# Time-Critical Rendering of Tetrahedral Meshes

CHUAN-KAI YANG AND TZI-CKER CHIUEH*
*Department of Information Management*
*National Taiwan University of Science and Technology*
*Taipei, 106 Taiwan*
**Department of Computer Science*
*Stony Brook University*
*Stony Brook, NY 11770, USA*

Very large irregular-grid volume datasets are typically represented as tetrahedral meshes and require substantial disk I/O and rendering computation. One effective way to reduce this demanding resource requirement is compression. Previous research showed how rendering and decompression of a losslessly compressed irregular-grid dataset can be integrated into a one-pass computation. This work advances the state of the art one step further by showing that a losslessly compressed irregular volume dataset can be simplified while it is being decompressed and that simplification, decompression, and rendering can again be integrated into a pipeline that requires only a single pass through the datasets. In particular, this rendering pipeline can exploit a multi-resolution representation to maintain interactivity on a given hardware/software platform by automatically adjusting the amount of rendering computation that could be afforded, or performing so called time-critical rendering. As a proof of the feasibility of our proposed framework, we modify an existing tetrahedral mesh simplification algorithm and integrate it with our previous volume decompression and rendering pipeline. Performance measurements on this prototype system show that simplification only adds less than 5% of performance overhead on an average; furthermore, with multi-resolution pre-simplification the end-to-end rendering delay indeed decreases in an approximately linear fashion with respect to the simplification ratio, thus a time-critical rendering of large tetrahedral mesh could be achieved.

***Keywords: Irregular Grids, Tetrahedral Mesh Compression, Volume Simplification, Volume Rendering, Time-Critical Rendering***

## 1. INTRODUCTION

Our previous paper [1] showed that it is possible to pipeline the decompression of a losslessly compressed tetrahedral mesh and the rendering of the resulting tetrahedra, thus significantly reducing the memory footprint requirements of rendering tasks whose target datasets are close to or larger than the rendering machine's physical memory. This performance advantage comes from the fact that volume decompression and volume rendering are integrated into a one-pass computation.

To further reduce the rendering delay associated with very large irregular volume datasets, one needs to trade accuracy for performance. Given a rendering hardware and a pre-defined level of interactivity, the goal is to develop a rendering algorithm that can meet the performance requirements while maintaining the highest rendering image quality. The enabling technology that allows making such a tradeoff is tetrahedral mesh simplification. Unfortunately, most existing mesh simplification algorithms are implemented as a stand-alone tool rather than as a tightly integrated component of an irregulargrid volume renderer, thus limiting their utility as a dynamic performance adaptation mechanism.

The main contribution of this paper is twofold. First, it is not on the invention of any new simplification algorithm for irregular volume data, but rather on proposing a framework that could integrate any simplification algorithm into the system pipeline should it satisfy certain assumptions (described more thoroughly in Section 3), which happen to be assumed by most existing algorithms. We demonstrate how an existing mesh simplification algorithm could be readily modified to fit nicely into a decompression-driven volume renderer, thus making it possible to integrate volume decompression, simplification, and rendering into a seamless pipeline that requires only one pass through the compressed input dataset. Because of this streamlined structure, the volume renderer can dynamically adjust the rendering accuracy to match the user-specified interactivity requirement and/or computation resource availability. Second and more importantly, simplification and lossless compression together make it possible to represent a tetrahedral mesh as a multi-resolution pre-simplification hierarchy, with the losslessly compressed version corresponding to the finest resolution. As a result, the proposed integrated decompression / simplification / rendering engine, not only greatly reduces the runtime disk access overhead and peak memory usage during the rendering stage, as in the previous integrated volume decompresser/renderer[1], but also enables the time-critical rendering of very large tetrahedral meshes

We have successfully implemented the integrated *decompression / simplification / rendering pipeline* in our system, called Gatun, as it is also a name of locks for the Panama Canal, whose operation inherently follows a pipeline structure. Our performance measurements on the Gatun prototype show that the proposed mesh simplification algorithm only adds less than 5% overhead on an average compared to an integrated volume decompresser/renderer, thus demonstrating the efficiency of the proposal's implementation simplicity. On the other hand, the proposed mesh simplification, together with our pre-simplification multiresolution scheme, can effectively reduce the total rendering time by a factor of up to 2 when 90% of the input mesh is simplified away.

The rest of this paper is organized as follows. Due to the limit of space, we only review related work on tetrahedral mesh simplification in Section 2. In Section 3, we describe the on-the- fly mesh simplification algorithm that is tightly integrated with the mesh decompresser and renderer. In Section 4, we demonstrate how to apply the simplification-capable integrated renderer to time-critical rendering. Section 5 reports the performance measurements of the proposed integrated mesh decompression / simplification / rendering pipeline on the Gatun prototype for six irregular-grid datasets with the number of tetrahedra ranging from 1.3K to 1.2M. Section 6 concludes this paper by summarizing the main research contribution of this work.

## 2. RELATED WORK

There are a few works on polygonal surface mesh simplification. These methods simplify a 3D surface model by merging neighboring polygons into one if they are "flat" enough [2–4], by re-tiling the triangular mesh through inserting vertices at places with maximal curvature and iteratively remove the old vertices [5], by clustering vertices to obtain a possibly topologically different mesh [6], by performing a wavelet-based multi-resolution analysis of the input mesh and re-meshing/re-sampling the mesh if necessary [7], by optimizing the input mesh through addition and/or removal of vertices or collapsing/swapping edges to minimize a certain pre-defined energy function that depends on the mesh [8], and by simply collapsing edges to form progressive meshes [9]. The last algorithm, in particular, provides a general framework for building view-independent multiresolution representation of a 3D surface. It uses an error metric that is defined for each simplification primitive (vertex, triangle, tetrahedron, etc.). At run time, the algorithm first computes the error metric for each simplification primitive that is applicable to the input triangular mesh, and then builds a priority queue for these primitives according to their error metrics. The simplification process starts with the primitive with the smallest error metric, collapses edges or patches the resulting mesh, re-computes the error metric for all the simplification primitives that are affected, and finds the next primitive with the smallest error metric, etc.

There is another line of research [10–12] in this area that focuses on view-dependent simplification, which is able to adapt the extent of simplification for each geometric primitive to view angles and other run-time parameters, and some can also provide error control mechanism. In general this class of algorithms first statically build a hierarchical data structure for the input 3D model and at run time use the viewing angle information to compute simplification parameters and efficiently locate the necessary part of the hierarchical data structure that is to be used for rendering. Some research extended this line of work to support the out-of-core view-dependent simplification as well [13].

Volume simplification, on the other hand, attracts relatively less research effort. Cignoni et al. [14] proposed decimating vertices iteratively and the resultant holes are retetrahedralized locally. Later they proposed another method which estimates the quality of simplification by utilizing two types of error, domain error and field error, with the first one referring to the deviation from the the original geometry while the second one the difference with the scalar fields defined over the original mesh [15]. Renze et al. [16] generalized the idea of progressive meshes to perform volume decimation for unstructured grids. Staadt et al. [17] proposed techniques for progressive tetrahedralization which can avoid some artifacts such as self-intersections due to an improper simplification. Trotts et al. [18] applied a piece-wise linear spline function that is defined over the scalar values over the input tetrahedral mesh as the basis of the error metric for volume simplification. The algorithm associates each tetrahedron with such an error metric and favors the removal of those tetrahedra that causes the least change in the spline function. Removing a tetrahedron is carried out by a sequence of edge collapses. The TetFusion approach, proposed by Chopra et al. [19], performs the simplification by shrinking a central tetrahedron towards its geometric center, and one such central tetrahedron's collapse will cause the degeneration of at least eleven tetrahedra. The reduction rate is fast, however, the dealing with boundary tetrahedra is tedious and thus is avoided, leading to low compression ratio. Furthermore, the error estimation scheme is not yet established. Co et al. [20] proposed a method that treats unstructured grids as point cloud and represents such grids by hierarchical clustering. The hierarchy is generated by applying the PCA (principle component analysis) for cluster generation and a simplified RBF (radial basis functions) for fitting the scalar fields. At the run time, the traversal of the cluster hierarchy could be level-based, or error-based. Gelder et al. [21] proposed a less computation-intensive approach for simplification, which aims to minimize the density or "mass" change due to an edge collapse. Boundary vertices come with extra geometry-related error metric, in addition to the so called "data-based" error metric required of internal vertices. The on-the-fly simplification scheme presented in this paper is based on this work.

In terms of integrated work, besides Gatun, volume compression and rendering have also been integrated in the work by Schneider et al. [22]. Through the use of vector quantization, their work has the elegance of trading fidelity for performance in a more unified framework. However, their work is designed only for dealing with regular grids and cannot be readily applied to unstructured grids. Probably the work from Farias et al. [23] is by far the most similar one to us. However, in their work, they applied basically the vertex clustering idea from [6] where the topology of a mesh may be changed. Gatun applies a simplification approach which can not only preserve the topology of a mesh but also makes it easier to be pipelined with the decompression process. In addition, Gatun uses an object space-based ray casting approach. Because of ray casting, the resulting rendered image quality is high. Also because of the object space architecture, rendering can be done incrementally and thus can be nicely tied with the mesh decompression process.

## 3. ON-THE-FLY TETRAHEDRAL MESH SIMPLIFICATION

To incorporate mesh simplification in the integrated rendering/decompression pipeline described [1], Gatun first statically computes a priority list of volume simplification operations, and at run time performs a selective subset of these vertex merge (in this paper, the term vertex merge is interchangeable with edge collapsing) operations based on user requirements and/or available computation resources. The simplification step is inserted between decompression and rendering in a way that is largely independent of the internal working of the renderer and decompresser.

### 3.1 Static Simplification Algorithm

Gatun makes the following assumptions on the volume simplification algorithm:

– Vertex merge is the only tetrahedral mesh simplification primitive used in the algorithm, and each vertex merge operation is denoted as $V_i \rightarrow V_j$ , which means that $V_i$ is merged into $V_j$ , and

– The algorithm can statically compute an error metric for all possible vertex merge operations, and derive a priority order among them based on this error metric.

The volume decimation algorithm [21] described in this subsection is one example of such volume simplification algorithm. *Gatun can inter-operate with any other simplification algorithms as long as they satisfy the above assumptions*. We re-implement [21]'s algorithm for our testing volume simplification algorithm.

This volume decimation algorithm takes into account two types of errors introduced by simplification: density-related ($Error_{density}$) and geometry-related ($Error_{geometry}$), and the final error metric associated with a vertex merge operation is

$$W_{density} * Error_{density} + W_{geometry} * Error_{geometry} \tag{5}$$

where $W_{density}$ and $W_{geometry}$ are weighting parameters that are tailored to the needs of individual application.

After computing the error metric for all neighboring vertex pairs, the volume decimation algorithm selects the vertex merge operation with the smallest error metric, say $V_n \rightarrow V_m$, eliminate all vertex merge operations of the form $V_n \rightarrow V_k$ for some $k$ from further consideration, re-computes the error metric of those vertex merge operations that are affected by the application of $V_n \rightarrow V_m$, and repeats the cycle by picking the one with the smallest error metric from the remaining vertex merge operations, etc. After a vertex merge operation, the geometry of the affected region of tetrahedral mesh is changed. Consequently, the error metric of those vertex merge operations associated with the affected region needs to be recomputed. More specifically, after the application of a vertex merge operation $V_i \rightarrow V_j$ , the error metric of all vertex merge operations of the form $V_k \rightarrow V_i$ needs to be recomputed based on the new geometry. Eventually the algorithm ends when all vertex merge operations have been eliminated. The list of selected vertex merge operations are ranked in an ascending order according to their error metric value, the smaller the error metric value is, the earlier a merge operation is performed. The rank value associated with a merge operation represents the order this operation is performed. For example, a operation with rank value 3 means it is the third operation.

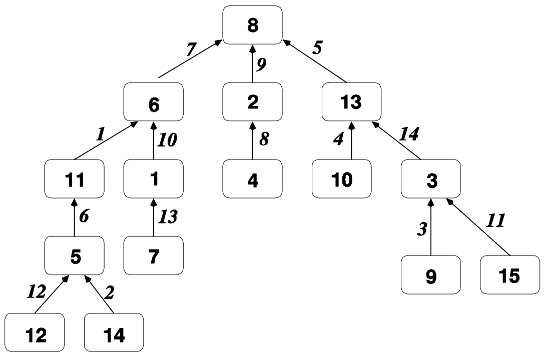


Fig. 1 An example merge tree that shows all the vertex merge operations, as represented as edges, that are being considered, and their relative priorities, as indicated by the weights on the edges.

A vertex that never needs to be merged into any other vertex is called an independent vertex. After applying the above volume decimation algorithm to a tetrahedral mesh, every vertex is scheduled to be merged into some other vertex at a certain priority except one or multiple independent vertices. Gatun organizes the list of resulting vertex merge operations into a forest of multiple trees (called a merge tree), each of whose root is an independent vertex. In these trees, each child vertex is to be merged to its parent vertex. In addition, every child vertex has a global rank that represents the priority of the corresponding vertex merge operation. Figure 1 shows an example merge tree for a hypothetic tetrahedral mesh. Each node in the tree represents a vertex in the mesh, and each edge represents a vertex merge operation. The weight on an edge represents the global rank of the edge's associated vertex operation. For example, the operation of merging Vertex 1 into Vertex 6 has a rank of 10, or equivalently, this is the 10th operation. At run time, if users ask the system to perform a simplification step that includes only the first 5 vertex merge operations, this operation will be ignored.

**3.2 Run-Time Simplification Algorithm**

Because the volume decimation algorithm already computes a global rank for all selected vertex merge operations, at run time Gatun only needs to perform the first N of these operations, where N is determined either by users or by the system based on available computation resource. Therefore whether a vertex merge operation needs to be performed for a given N is a local decision. This localness property makes it possible to perform on-the-fly simplification on the resulting tetrahedra stream that the decompresser produces. The only question left is how to effect each eligible vertex merge operation as a compressed tetrahedral mesh is being decompressed. The key insight behind inserting an on-the-fly volume simplification step between decompression and rendering is that all the renderer wants is that all the tetrahedra it uses during the rendering computation are valid tetrahedra that exist in the final mesh after all the selected simplification operations have been applied to the original mesh. That is, as long as every tetrahedron the simplification step passes to the renderer is a valid tetrahedron in the final mesh the rendering result of this integrated decompression/simplification/rendering pipeline is guaranteed to be correct.

Whenever a tetrahedron is decompressed, the renderer in the integrated decompression/rendering engine described in [1] advances each cast ray that intersects with the tetrahedron as much as possible, and then stops to wait for more tetrahedra to come so that these rays can move further ahead. As far as the renderer is concerned, it does not care about where and how the tetrahedra are generated. Without simplification, the decompresser outputs all the tetrahedra in the input mesh; with simplification, the simplification module outputs only those tetrahedra that exist in the simplified mesh. Therefore, whenever a tetrahedron from the decompresser arrives, the simplification module needs to check whether the tetrahedron is collapsed after simplification and, if it is not, whether the tetrahedron's vertices change because of vertex merging. Only when a tetrahedron is not going to be collapsed, and the ultimate target vertices of the tetrahedron after application of all vertex merge operations already appear in the decompresser's output stream can the simplification module forward this tetrahedron to the renderer. Because such tetrahedra are valid in the simplified mesh and their data density and coordinate are known, the renderer can safely perform rendering computation based on them and produce provably correct simplified results.

Because what is needed is the set of tetrahedra in the final simplified mesh, one needs an efficient algorithm to summarize the accumulative effect of a set of chosen vertex merge operations. More concretely, one needs to compute the ultimate target vertex into which each vertex is to be merged, based on the input mesh's merge tree data structure and a selective set of verge merge operations. With this per-vertex information, one can easily check whether a given tetrahedron from the decompresser is valid in the final mesh or not, or if the decompresser has traversed all its ultimate target vertices.

The on-the-fly simplification algorithm of Gatun therefore consists of two steps: one before and one during the decompression/rendering process. First, given a threshold N, the on-the-fly simplification algorithm performs a top-down traversal of the input mesh's merge tree and determine which vertex merge operations are eligible by comparing the vertices' rank with N. For each eligible vertex merge operation, which corresponds to an edge in the merge tree whose child vertex's rank is smaller than or equal to N, the child vertex's ancestor field is filled with its parent vertex's ancestor field. For each non-eligible vertex merge operation, which corresponds to an edge in the merge tree whose child vertex's rank is larger than N, the child vertex's ancestor field is filled with its own ID(rank). A root vertex's ancestor field is always filled with its own ID. After this traversal, the ultimate target of each vertex is going to be merged into is kept in the vertex's ancestor field. If the ancestor field points to itself, the vertex does not get merged into anyone, and is present in the final mesh. For example, if the simplification threshold N is set to 8 for the merge tree in Figure 1, then after simplification, or equivalently after the first 8 vertex merge operations are done, Vertex 14's ancestor is Vertex 8, Vertex 9's ancestor is Vertex 3, etc., and Vertex 1, 2, 7, 3, 8, 12, and 15 are present in the final simplified mesh, as shown in Figure 2.

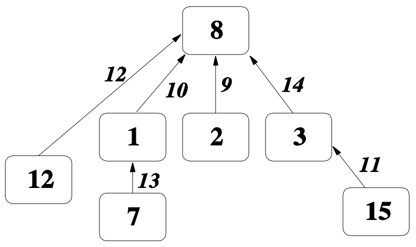


Fig. 2 The resulting mesh after the first 8 merges are carried out.

After decompression/rendering starts, whenever the decompresser passes a tetrahedron, Gatun first checks whether the ancestors of the tetrahedron's four vertices are distinct. If they are not distinct, i.e., multiple vertices of the tetrahedron collapsed into one, the tetrahedron is not a valid one after all vertex merge operations and thus should be discarded. If they are distinct, it is a valid tetrahedron. However, a valid tetrahedron may not be usable yet because not all its vertices' ancestors have already appeared in the vertex stream that the decompresser outputs at that point. A tetrahedron can be forwarded to the rendered if and only if it is both valid and usable. When a non-usable tetrahedron first appears, it is attached to all the ancestor vertices that it waits for (the vertex table data structure), and its wait_count field is initialized to the number of ancestors that have yet to appear (in the tetrahedron table data structure). Every time the decompresser enumerates a new vertex, it checks whether there are any tetrahedra waiting for the new vertex, if so it decrements the wait_count field of each such non-usable tetrahedron by one, and forwards to the renderer those tetrahedra whose wait-count field reaches zero.

The on-the-fly volume renderer described in [1] needs to identify the input tetrahedral mesh's boundary surface first so that it can compute the intersections between the cast rays and boundary faces. However, the boundary surface may also be affected by the simplification step. For the boundary surface of an input tetrahedral mesh, Gatun performs a similar usability check on each boundary face, based on whether the ancestors of each face's three vertices are distinct. Only after all valid boundary faces have been completely identified can the renderer-cast rays be properly attached to the boundary surface. Only after successful attachment of all cast rays can the renderer advance the rays into the simplified tetrahedral mesh to interpolate/composite the data density values at sampling points.

The time complexity for the run-time simplification alone is linear, in terms of the number of vertices in a dataset, and the reasons are the following. First, the top-down traversal of the merge tree needs to be done only once. Second, the checking for waiting tetrahedra and boundary faces is also with linear time complexity, as each tetrahedron contains four vertices and each face three vertices.

## 4. TIME-CRITICAL RENDERING

The original motivation for incorporating volume simplification into an integrated pipeline is to apply it at run time to trade quality for performance, or more specifically to time-critical rendering, where the goal is to maximize the rendering quality for a fixed timing budget by simplifying the input mesh appropriately. Although the integrated decompression/simplification/rendering pipeline described in the last section makes a good starting point, it itself is not sufficient to support time-critical rendering, because *the decompression overhead dominates the end-to-end user-perceived delay regardless of the extent of mesh simplification* and thus the reduction in rendering time. In other words, while simplification does decrease the rendering time, it does not affect the end-to-end delay that much because *the system still needs to decompress the entire input dataset* and simplify it to a chosen level. To address this issue, we pre-compute multiple simplified versions of each input dataset, each corresponding to a particular simplification ratio, which is defined as the number of vertices that are simplified away divided by the number of all vertices. As shown in Figure 3, an input tetrahedral mesh is pre-simplified at the simplification ratios of $1-1/2^i$, and all these simplified versions are independently compressed and stored to the disk. The maximal value of i depends on the available system resource and performance requirements.

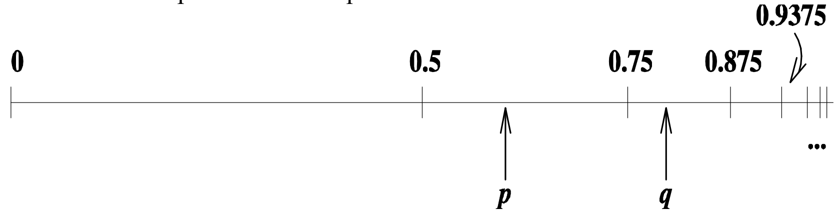


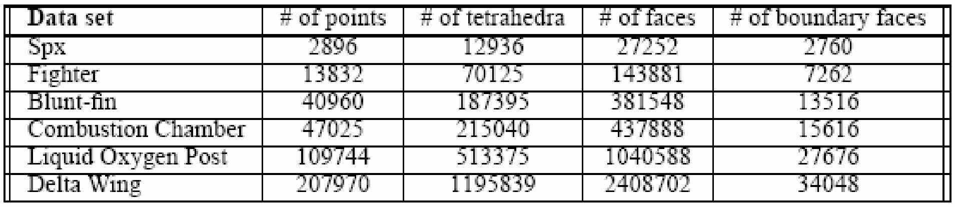
Fig. 3 Given a target simplification ratio, choose the most simplified version of a multi-resolution input mesh that is finer than the target as the starting mesh for simplification.

At run time, given a rendering time budget, the system first maps it to the corresponding simplification ratio, which may be different for different datasets, and then applies simplification to the pre-computed version of the input data whose simplification ratio is the largest among those whose simplification ratio is smaller than the target simplification ratio. For example, in this Figure, the version with the simplification ratio of 0.5 is selected as the starting mesh for simplification if the target simplification ratio is p, whereas the version with the simplification ratio of 0.75 is selected if the target simplification ratio is q. The basic idea of this approach is similar to mip-mapping[25], which could be used to speed up texture mapping. Note that although the disk storage cost of this scheme is doubled, the run-time performance is improved significantly as shown in the next section, because both decompression and rendering overheads are now about inversely proportional to the simplification ratio.

## 5 PERFORMANCE EVALUATIONS

The input datasets used for the performance evaluation of Gatun are listed in Table 1, as ordered by the number of tetrahedra. While the first two datasets are unstructured grids, the remaining four are curvilinear grids converted into tetrahedral grids.

**Table 1 Characteristics of input datasets used in this performance study.**



| Data set | # of points | # of tetrahedra | # of faces | # of boundary faces |
|---|---|---|---|---|
| Spx | 2896 | 12936 | 27252 | 2760 |
| Fighter | 13832 | 70125 | 143881 | 7262 |
| Blunt-fin | 40960 | 187395 | 381548 | 13516 |
| Combustion Chamber | 47025 | 215040 | 437888 | 15616 |
| Liquid Oxygen Post | 109744 | 513375 | 1040588 | 27676 |
| Delta Wing | 207970 | 1195839 | 2408702 | 34048 |

The compression efficiency (2.5 bits/tetrahedron, connectivity only) remains the same as reported previously, because we use the same tetrahedral compression algorithm. In addition, the peak memory usage saving (50% to 70%) also carries over when simplification is integrated into the pipeline, because the additional memory space requirement introduced by simplification is minimal, especially when compared with the rendering stage, whose memory footprint requirement is always dominant. The overall reduction of memory footprint in Gatun not only speeds up the rendering process by one to two orders of magnitude when input datasets are too large to fit into physical memory, but also shortens the perceived rendering delay when input datasets are completely memory-resident.

### 5.1 Simplification Overhead

Run-time simplification consists of two steps: a top-down traversal of the merge tree to adjust each vertex's ancestor field, and the enqueuing and checking for pending tetrahedral or faces that are waiting for their vertices to be decompressed. Table 2 shows the performance overheads for these two steps as measured from the Gatun prototype. The rendered image plane is set to 256×256. For each input dataset, we also varied the simplification ratio from 0.1 to 1.0. The hardware testbed used to collect performance measurements is a P4 1.5GHz machine with 512 MBytes of memory running RedHat Linux 7.1 . Table 2 shows the simplification overhead in terms of the percentage of the total time. Notice the simplification overhead drops as simplification ratio goes up, which is because more tetrahedra become degenerate and get discarded therefore the associated enqueuing and checking (for pending tetrahedra) overheads are also reduced. In all cases, the run-time simplification overhead is less than 5% on an average for all six test datasets. This result demonstrates that simplification is a low-overhead runtime performance adaptation mechanism for irregular-grid volume rendering.

**Table 2 The run-time simplification overhead expressed as the relative percentage of the total rendering time for different simplification ratios.**

| | Simplification Ratio | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dataset** | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 0.99 |
| **Spx** | 2.81% | 2.85% | 2.57% | 2.34% | 2.06% | 1.99% | 1.44% | 1.17% | 0.91% | 0.61% | 0.26% |
| **Fighter** | 5.00% | 5.05% | 4.66% | 4.30% | 3.91% | 3.51% | 3.08% | 2.63% | 2.11% | 0.78% | 0.19% |
| **Blunt-fin** | 7.01% | 6.58% | 6.73% | 5.85% | 5.21% | 4.87% | 3.91% | 3.20% | 2.78% | 1.56% | 0.34% |
| **Combustion** | 5.86% | 5.60% | 5.20% | 5.03% | 4.83% | 4.22% | 3.80% | 3.42% | 2.57% | 1.62% | 0.44% |
| **Post** | 7.99% | 7.60% | 7.13% | 6.66% | 6.22% | 5.48% | 4.85% | 3.92% | 2.74% | 1.93% | 0.41% |
| **Delta Wing** | 8.08% | 7.88% | 7.60% | 7.18% | 7.26% | 6.59% | 5.62% | 4.64% | 3.48% | 2.13% | 0.31% |

## 5.2 Rendering Performance Improvement

A major performance advantage of the proposed integrated pipeline is that each piece of volume data is brought into the main memory exactly once. In contrast, a baseline (Generic) implementation of the same three steps may involve reading the input tetrahedral mesh from the disk, simplifying it, and rendering the simplified mesh. The whole decompressed data is generated first before any simplification step can be applied. Similarly simplification must be completed before rendering can start. Figure 5 shows the end-to-end rendering time comparison between the baseline implementation and Gatun for each of the six test datasets. For both implementations, as the simplification ratio increases, the rendering delay decreases as expected. The rendering performance improvement due to simplification is only up to a factor of two (the Delta Wing dataset and simplification ratio 0.99). The performance improvement is not linear with respect to the simplification ratio because when the overhead of "touching" the input dataset once in order to simplify it dominates the overall performance cost. This result clearly demonstrates why simplification alone does not provide sufficient flexibility for run-time adaptation.

Between Gatun and the baseline implementation, Gatun always wins because Gatun's pipelined structure significantly reduces the amount of disk I/O due to "store and compute," especially for very large input datasets. Although there is plenty of main memory in the test machine, this performance difference is still quite noticeable. When memory resource becomes less abundant, Gatun is expected to be far even better than the baseline implementation in the end-to-end rendering time, as shown in the previous work[1].

## 5.3 Time-Critical Rendering

Figure 5 shows that without pre-computing multiple simplified versions, *even at the simplification ratio of 99%, the end-to-end rendering performance is still far from being interactive* for most datasets. To show how multi-resolution pre-simplification helps time-critical rendering, we first presimplified the Blunt-fin dataset into 13 different versions at simplification ratios $1 - 1/2^i$ where i ranges from 1 to 13. As shown in Table 3, at the image resolution of $128 \times 128$, the end-to-end delay drops to below 0.17 seconds (or 6 frames/sec), when the simplification ratio is greater than 0.984. Figure 6 shows that with pre-computation, the rendering time now indeed becomes approximately linear with respect to the target simplification ratio. It also suggests that the proposed multi-resolution pre-computation scheme can also benefit the baseline case as well, as their performances are rather close to each other (shown in Figure 5).

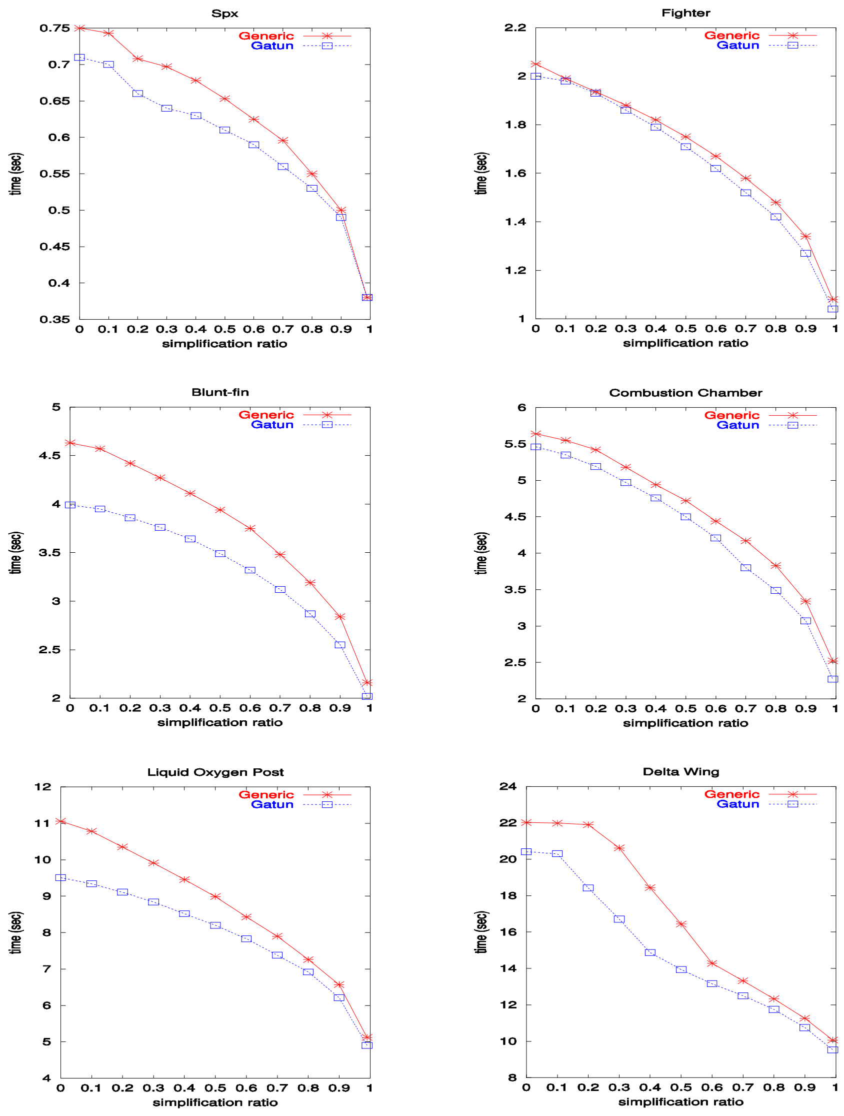


Fig. 5 Comparisons between the generic renderer (the baseline) and Gatun for different simplification ratios. There is only one version of the input dataset stored on the disk. Image resolution is set at 256 × 256.

The fact that this performance vs. simplification ratio figure is highly linear also means that it could serve as the basis for dataset-specific adaptation for time critical rendering. That is, given a desired frame rate, the system could first perform the multi-resolution pre-computation, plot the corresponding figure as in Figure 6, then it could automatically determine the most appropriate simplification ratio from such a figure, retrieve the appropriate pre-computed version, and initiate the simplification process from there. For example, for the Blun-fin dataset, if the target frame rate is two frames per second, or equivalently 0.5 seconds per frame, one can determine from Figure 6 that a simplification ratio of 0.905 could support such an interactivity requirement. In short, by applying multi-resolution pre-computation, a much more simplified mesh does not need to suffer from the huge latency introduced by decompressing the whole mesh, especially when most of the decompressed data would eventually be simplified away anyhow, thus making most of the waiting in vain. Instead, with multi-resolution pre-computation, we could make such associated decompression overhead as less as possible.

**Table 3 Rendering time for the Blunt-fin dataset for different simplification ratios, where the simplification ratio is defined by $1-1/2^i$. Image resolution is set at 128 × 128.**

| value of i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| time (sec) | 3.06 | 1.80 | 1.05 | 0.62 | 0.39 | 0.25 | 0.17 | 0.12 | 0.10 | 0.08 | 0.07 | 0.05 | 0.04 | 0.03 |

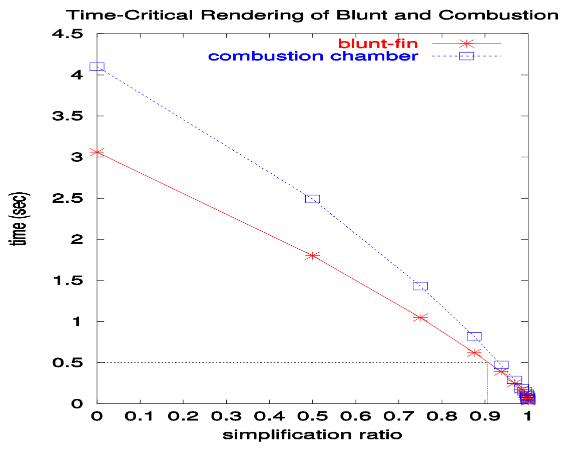


Fig. 6 Time-critical rendering of the Blunt-fin and Combustion Chamber datasets for different simplification ratios. Image resolution is set at 128 × 128.

## 6 CONCLUSION

Although lossless compression is an effective technique to reduce the storage requirement and run-time disk access cost for very large irregular volume datasets, it cannot reduce the rendering computation overhead because the number of tetrahedra that a renderer needs to process remains unaffected with or without lossless compression. Volume data simplification, on the other hand, provides a volume rendering system the additional flexibility to trade off rendering time and quality. Unfortunately, most previous research on volume data simplification focused on the development of standalone simplification tools that are never integrated into the renderer. As a result, unlike in surface rendering, where a polygonal renderer uses simplification as an effective control tool to adjust rendering accuracy and performance at run time, volume simplification has rarely been an integral part of a volume renderer as a dynamic adaptation mechanism. This paper describes the first irregular-grid volume rendering system that integrates not only volume simplification with rendering, but also volume decompression, into a single seamless pipeline. With this integrated pipeline, simplification becomes an active element of a volume rendering system, and each piece of volume data is brought into the main memory only once as it travels through the decompression, simplification, and rendering steps. We have successfully implemented the proposed integrated decompression / simplification / rendering pipeline on our Gatun system. Empirical measurements on the Gatun prototype show that the additional performance overhead associated with runtime simplification is less than 5% on an average compared to the same pipeline without simplification. However, with simplification in place, the rendering performance can be improved only by a factor of up to 2 because the entire dataset still needs to be touched at least once regardless of the simplification ratio. To address this problem, we propose a multi-resolution pre-simplification scheme similar in spirit to mip-mapping, which effectively reduces the end-to-end rendering time to be about inversely proportional to the simplification ratio, and makes a powerful building block for time-critical rendering.

In the future, we plan to integrate not only the view-independent simplification, but also the view-dependent simplification into our framework, given that the latter should provide more room for more aggressive simplification during run-time data browsing. We are also investigating other simplification primitives other than vertex merge or edge collapse which can be integrated into our framework.

## REFERENCES

1. C. Yang, T. Mitra, and T. Chiueh, "On-the-Fly Rendering of Losslessly Compressed Irregular Volume Data," in IEEE Visualization '2000, pages 101–108, 2000.
2. W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of Triangle Meshes," in SIGGRAPH '92, pages 65–70, July 1992.
3. Paul Hinker and Charles Hansen, "Geometric Optimization," in IEEE Visualization '93, pages 55–64, 1992.
4. Alan D. Kalvin and Russell H. Taylor, "Superfaces: Polygonal Mesh Simplification with Bounded Error," IEEE Computer Graphics and Applications, 16(3):64–77, May 1996.
5. Greg Turk, "Re-tiling Polygonal Surfaces," in IEEE Visualization '92, pages 55–64, 1992.
6. J. Rossignac and P. Borrel, "Multi-Resolution 3D Approximation for Rendering Complex Scenes," Geometric Modeling in Computer Graphics, pages 455–465, 1993.
7. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution Analysis of Arbitrary Meshes," in SIGGRAPH '95, pages 173–182,

August 1995.
8. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization," in SIGGRAPH '93, pages 19–26, July 1993.
9. H. Hoppe, "Progressive Meshes," in SIGGRAPH '96, pages 99–108, August 1996.
10. H. Hoppe, "View-Dependent Refinement of Progressive Meshes," in SIGGRAPH '97, pages 189–198, August 1997.
11. Julie C. Xia and Amitabh Varshney, "Dynamic View-dependent Simplification for Polygonal Models," in IEEE Visualization '96, pages 327–334, 1996.
12. J. Cohen, A. Varshney, D. Manocha, G. Turk, and H.Webber, "Simplification Envolopes," in SIGGRAPH '96, pages 119–128, August 1996.
13. J. El-Sana and Y. Chiang, "External Memory View-Dependent Simplification," Computer Graphics Forum (Eurographics '2000), 19(3):139–150, 2000.
14. P. Cignoni, L. D. Floriani, C. Montani, E. Puppo, and R. Scopigno, "Multiresolution Representation and Visualization of Volume Data," IEEE Transactions on Visualization and Computer Graphics, 3(4):352–369, December 1997.
15. P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno, "Simplification of Tetrahedral Meshes with Accurate Error Evaluation," in IEEE Visualization 2000', pages 85–92, October 2000.
16. K. J. Renze and J. H. Oliver, "Generalized Unstructured Decimation," IEEE Computer Graphics and Applications, 16(6):24–32, 1996.
17. M. H. Gross O. G. Staadt, "Progressive Tetrahedralizations," in IEEE Visualization '98, pages 397–402, October 1998.
18. I. J. Trotts, B. Hamann, and K. I. Joy, "Simplification of Tetrahedral Meshes with Error Bounds," IEEE Transactions on Visualization and Computer Graphics, 5(3):224–237, 1999.
19. P. Chopra and J. Meyer, "TetFusion: An Algorithm for Rapid Tetrahedral Mesh Simplification," in IEEE Visualization '2002, pages 133–140, October 2002.
20. C. S. Co, B. Heckel, H. Hagen, B. Hamann and K. I. Joy, "Hierarchical Clustering for Unstructured Volumetric Scalar Fields," in IEEE Visualization '2003, pages 325–332, October 2003.
21. A. V. Gelder, V. Verma, and J. Wilhelms, "Volume Decimation of Irregular Tetrahedral Grids," in Computer Graphics International 1999 Conference Proceedings, pages 222–, 1999.
22. J. Schneider and R. Westermann, "Compression Domain Volume Rendering," in IEEE Visualization '2003, pages 293–300, October 2003.
23. R.Farias, J. Mitchell, C. Silva, and B. Wylie, "Time-Critical Rendering of Irregular Grids," in Proceedings of the XIII SIBGRAPI International Conference, pages 243–, 2000.
24. T. Mitra and T. Chiueh, "A Breadth-First Approach to Efficient Mesh Traversal," in Proceedings of the 13th ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware, pages 31–38, September 1998.
25. L. Williams, "Pyramidal Parametrics," in SIGGRAPH '83, pages 1-11, 1983.

**Chuan-Kai Yang (楊傳凱)** received his Ph.D. degree in computer science from Stony Brook University, New York, USA, in 2002, and his M.S. and B.S. degree in computer science and in mathematics from National Taiwan University in 1993 and 1991, respectively. He has been an Assistant Processor of the information management department, National Taiwan University of Science and Technology, since 2002. His research interests include computer graphics, scientific visualization, multimedia systems, and computational geometry.

**Tzi-Cker Chiueh** (闕志克) is currently a Professor in Computer Science Department of Stony Brook University. He received his B.S. in EE from National Taiwan University, M.S. in CS from Stanford University, and Ph.D. in CS from University of California at Berkeley in 1984, 1988, and 1992, respectively. He received an NSF CAREER award in 1995. Dr. Chiueh's research interest is on computer security, network/storage Qos, and wireless networking.