{h_i \mid h_i: v_i \rightarrow v_j, \text{ for } v_i \notin V_d \wedge v_j \in V_d\}. \quad (11)$$

In our implementation, we associate each vertex with a flag variable, called the *contraction flag*, to keep track of the state of the vertex and its ring. The flag values and their corresponding states are depicted in Table 1.

Table 1. Contraction flag values.

Contraction Type	Flag Value
Invalid (disable)	0
Valid (enable)	1
Dependent	2
Heavy	3

Prior to execution of the contraction, the contraction flags of the source and target vertices of the contraction are checked. If the contraction flag f_s^c of the source vertex v_s is not in the *valid* state $(f_s^c = 1)$, the contraction is considered to be a tight dependent contraction; on the other hand, if the contraction flag f_s^c of the source vertex v_s is in the *valid* state but the contraction flag f_t^c of the target vertex v_t is not, then the contraction is regarded as a loose dependent contraction. In general, both tight and loose dependent contractions are not executed. If lazy update [25] is employed, the loose dependent contractions are renewed and then executed instead of being discarded.

From experimental results, we find that the running times of the four strategies are ranked in increasing order as follows: Strategies I < II < III < IV. Furthermore, the outputs of simplification obtained by applying Strategy II have the smallest mean geometric error. Consequently, we adopted Strategy II in all of the other experiments.

In addition to removing the source vertex and the adjacent faces of the half-edge as well as replacing the index of the source vertex with that of the target vertex in non-adjacent faces of the half-edge, we set the contraction flags of the vertices in the ring according to the dependency control strategy in use. To carry out the half-edge collapse and set up contraction flags according to the specified dependency control strategy, the half-edge collapse routine is designed as follows:

Procedure HalfEdgeCollapse(h_s, *DCS*)
Input:
  HalfEdge h_s;
  unsigned integer *DCS*;
begin
1. Set the vertex flag of v_s to the *invalid* state $(f_s^v = 0)$.
2. Set the contraction flag of v_s to the *dependent* state $(f_s^c = 2)$.
3. If $DCS \geq 2$, set the contraction flag of v_t to the *dependent* state $(f_t^c = 2)$.
4. Find the set of adjacent faces of h_s: $A = r_s \cap r_i$ and let $B = r_x - A$.

5. For all f_i of A, where $f_i = \{v_s, v_t, v_j\}$ {
6. Set the face flag of f_i to the invalid state ($f_i^f = 0$).
7. If $DCS \geq 3$, set the contraction flag of v_j to the *dependent* state ($f_j^c = 2$). }
8. For all f_i of B, where $f_i = \{v_s, v_j, v_k\}$ {
9. Replace the index of v_s with that of v_t.
10. If $DCS \geq 4$, set the contraction flags of v_j and v_k to the *dependent* state ($f_j^c = f_k^c = 2$). }
11. $\boldsymbol{Q}_t = \boldsymbol{Q}_s + \boldsymbol{Q}_t$

end

2.1.6 The output stage

The input mesh is denoted as M_0. Assume n passes of simplifications are needed to generate the desired approximation. The first pass of simplification outputs a simplified mesh M_1 by simplifying the input mesh M_0. Then, the second pass outputs a simplified mesh M_2 by simplifying M_1. Thus, the i-th pass of simplification generates M_i. The set of output meshes together with the input mesh M_0, or $\{M_0, M_1, M_2, \ldots, M_n\}$, naturally forms a set of static-LODs of the input mesh M_0.

In the output stage, the remaining vertices and faces are output as a new LOD mesh following a compaction to the vertex and face caches if the static-LOD generation status bit is set; otherwise, only a compaction process that discards deleted vertices and faces is performed on the vertex and face caches.

2.2 The Cache Layer

The cache layer implements the fully associative cache algorithm discussed in [28], which acts as an intermediate layer between the algorithm layer and the data storage in order to improve disk access efficiency by reducing the number of external memory references. To provide shorter search time in cache block replacement, we use a version of the WSClock algorithm other than the least recently used (LRU) algorithm [29]. The cache layer comprises nine separate caches for each data structure shown in Table 2. The size of the cache is determined by the memory size limitation given by the command line parameter.

Since we only implement the previously proposed cache and replacement algorithms instead of developing a new one, we will not restate these algorithms in this paper. Readers who are interested in these algorithms can refer to the relevant textbooks or papers listed in the reference section.

The cache performance is related to at least two factors, i.e., the cache hit-ratio and the cache search time. The first factor is affected by the cache memory size, the mapping scheme, the replacement scheme, and so forth. On the other hand, the cache search time is proportional to the number of cache lines. In general, the fully associative cache has the highest cache hit-ratio but takes the longest cache search time.

To minimize the cache search time while maintaining a high cache-hit ratio, we set an upper bound on the number of cache lines and configure the cache parameters as follows.

Table 2. The external files and their associate element types used in our algorithm.

File	Element Type	Element Size	Number of Elements
Input mesh	VertexT	12	N^1
	FaceT	12	M^2
Vertex ring	VertexRingT	$(MAXDEG + 1)^3 \times 4$	N
Vertex quadric	QuadricT	44	N
Vertex flag	unsigned character	1	N
Face flag	unsigned character	1	M
Contrition flag	unsigned character	1	N
New vertex Index	unsigned integer	4	N
Contraction queue	ContractionRecordT	8	N

1. *N:* the number of vertices.
2. *M:* the number of faces.
3. *MAXDEG:* the upper bound of vertex connectivity.

Given n caches, let the number of cache lines, the cache block size, the element size, and the weight of the i-th cache, respectively, be cl_i, bs_i, es_i, and cw_i, for $i = 1, 2, \ldots, n$. We further assume that *CMS* is the individual cache size in number of elements and that *MemoryLimit* is the size of the main memory space allocated to the cache layer, where

$$CMS = \frac{MemoryLimit}{\sum_{i=1}^{n} cw_i \times es_i} . \qquad (12)$$

Let the memory cost of the algorithm be *MemoryCost*. In our implementation, the cache parameters, i.e., cs_i and bs_i for $i = 1, 2, \ldots, n$, are determined as follows:

If MemoryLimit ≥ MemoryCost,
  for $i = 1 \sim n$, $cl_i = 1$, and $bs_i = |V|$ or $|F|$.
else if *MemoryLimit* = 0,
  for $i = 1 \sim n$, $cl_i = 1$, and $bs_i = 1$.
else
for $i = 1 \sim n$,
  if $CMS \times cw_i \geq \text{MAXBS} \times \text{MAXCL}$, $cl_i = \text{MAXCL}$ and $bs_i = \text{MAXBS}$.
  else if $CMS \times cw_i \geq \text{MAXBS} \times \text{MINCL}$, $cl_i = \lfloor CMS \times cw_i/\text{MAXBS} \rfloor$ and
    $bs_i = \text{MAXBS}$.
  else if $CMS \times cw_i \geq \text{MINBS} \times \text{MINCL}$, $cl_i = \text{MINCL}$ and
    $bs_i = \lfloor CMS \times cw_i/\text{MINCL} \rfloor$.
  else $cl_i = bs_i = \left\lfloor \sqrt{CMS \times cw_i} \right\rfloor$.

To evaluate the performance of the cache layer, we conducted a number of experiments on the Dragon and Happy Buddha meshes. The results are shown in Fig. 5. According to Fig. 5, the running times of cached simplification were significantly lower than those without a cache.

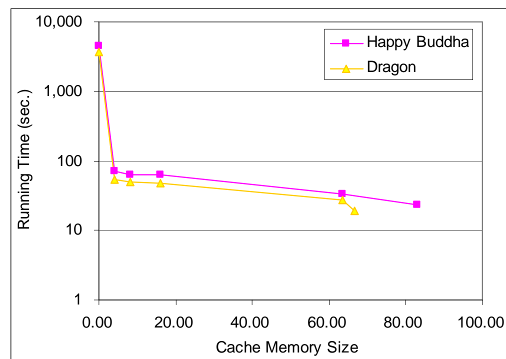


Fig. 5. The simplification times of CBPMS for the simplification of the Dragon and Happy Buddha meshes using 0-128 Mbytes cache buffers.

3. EXPERIMENTAL RESULTS

In this section, we will present the results of our experiments performed on a number of public domain meshes to evaluate the overall performance of the CBPMS algorithm. The test meshes used in our experiments were downloaded from Stanford 3D Scanning Repository of the Stanford University. The mesh size ranged from 871K to over 28 million triangular faces. A detailed list of the information associated with these meshes is given in Table 3.

Table 3. The test meshes

Model	Dragon	Happy Buddha	Lucy
Number of Vertices	437,645	543,652	14,027,872
Number of Faces	871,414	1,087,716	28,055,742
File size (Kbytes)	15,341	19,118	493,168

Prior to simplification, all these meshes were converted into a binary raw format that comprised a header with two unsigned integers to indicate the numbers of vertices and faces, a chunk of vertex coordinates in three floating-point numbers, and a chunk of triangular faces of a triplet of three vertex indices.

The CBPMS algorithm was implemented in C++ code and compiled with Microsoft Visual C++ V6.0. All the tests were performed on a personal computer equipped with an Intel Pentium 4 Celeron 2.4 GHz CPU, 1Gbyte of PC2100 DDR-DRAM, and an IDE RAID system with two 7,200 RPM, 2Mbyte buffer hard disks running the Microsoft Windows 2000 operating system.

The performance of a simplification algorithm can be evaluated in terms of the following: the memory cost, program running time, and output quality. With regard to the output quality, both the geometric errors and rendered images of the output meshes can be employed for numerical comparisons and visual examination. Since the source codes of most previous works are not available in the public domain, it is not easy to make a fair comparison between our works and the previous ones. Thus, here, we only compare

our work with the public domain software QSlim V2.0 provided by Garland [30] and an implementation of the OoCSx algorithm proposed by Lindstrom *et al*. [18].

3.1 The Memory Costs and Running Times

The new approach has two memory costs, i.e., the main memory and external memory costs. Regardless of the output space, the external memory cost amounts to $12n + 12m + (\text{MAXDEG} + 1) \times 4 \times n + 44n + n + m + 4n + 8n = 160n$ bytes according to Table 2. On the other hand, the main memory cost is determined by the amount of memory space pre-allocated for the cache buffers and the heap, which remains constant during simplification.

The memory costs as well as the running times of the CBPMS and OoCSx algorithms for simplification of the four test meshes are presented in Table 4. It should be noted that the running times of the OoCSx method presented in Table 4 include the preprocessing time involved in conversion from the index faced mesh format to the STL format and the dereferencing time necessary for converting from the STL format to the index faced format; thus, the running times are longer than those presented in [18].

Table 4. The memory costs and running times of the CBPMS algorithm.

Mesh name	Method	Output Size	Running times	Memory costs		Reduction rates
				RAM	DISK	
Dragon	OoCSx	1,214	18	36	141	49,067
	CBPMS[1]	1,000	22	20	67	38,854
Happy Buddha	OoCSx	6,008	23	36	176	46,801
	CBPMS[1]	1,000	28	20	83	38,756
Lucy	OoCSx	2,202	1,167	36	4,549	24,030
	CBPMS[1]	1,000	2647	20	2,140	10,597
1. MINCL = 20, MAXCL = 128, MINBS = 128, MAXBS = 4096.						

According to the results shown in Table 4, the reduction rates for the Dragon and Happy Buddha meshes are around 38.8K triangles per second, but the reduction rate decreases to about 10.6K triangles per second for the Lucy mesh. Apparently, the OoCSx approach performs faster than the CBPMS approach. However, considering the purpose of simplification, i.e., the generation of LOD meshes, the OoCSx approach can only output a single resolution mesh at a time, while the CBPMS is capable of creating a set of LOD meshes in a single run. To generate a set of LOD meshes, one has to run the OoCSx method iteratively with different grid resolutions. Hence, if a set of LOD meshes is needed rather than a single LOD mesh, the CBPMS approach will be more preferable than the OoCSx method. Furthermore, it takes less execution time to generate a high resolution output than to create a low resolution one with the CBPMS method, but the situation is reversed when the OoCSx method is adopted.

Regarding the memory costs, since the main memory costs of both methods are constant, the amount of disk space in use determines the winner. From Table 4, it is obvious

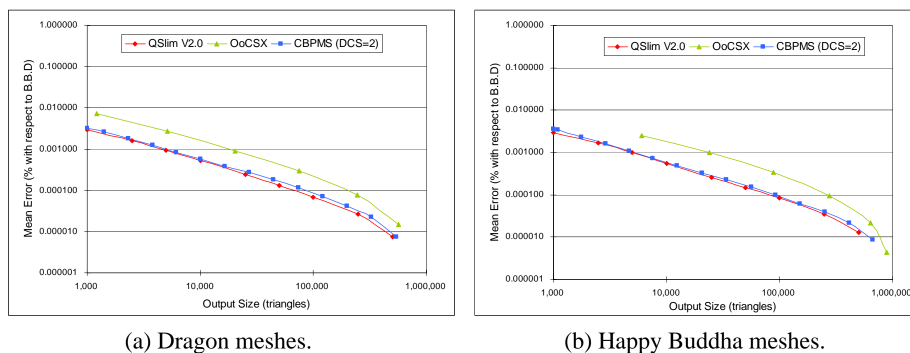
that the CBPMS method outperforms the OoCSx, since it uses much less disk space. From the above comparisons, we can conclude that our method is significantly better than OoCSx.

3.2 Evaluation of Output Quality

The quality of the output meshes obtained with our new approach is examined here based on their approximation errors and visual appearance. The outputs of two previous works, an in-core greedy-based full edge collapse based method (QSlim V2.0) [30] and an out-of-core uniform grid vertex re-sampling method (OoCSx) [18], are compared with the outputs of our method (CBPMS).

3.2.1 The quantitative approach

To measure the approximation errors of the output meshes, we used a famous public domain software program called Metro V3.1 [31]. However, owing to the limitations of QSlim and Metro, the comparisons presented here are restricted to model sizes under two million triangles. Thus, only the results of simplification of the Dragon and Happy Buddha meshes are provided in Fig. 6.



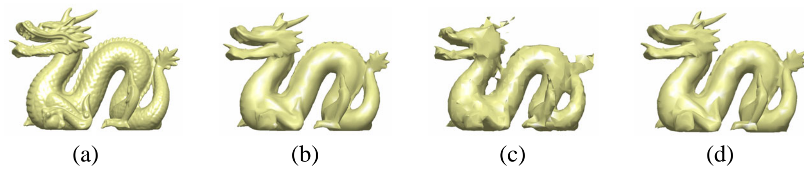
(a) Dragon meshes. (b) Happy Buddha meshes.

Fig. 6. The approximation errors of the simplified.

According to the results shown above, the mean geometric errors of the output meshes generated by our method (CBPMS) are very close to those generated by QSlim and are significantly lower than those created by the OoCSx method.

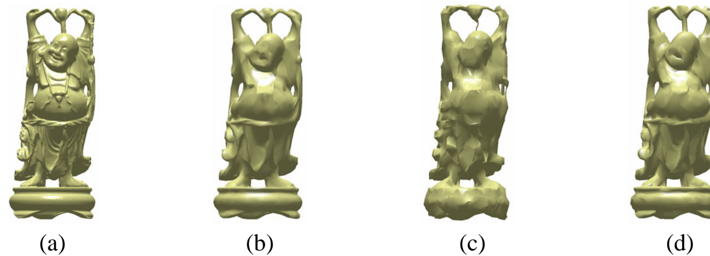
3.2.2 The visual approach

In addition to the approximation errors, another commonly used way to evaluate the output quality of a simplification algorithm is to visually examine the rendered images of its outputs. Figs. 7-9 show the rendered images of the original models as well as the simplified outputs of QSlim V2.0, OoCSx, and the CBPMS.



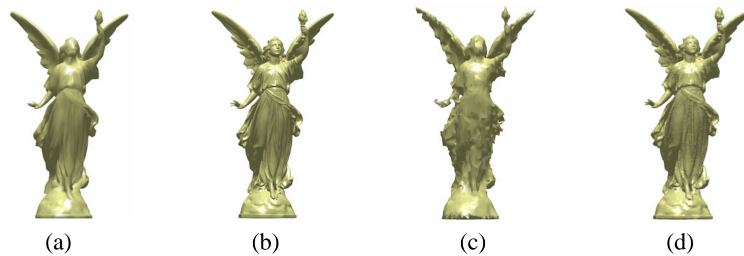
(a) (b) (c) (d)

Fig. 7. The rendered images of the Dragon meshes: (a) the original resolution and its simplified ones with 5,000 faces generated from: (b) the QSlim V2.0; (c) the OoCSx (5,086 faces, re-sampled using $32 \times 23 \times 15$ Grids); (d) the CBPMS (DCS = 2).



(a) (b) (c) (d)

Fig. 8. The rendered images of the Happy Buddha meshes: (a) the original resolution and its simplified ones with 5,000 faces generated from: (b) the QSlim V2.0; (c) the OoCSx (5,268 faces re-sampled using $15 \times 37 \times 16$ Grids); (d) the CBPMS (DCS = 2).



(a) (b) (c) (d)

Fig. 9. The rendered images of the simplified Lucy meshes generated by the CBPMS (DCS = 2): (a) 5,728 triangles (0.02%); (b) 317,826 triangles (1.13%); and the OoCSx: (c) 8,970 triangles (0.03%); (d) 53,800 triangles (1.92%).

Owing to the limitation imposed by the large main memory cost, it is not possible to create a simplified Lucy mesh using the QSlim V2.0. Therefore, we only show two approximations of the Lucy mesh generated by the OoCSx and CBPMS methods in Fig. 9. From the rendered images shown in Figs. 7-8, it is clear that the quality of the output meshes generated by our method are nearly as good as that of the meshes created by QSlim V2.0 and are significantly better than that of the meshes created by the OoCSx method.

4. CONCLUSIONS

In this paper, we have proposed a novel cache-based polygonal mesh simplification (CBPMS) algorithm that uses iterative half-edge collapses and quadric error metrics

based on a framework similar to the independent queuing. Essentially, it is an external memory version of traditional iterative edge collapse based PMS algorithm with the following modifications. First, instead of applying traditional strategy, we invent a set of more flexible strategies for the removal of dependent contractions. Second, , a small constant sized heap is built inside the main memory space rather than a large input-dependent sized greedy queue in external memory space for better locality of references and thus better run-time efficiency. Third, a cache layer is implemented to improve external memory accesses efficiency. Different from the previous related works [12, 18, 20, 21], our new approach accepts common index faced meshes instead of the STL formatted meshes and does not require pre-sorting of the input sequence. Furthermore, the main memory cost of our new approach is not only independent of the input or output mesh size but also constant and scalable [12, 19-22, 24].

According to the results presented in this paper, we can conclude that our new approach, CBPMS, has at least four attractive features. First, the main memory cost is constant. By placing the data structures in the external memory space and building a constant sized heap rather than a large one in external memory, the CBPMS algorithm requires only a fixed amount of main memory space for the cache buffers and the heap. Second, the use of a cache layer enables us to design the simplification layer regardless of the main memory size. This feature could be introduced into most of the previously proposed algorithms. Third, the CBPMS algorithm is insensitive to both the input and the output mesh sizes; hence, theoretically, it is capable of simplifying arbitrarily large meshes regardless of the constraints set by the disk space and the numeric system. According to the presented experimental results, our new approach is capable of simplifying a very large mesh that comprises over 28 million triangles. The last feature of CBPMS is that it is capable of generating very high quality approximations that approach the quality of the outputs of traditional in-core simplification algorithms (QSlim V2.0).

From the experiments, we have noticed that the reduction rate for simplifying the Lucy mesh is lower than those for the Dragon and Happy Buddha meshes. Hence, a further improvement on the reduction rate may be achieved. To address this issue, we suggest two possible improvements to CBPMS. The first involves improving the locality of the references of the input sequence by applying a preprocessing stage, such as an external spatial sort. The second improvement could be made in the simplification layer, where we may further integrate the operations to yield better runtime efficiency and reference locality.

ACKNOWLEDGMENTS

We would like to thank the Stanford 3D Scanning Repository for providing all the test models, Michael Garland et al. for providing QSlim V2.0 [30], and Paolo Cignoni et al. for providing Metro V3.1 [31] for use in the experiments.

REFERENCES

1. M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk, "The digital michelangelo

project: 3D scanning of large statues," in *Proceedings of the ACM SIGGRAPH*, Vol. 34, 2000, pp. 131-144.
2. J. Trimble and M. Levoy, "Stanford digital forma Urbis Romae project," http:// formaurbis.stanford.edu/docs/FURproject.html, 2002.
3. M. J. Ackerman, "The visible human project," *Proceedings of IEEE*, Vol. 86, 1998, pp. 504-511.
4. H. Hoppe, "Progressive meshes," in *Proceedings of the ACM SIGGRAPH*, Vol. 30, 1996, pp. 99-108.
5. M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the ACM SIGGRAPH*, Vol. 30, 1996, pp. 209-216.
6. M. Garland and Y. Zhou. "Quadric-based simplification in any dimension," to appear in *ACM Transactions on Graphics*, Vol. 24, 2005, pp. 209-239.
7. P. Lindstrom and G. Turk, "Evaluation of memoryless simplification," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 5, 1999, pp. 98-115.
8. L. Kobbelt, S. Campagna, and H. P. Seidel, "A general framework for mesh decimation," in *Proceedings of Graphics Interface*, 1998, pp. 43-50.
9. W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of triangle meshes," in *Proceedings of the ACM SIGGRAPH*, Vol. 26, 1992, pp. 65-70.
10. J. Rossignac and P. Borrel, "Multi-resolution 3D approximation for rendering complex scenes," in *Proceedings of 2nd Conference on Geometric Modeling in Computer Graphics*, 1993, pp. 455-465.
11. K. L. Low and T. S. Tan, "Model simplification using vertex clustering," in *Proceedings of the ACM Symposium on Interactive 3D Graphics*, 1997, pp. 75-82.
12. P. Lindstrom, "Out-of-core simplification of large polygonal models," in *Proceedings of the ACMSIGGRAPH*, Vol. 34, 2000, pp. 259-270.
13. C. Prince, "Progressive meshes for large models of arbitrary topology," Master Thesis, Dept. of Computer Science and Engineering, University of Washington, Seattle, 2000.
14. E. Shaffer and M. Garland, "Efficient adaptive simplification of massive meshes," in *Proceedings of the IEEE Visualization*, 2001, pp. 127-134.
15. H. Hoppe, "Smooth view-dependent level-of-detail control and its application to terrain rendering," in *Proceedings of the IEEE Visualization*, 1998, pp. 35-42.
16. J. El-Sana and Y. J. Chiang, "External memory view-dependent simplification," *Computer Graphics Forum*, Vol. 19, 2000, pp. 139-150.
17. P. Choudhury and B. Watson, "Completely adaptive simplification of massive meshes," Technical Report CS-02-09, Dept. of Computer Science, Northwestern University, Evanston, Illinois, 2002.
18. P. Lindstrom and C. T. Silva, "A memory insensitive technique for large model simplification," in *Proceedings of the IEEE Visualization*, 2001, pp. 121-126.
19. P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno, "External memory management and simplification of huge meshes," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 9, 2003, pp. 525-537.
20. J. Wu and L. Kobbelt, "A stream algorithm for the decimation of massive meshes," in *Proceedings of Graphics Interface*, 2003, pp. 185-192.
21. M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink, "Large mesh simplification using processing sequences," in *Proceedings of the IEEE Visualization*, 2003,

pp. 65-472.
22. E. Shaffer and M. Garland, "A multiresolution representation for massive meshes," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 11, 2005, pp. 139-148.
23. D. Brodsky and B. Watson, "Model simplification through refinement," in *Proceedings of Graphics Interface*, 2000, pp. 221-228.
24. M. Isenburg, S. Gumhold, and J. Snoeyink, "Processing sequences: a new paradigm for out-of-core processing on large meshes," preprint available at http://www.cs.unc.edu/˜isenburg/oocc/, 2003.
25. D. Lubke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner, *Level-of-Detail for 3D Graphics*, Morgan Kaufman, San Francisco, California, 2003.
26. D. Dobkin and D. Kirkpatrick, "A linear algorithm for determining the separation of convex polyhedra," *Journal of Algorithms*, Vol. 6, 1985, pp. 381-392.
27. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, New Jersey, 1973.
28. J. Hennessy and D. A. Patterson, *Computer Architecture – A Quantitative Approach*, Morgan Kaufman, San Francisco, California, 1990.
29. R. W. Carr and J. L. Hennessy, "WSClock – a simple and effective algorithm for virtual memory management," in *Proceedings of 8th Symposium on Operating Systems Principles*, 1981, pp. 87-95.
30. M. Garland and P. S. Heckbert, "QSlim V2.0 simplification software," Dept. of Computer Science, University of Illinois, Urbana-Champaign, Illinois, http:// graphics.cs.uiuc.edu/~garland/software/QSlim.html, 1999.
31. P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: measuring error on simplified surfaces," *Computer Graphics Forum*, Vol. 17, 1998, pp. 167-174.

Hung-Kuang Chen (陳宏光) received both his M.S. and Ph.D. degrees from the Department of Electronic Engineering of the National Taiwan University of Science and Technology, Taipei, Taiwan, in 1995 and 2006, respectively. Through 1995 to 2002, he had served as a Lecturer in the Department of Electronic Engineering of the Lunghwa University of Science and Technology. Since 2002, he has been on the faculty of the Department of Information and Design of Asia University. His research interests include geometric modeling, computer graphics, 3D computer games, virtual reality, and parallel computing.

Chin-Shyurng Fahn (范欽雄) received the B.S. degree in Electronic Engineering from National Taiwan Ocean University, Keelung, Taiwan, in 1981, and the M.S. and Ph.D. degrees, both in Electrical Engineering, from National Cheng Kung University, Tainan, Taiwan, in 1983 and 1989, respectively. From August 1983 to July 1984, he served as a Teaching Assistant in the Dept. of Electrical Engineering at National Cheng Kung University.

From August 1983 to July 1989, he worked as an Adjunct Research Assistant in the Institute of Electrical Engineering of that university. From November 1989 to May 1991, he served in the Chinese Army as a Signal Officer. In the meantime, he served as a Lecturer in the Department of Mechanical Engineering of the Chinese Army Engineering School, Yenchao, Kaohsiung, Taiwan. From August 1991 to July 2002, he served as an Associate Professor in the Department of Electrical Engineering of the National Taiwan University of Science and Technology, Taipei, Taiwan. Since August 2002, he has been on the faculty of the Department of Computer Science and Information Engineering of the same university. His current fields of interest include fuzzy systems, neural networks, and evolutionary computing applied in the areas of image processing, pattern recognition, computer vision, computer graphics, and virtual reality.

Jeffrey J. P. Tsai (蔡進發) received his Ph.D. degree in Computer Science from Northwestern University, Evanston, Illinois. He is a Professor in the Department of Electrical Engineering and Computer Science at the University of Illinois at Chicago, where he is also the Director of the Distributed Real-Time Intelligent Systems Laboratory. He co-authored Knowledge-Based Software Development for Real-Time Distributed Systems (World Scientific, 1993), Distributed Real-Time Systems: Monitoring, Visualization, Debugging, and Analysis (John Wiley and Sons, Inc., 1996), Compositional Verification of Concurrent and Real- Time Systems (Kluwer, 2002), co-edited Monitoring and Debugging Distributed Real- Time Systems (IEEE/CS Press, 1995), and has published over 160 papers in the areas of knowledge-based software engineering, software architecture, requirements engineering, formal methods, agent-based systems, and distributed real-time systems. Dr. Tsai was the recipient of the University Scholar Award from the University of Illinois in 1994 and the Technical Achievement Award from the IEEE Computer Society in 1997. He is the Co-Editor-in-Chief of the International Journal of Artificial Intelligence Tools and on the editorial board of the International Journal of Software Engineering and Knowledge Engineering, and chairs the IEEE/CS Technical Committee on Multimedia. He is a Fellow of the IEEE, the AAAS, and the SDPS.

Ming-Bo Lin (林銘波) received the B.Sc. degree in Electronic Engineering from the National Taiwan University of Science and Technology, Taipei, the M.Sc. degree in Electrical Engineering from National Taiwan University, Taipei, and the Ph.D. degree in Electrical Engineering from the University of Maryland, College Park. Since February 2001, he has been a Professor in the Department of Electronic Engineering at the National Taiwan University of Science and Technology, Taipei, Taiwan. His research interests include VLSI systems design, mixed-signal integrated circuit designs, parallel architectures and algorithms, and computer arithmetic.