Volume Graphics (2005)
E. Gröller, I. Fujishiro (Editors)

Integrated Irregular Volume Decompression and Out-of-Core Iso-Surface Extraction

Chuan-Kai Yang[†] and Tzi-Cker Chiueh[2‡]

Department of Information Management,National Taiwan University of Science and Technology, Taiwan
[2] Department of Computer Science, Stony Brook Univerity, USA

Abstract

Volume data sets tend to grow bigger and bigger as modern technology advances, thus imposing a storage constraint on most systems. One general solution to alleviate this problem is to apply volume compression on the volume data sets. However, as volume rendering is often the most important purpose why a volume data set was generated in the first place, we must take into account how a volume data set could be efficiently rendered when it is stored in a compressed form. Our previous work [YMC00] has shown that it is possible to perform a on-the-fly direct volume rendering from irregular volume data. *In this work, we further extend that work to demonstrate that a similar integration can also be achieved on* iso-surface extraction *and volume decompression for irregular volume data. In particular, our work involves a data set decomposition process, where instead of a* coordinate-based *decomposition used by conventional out-of-core iso-surface extraction algorithms, we choose to use a* layer-based *structure. Each such layer contains a collection of tetrahedra whose associated scalar values fall within a specific range, and is compressed independently to reduce the storage requirement. The layer structure is particularly suitable for* out-of-core iso-surface extraction*, where the required memory exceeds the physical memory capacity of the machine that the process is running on. Furthermore, with this work, we can perform* on-the-fly iso-surface extraction during decompression*, and the computation only involves the layer that contains the query value, rather than the entire data set. Experiments shows that our approach could improve the performance up to ten times when compared with the results based on the traditional coordinate-based approaches.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Graphics UtilitiesGraphics Packages I.3.6 [Computer Graphics]: Methodology and TechniquesInteraction Techniques

1. Introduction

Modern sensors/simulators can offer more and more accurate sampling in both spatial and temporal dimensions, thus leading to volume data sets with enormous size. To reduce the storage requirement for volume data sets, volume compression techniques are often used. However, as volume rendering should eventually be performed on each volume data set, how to efficiently render a volume data set when it is stored on disks in a compressed form becomes an important issue. A naive approach would be deferring the whole rendering process until the decompression process is complete. Although simple and flexible, this approach introduces an undesired long latency for the renderer and the required memory for execution cannot take advantage of volume compression at all. Previously we have done similar work on combining direct volume rendering with volume decompression [YMC00]. In this paper, we further extend our work to *indirect volume rendering* (i.e. *iso-surface extraction*) and propose a scheme to integrate the volume decompression and iso-surface extraction into a single pipeline, thus minimizing both the rendering latency and required memory.

As volume data sets grow bigger and bigger, the iso-surface extraction process may become *out-of-core*, that is, the required memory for execution is more than what a sys-

[†] Email: ckyang@cs.ntust.edu.tw
[‡] Email: chiueh@cs.sunysb.edu

tem can provide. To deal with this, one of the common conventional approaches is to decompose a data set into partitions based on its coordinate information, so that at run time each partition can be loaded into memory for iso-surface extraction, and the final result is the union of all the iso-surfaces extracted from each partition. Rather than applying the conventional approaches, this work employs a different way of decomposing a volume data set into *layers*, where each layer corresponds to a collection of tetrahedra whose associated scalar values fall within a specific range. We argue that this decomposition is more "friendly" to the iso-surface extraction process than the conventional ones, because at run time, for any query, at most one layer is needed to extract the corresponding iso-surfaces.

2. Related Work

An iso-surface extraction algorithm extracts surfaces from a volumetric data set when given an iso-value or iso-density of interest. The given data sets can be either regular or irregular. Many different iso-surface extraction algorithms have been proposed. Lorensen et al. [LC87] pioneered the iso-surface extraction research field with the proposal of *Marching Cubes* algorithm on regular data sets. *Marching Tetrahedra* is a similar idea to marching cubes, but instead of operating on cubes, it operates on tetrahedra.

There are essentially two phases in iso-surface generation. The first phase is the search for intersecting cells. The second phase is the generation of triangles. While the second phase can be sped up by using a table-lookup and thus leaves little room for further improvement, the first phase can be accelerated significantly by using different data structures and/or searching algorithms. There are basically three approaches: space-based, range-based and surface-based. The first approach partitions the input data set based on cells' spatial coordinates, and the second approach on the scalar value associated with each cell. The last approach tries to grow iso-surface using some adjacency information from a much smaller *seed-set*, which is identified in the preprocessing time.

Using a range-based approach, Cignoni et al. [CMM*97] demonstrated how to use the concept of *interval tree* to answer an iso-surface query, which is essentially a *stabbing query*, and the original idea came from Edelsbrunner [Ede80] and McCreight [McC80]. This method could achieve the optimal time complexity for searching intersecting cells. Chiang et al. [CS97] applied this idea but changed the branching factor to make the size of each node in the interval tree no bigger than a disk block to significantly reduce the number of disk I/Os and the memory required for query processing. Later Chiang et al. [CSS98] extended their work to perform *out-of-core* iso-surface extraction as well by partitioning the original grids into *meta-cells* and used a KD-tree decomposition to make sure each meta-cell fit into the memory.

3. Layered Structures for Out-of-Core Iso-Surface Extraction

Unlike the work by Chiang et al. [CSS98], which decomposes a data set "spatially" according to its coordinates information, we argue that a decomposition of a data set based on its scalar values should be more efficient, as iso-surface extraction is to extract surfaces for a particular *iso-value*.

The *value-based* decomposition approach works as follows. First, we compute the range that encompasses all scalar values. Second, this range is partitioned into sub-ranges. There are many ways to partition a range into sub-ranges. We make use of the histogram and ensure that each sub-range catches more or less the same number of tetrahedra. This method seems very attractive because the number of sub-ranges, say N, can more of less guarantee that only $1/N$ of the total tetrahedra need to be touched to answer a query, assuming the number of tetrahedra that span multiple sub-ranges is relatively small. However, the implementation of this is amazingly difficult, because the overlap among layers is not negligible. Currently we try to solve this balanced-partitions problem through the use of a well-known optimization technique called *simulated annealing*.

After determining the boundaries of sub-ranges, the third phase is the distribution phase. Each tetrahedron is distributed to the sub-range with which its data value range overlaps. As mentioned, a tetrahedron may intersect with multiple sub-ranges, but in general such cases should mostly happen at the boundary of each sub-range. Notice the resulting sub-mesh, corresponding to each sub-range, need not be connected, even though the original mesh is. Worse yet, the sub-mesh may not even be a *manifold*, even though the original mesh is. As our current tetrahedral mesh compression algorithm still cannot compress a non-manifold mesh, this requires a little modification to the decomposition algorithm.

4. Performance Results

In this sub-section we demonstrate the performance of our system. A Pentium 4 1.5GHz machine, with 512MB memory and running on Linux Redhat 7.2, is used to conduct our experiments. We compare the results among the following four approaches. First, a query is directly performed on the original mesh. Second, we partition each of the original six meshes into $4 \times 4 \times 4 = 64$ spatial partitions, i.e., making 4 partitions along each primary axis. A query is sent to each partition whose aggregated scalar values range contains the query value. Notice that such a spatial partition is common when performing *out-of-core* iso-surface extraction, so that each partition can be processed within the memory. The choice of $4 \times 4 \times 4 = 64$ is rather arbitrary as their associated storage overhead is relatively small. Here the overhead is defined as the total size, in terms of number of tetrahedra, of all partitions, excluding the original data set size. Notice that, compared with the space-based approach, the overhead of the layer-based approach is usually larger. This is mainly because to overlap with a spatial partition, a tetrahedron must have its coordinates intersect with the spatial partition in all x, y, and z components, thus reducing the probability of intersections. This in general leads to smaller overhead. Third, we partition each of the six meshes into layers, and each layer is patched to a manifold and compressed individually. At run time, only one layer (there may be more than one layers qualify) is retrieved and an *on-the-fly iso-surface extraction during decompression* step is performed exclusively on that layer. The fourth approach is the same as the third one except that we do not compress each layer but store each layer as a binary file on the disk. At run time, only one layer will be retrieved and searched for intersecting tetrahedra, but no decompression will be involved.

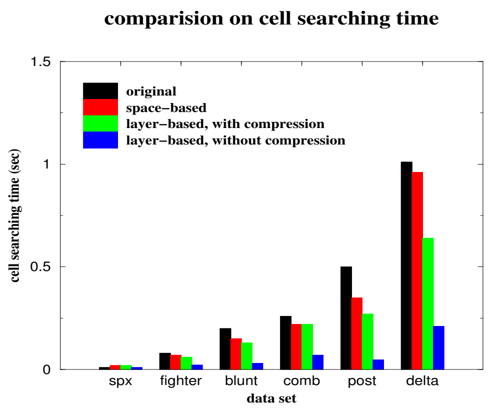


Figure 1: *Comparison of the cell searching time among four approaches to perform iso-surface extraction for each data set. We uniformly select 100 sampled scalar values from the scalar value range of each data set, to be used as the query values.*

As mentioned before, iso-surface extraction includes two phases, searching for intersecting tetrahedra, and generation of triangles from those intersecting tetrahedra. The results shown in Figure 1 only include numbers from the first phase, as the generation of triangles from the intersecting tetrahedra should take roughly the same time when the resulting tetrahedra are the same for all four approaches. Our goal is to be able to perform *out-of-core iso-surface extraction*, where a input data set could not be completely memory-resident. For such data sets, many techniques proposed to speed up the search phase, becomes either useless or difficult to use. For example, the *interval tree* requires the same order of space complexity as the data set size itself, in terms of required memory. Either building an interval tree before searching, which takes more than linear time of processing, or loading an interval tree that was pre-built in the preprocessing time, requires at least a linear time of processing. This makes the use of an interval tree to speed up the searching phase pointless. We therefore just use a linear search for all three approaches in the searching phase. As can be seen from this figure, our layer-based approach, or the third approach, wins in most cases. One of the benefits of our system is the uniform size of each partition and each time only one layer needs to be touched to extract the target iso-surface. On the contrary, a spatial-decomposition may fail to provide such an advantage. For example, if the *blunt-fin* data set is to be spatially decomposed into $3 \times 3 \times 3$ partitions, then for a query value of 3, only 11.11% of the partitions need to be touched. 33.3% of the partitions need to be touched if a query value of 2 is given. However, all partitions are needed if the input query value is 1. Figure 4 further compares the four approaches in terms of the number of IOs (just reads, no writes) performed, and the number of bytes read in for each of the four approaches. As can be seen here, the third approach, i.e., the layer-based approach with compression, requires the least number of IOs and least number of bytes to read. However, due to its decompression overhead, it is still slower than the fourth approach. On the other hand, the fourth approach only requires more number of IOs and more bytes to read than the third approach. Without the decompression overhead, it is much faster than all other approaches. Therefore, in the case where storage is not a big issue and memory capacity is of a bigger concern, such as the case for out-of-core iso-surface extraction, the layer-based approach without compression could provide an even better solution. Even when storage and memory do not pose as issues, the fourth approach could still give us up to 10 times performance improvement, as shown in Figure 1. If both storage and memory are concerns, then the third approach, i.e., the layer-based approach with compression, may provide a better solution.

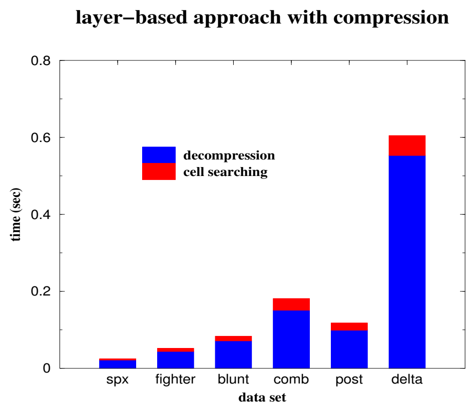


Figure 2: *Breakdown of the cell searching time for iso-surface extraction using the layer-based approach with compression.*

5. conclusion

We have successfully integrated iso-surface extraction with volume compression. Our system first decomposes an in-

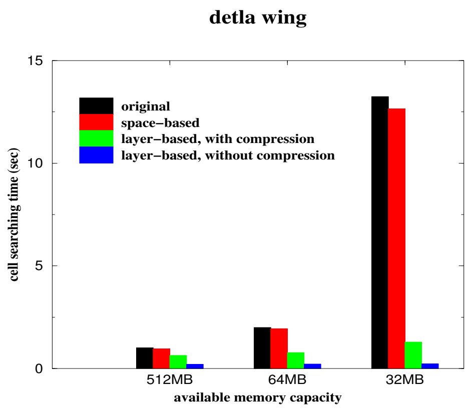
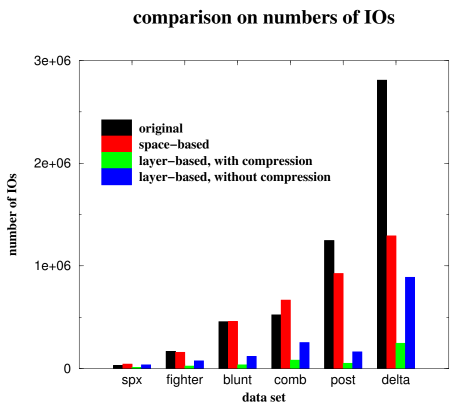


Figure 3: *The cell searching time for iso-surface extraction on the* delta wing *data set under different memory capacity.*



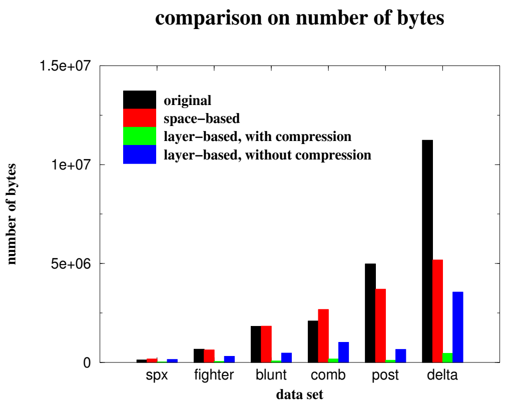


Figure 4: *Comparisons on the number of IOs and number of bytes read from the disk, for all the four approaches mentioned in the text.*

put tetrahedral mesh into layers. Each such layer consists of tetrahedra whose scalar values correspond to a particular sub-range of the scalar value range of the original mesh. For an iso-surface query, only one such layer is required to generate the desired iso-surface results. We individually compress each of these layers to save their storage requirements. At run time, we first locate a layer that contains the query value, then perform the *on-the-fly iso-surface extraction during decompression* exclusively on that layer. This layer-based decomposition is especially useful for the purpose of *out-of-core* iso-surface extraction, as we could make a layer as small as we desired so that it can be fit into the main memory, and no other layers will be needed for outputting the correct results. In contrast, the traditional approach to handle out-of-core iso-surface extraction is to decompose a data set spatially. Although each such partition can be made small enough to fit into the main memory, there is no guarantee that other partitions won't be touched. In the case where disk storage is not a concern but memory is, the use of layer-based partition should be even more attractive. Our system could improve the cell searching time in the iso-surface extraction process up to 10 times faster, without increasing the disk storage requirement of the input mesh. However, we must point out that for data sets containing high variations where most of the iso-surfaces span most regions of the entire data sets, out approach may not provide much improvement over the space-based approaches for obvious reasons.

References

[CMM*97] Cignoni P., Marino P., Montani C., Puppo E., Scopigno R.: Speeding Up Isosurface Extraction Using Interval Trees. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (April 1997), 158–170.

[CS97] Chiang Y., Silva C.: I/O Optimal Isosurface Extraction. In *IEEE Visualization '97* (October 1997), pp. 293–300.

[CSS98] Chiang Y., Silva C., Schroeder W.: Interactive Out-of-Core Isosurface Extraction. In *IEEE Visualization '98* (October 1998), pp. 167–174.

[Ede80] Edelsbrunner H.: *Dynamic Data Structures for Orthogonal Intersection Queries*. Tech. Rep. Report F59, Inst. Informationsverarb.,Tech. University Graz, 1980.

[LC87] Lorensen W. E., Cline H. E.: Marching Cube: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics 21*, 4 (July 1987), 163–169.

[McC80] McCreight E. M.: *Efficient Algorithms for Enumerating Intersecting Invervals and Rectangles*. Tech. Rep. Report CSL-80-9, Xerox Palo Alto Res. Center, 1980.

[YMC00] Yang C., Mitra T., Chiueh T.: On-the-Fly Rendering of Losslessly Compressed Irregular Volume Data. In *Proceedings on Visualization '2000* (October 2000).