EUROGRAPHICS 2004 /
Oliver Deussen, Charles Hansen, Daniel A. Keim and Dietmar Saupe
(Editors)

Volume 22 (2003), Number 3

Compression and Occlusion for Fast Isosurface Extraction from Massive Datasets

Benjamin Gregorski,[1,2†] Joshua Senecal[1,2], Mark Duchaineau[1] and Kenneth I. Joy[2]

[1]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory
[2] Visualization and Computer Graphics Group, Center for Image Processing and Integrated Computing(CIPIC),
Department of Computer Science, University of California Davis

Abstract

We present two algorithms for data compression and occlusion culling for interactive, adaptive isosurface extraction from large volume datasets. Our algorithm, based on hierarchical tetrahedral meshes defined by longest edge bisection, allows arbitrary isosurfaces to be adaptively extracted from losslessly compressed volumes where only the region of interest needs to be decompressed. For interactive applications, we exploit frame-to-frame coherence between consecutive views to simplify the mesh structure in occluded regions and eliminate occluded triangles. We extend the use of hardware accelerated occlusion queries to adaptive isosurface extraction applications where the surface geometry and topology change with the level-of-detail and view-point, and the user can select an arbitrary isovalue for visualization.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation I.3.3 [Computer Graphics]: Viewing Algorithms

1. Introduction

Numerical simulations and data acquisition devices such as Computed Tomography scanners are generating increasingly larger datasets that still need to be visualized at interactive rates. To accomplish this, researchers have developed parallel and/or hierarhical algorithms for visualization. The former allow interactive visualization by leveraging the processing power of a large number of computers, and the later allow interactive visualization by using lower resolution versions of the volume, adapting the visualization to some user defined criteria, and organizing the data for coherent access across the memory hierarchy.

In this paper, we focus on hierarchical algorithms based on tetrahedral meshes. These meshes have been used extensively in scientific visualization for performing isocontouring [GP00], [Ger01], and can effectively deal with large datasets through the use of a hierarchical data layout based on space filling curves [GDL*02, LP01, Pas02].

The visualization of massive volume datasets is made difficult by the fact that the entire dataset cannot fit into a computer's core memory. In addition, disk storage speed has been growing at a much slower rate when compared with CPUs and GPUs and direct memory transfers. Thus, visualization algorithms need to effectively organize data on disk so that it can be paged in when needed. Data compression techniques are essential to overcoming this performance gap; they reduce the number of times data must be loaded from slower disks, and they increase the amount of data that can be read at a time. In order to be effective for interactive applications, decompression must be fast and localized to the region of interest. As CPUs and GPUs improve in performance, the cost of decompression becomes much smaller than the cost of reading massive, uncompressed data from disk. Thus, regions of the compressed volume can be prefetched and cached in memory until they are needed for visualization.

† gregorski1,senecal1,duchaineau1@llnl.gov,kijoy@ucdavis.edu

Compression techniques for interactive visualization have been utilized in the context of volume rendering by Guthe et al. [GS01], where wavelets and mpeg style encoding are used to compress time-varying volumes and by Lum et al. [LMC01] where quantization of DCT coefficients is used. Hierarchical visualization, which allows regions of the dataset to be accessed separately from others and at lower resolutions if necessary, is a very effective technique for visualizing large volumes. This is especially true when subregions of the dataset are being visualized and when lower resolution versions of the dataset may be acceptable for visualization, e.g. when interactive frame rates can be maintained. In [WKE99], Westermann et al. use octrees to perform adaptive isosurface extraction. Their algorithm is similar to ours, however, we use the tetrahedral mesh to extract a crack-free isosurface and they fix cracks between transition levels with triangle fans. Guthe and Strasser [GWGS02] utilize wavelet based compression to adaptively render volume datasets using texture hardware. An octree decomposition breaks the volume into a hierarchy of 2^{3k} blocks and linear spline wavelets are used to transform the data. Arithmetic and Huffman coding are used to compress the wavelet coefficients. Schneider and Westermann [Wes03] utilize a hierarchical vector quantization scheme for compressing static and time-varying volumes. Their algorithm could also be used for hierarchical visualization and it has the advantage that rendering can occur directly from the compressed representation removing the overhead of decompression. Their algorithm also uses an octree-based(cubical) decomposition scheme to divide the dataset.

Our data compression algorithm uses a tetrahedral mesh hierarchy defined by longest-edge bisection. The cubical blocks of an octree decomposition, used to define the data prediction algorithms, are replaced with the diamond shapes developed in Gregorski et al. [GDL*02] and used by Linsen et al. [LGP*03] for hierarchical representation of large dataset. We use the top-down traversal of the mesh hierarchy to predict data values and gradient components for lossless compression of large volume datasets, This prediction scheme follows the data access pattern dictated by the mesh refinement. Combined with the z-order data layout scheme of Pascucci et al. [Pas02], it ensures that the reconstruction masks are of minimal size and that data values necessary for reconstruction can be accessed in a memory and disk coherent manner. Furthermore, it ensures that the decompression can be confined to local regions of the volume which is essential for visualization of large datasets. The levels of our hierarchy are created from subsampled versions of the dataset. Thus, as finer (higher resolution) levels of the volume are reconstructed, values at coarser levels do not need to be updated (as they must be when wavelet lifting schemes are used).

Another problem with contouring massive volumes is that the extracted isosurfaces contain millions of triangles and almost always have a depth complexity greater than one. This means that a pixel on the screen can have a large number of fragments rendered to it even though many of those fragments are invisible. In general, a large number of occluded triangles, triangles within the view frustum that do not affect the final image, are extracted and rendered. When viewing an isosurface at a high resolution, extracting and rendering a large number of occluded triangles consumes a large amount of processing, memory, and rendering resources. Occlusion culling prevents the invisible triangles from being extracted and rendered, thus saving memory and rendering time. This allows valuable system resources to be allocated to the visible regions of the surface, and it is essential for maintaining interactivity during isosurface extraction from massive datasets.

A large amount of work has been done on occlusion for polygonal models. Software based occlusion methods precompute a visibility database to help determine occluded areas. In [ESSS01], the dataset is divided against a grid and a solidity value is determine for each cell. A cell's visibility is based on the solidity values of the cells that intersect the line segment between the view point and the cell's center. In [ZMHI97], a hierarchical occlusion map is used to select occluders from a database. Occluders are rendered as white polygons on a black background. A estimated depth buffer is constructed to allow occluded regions to be detected. Modern graphics hardware has the ability to perform occlusion queries and to report the number of pixels affected by some geometry. This new feature has been used extensively for occlusion culling in large polygonal environments, see [GLY*03, YSM03, HSLM02, Ha03].

Occlusion culling methods have also been extended to volume datasets for isosurface extraction and volume rendering applications. Livnat and Hansen [LH98] traverse and octree decomposition of the dataset front-to-back and use a hierarchical visibility test based on coverage masks to determine occluded regions. As occluded regions are determined during the traversal, blocks of the volume are culled and no isosurface is extracted from them. In [XZ02], the dataset is decomposed into a set of blocks, and ray casting is used to select a group of blocks that are occluders which are then rendered to create a mask of covered screen pixels. The remaining unoccluded blocks, as determined by this mask, are rendered. Recent work such as [LMK03] and [GHSK03] has extended occlusion culling techniques to volume rendering applications.

Our occlusion culling algorithm is similar to the algorithm of Livnat and Hansen [LH98]. We replace their octree with our tetrahedral mesh hierarchy based on diamonds (Section 2). Since the diamond hierarchy does not allow for front-to-back rendering, we utilize frame-to-frame coherence between successive view-points to create an approximate depth buffer against which occlusion tests are performed. The software occlusion tests are replaced with hardware occlusion

queries, and the mesh structure is dynamically refined and coarsened as regions become visible and occluded.

2. Mesh Refinement

In this section, for reference purposes, we review the mesh refinement and its relationship with the z-order space filling curve [Pas02]. The refinement of a tetrahedral mesh by longest-edge bisection has three phases and begins with a cube divided into six tetrahedra around its major diagonal. The three refinement phases, which occur around cube diagonals, face diagonals, and edges of the mesh, are illustrated in Figure 1. Figure 2 shows the three types of polyhedral shapes or diamonds that occur around the edges of the mesh. The diamonds are the unit of refinement.

The *split* and *merge* operations are used to refine and coarsen the mesh. When a diamond is split, all tetrahedra in the diamond are split into two child tetrahedra. Merging a diamond removes all the child tetrahedra and merges two child tetrahedra into one tetrahedron. Splitting and merging diamonds ensures that the mesh is always crack-free. The split/merge process is implemented with two error-based priority queues [DWS*97, GDL*02], the split queue and the merge queue, which contain all diamonds that can be split and merged respectively. At runtime, given an error threshold E, diamonds in the split queue with an error greater than E are refined, and diamonds in the merge queue with an error less than E are coarsened.

Figure 3 illustrates the connection between the mesh refinement and the z-order space filling curve. The order of the points introduced by each level of mesh refinement is essentially the same as the order given by the z-order curve. The only difference is that the mesh refinement operates on $(2^k+1)^3$ grids, and the z-order curve works on $(2^k)^3$ grids. The z-order tiles the grid in space causing index 2^k to wrap to 0. Each (i,j,k) index of point P in the dataset corresponds to a $z-order$ index on the one dimensionsl z-order curve. It is easy and fast to convert between (i,j,k) and $z-order$ indices, details in [Pas02]. These features of the mesh hierarchy and the z-order curve give excellent disk and memory coherence making them well suited for adaptive, out-of-core visualization of large datasets.

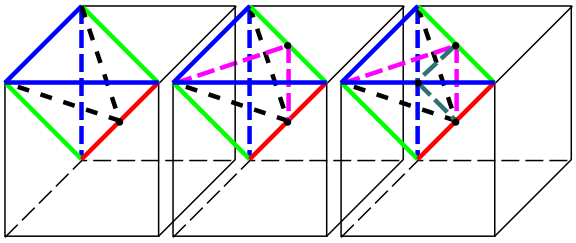

Figure 1: *From left to right: phases two, one, and zero of the mesh refinement.*

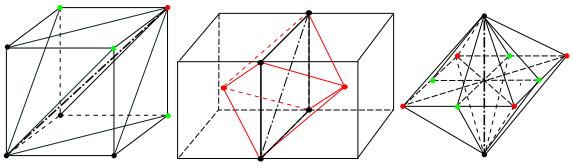

Figure 2: *From left to right: diamond shapes for the phases two, one, and zero of the mesh refinement. The split or refinement edge is the bold, circle-segment dashed line.*

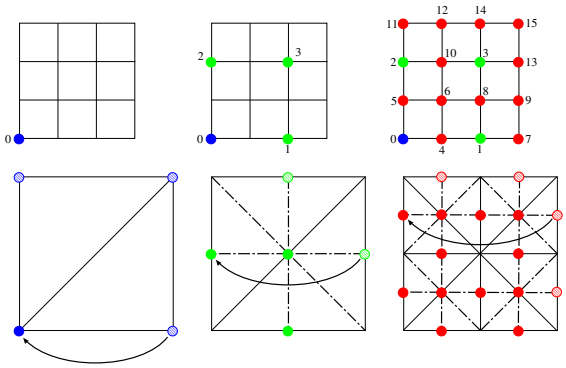

Figure 3: *2-D example of z-order and mesh refinement. Top: the order of the data points on the z-order curve. Bottom: The data points introduced by the mesh refinement at each hierachy level. The dashed circles and curved arrows illustrate how* (2^k+1) *indices in the mesh hierarchy wrap to* 0 *on the z-order curve.*

3. Data Compression

Our data compression algorithm is divided into three phases. In the first phase, the original volume is traversed along the tetrahedral mesh hierarchy, and a prediction scheme is used to build a histogram of the differences between the actual data and the data given by the prediction. In the second phase, these delta values are passed to the encoder which builds a set of codes used to compress the delta values. In the final phase, the stream of delta values is divided into pages and compressed using the codes generated in phase two.

For multiresolution extraction of isosurfaces, it is important to use hierarchical prediction that follows the refinement of the multiresolution mesh. This ensures that the runtime decompression algorithm can efficiently locate the data needed for reconstruction. It also ensures and that decompression is localized across the space and scale of the volume hierarchy. Decompressing a region at a high resolution only occurs in a small localized space around the region and in similarly localized spaces at coarser levels of the hierarchy.

3.1. Prediction

Our data prediction algorithm, based on difference from linear prediction along a diamond's refinement edge, is illustrated for two-dimensional refinement in Figure 4. The predicted data is

$$D_p = \frac{(D_{SV_0} + D_{SV_1})}{2}. \quad (1)$$

The delta value encoded later is

$$\delta = D_V - D_p. \quad (2)$$

In three dimensions, the predicted value is computed in the same manner since the diamond whose split vertex is V always has two points on its split edge regardless of dimension. This simple predictor handles boundary conditions easily because the diamonds on the faces of the volume are always phase one and zero diamonds, and the diamonds on the edges (except for the corners which are not predicted) are phase zero diamonds (Section 2). Thus, there are no special cases. Figure 9 shows histograms of two test datasets before and after prediction.

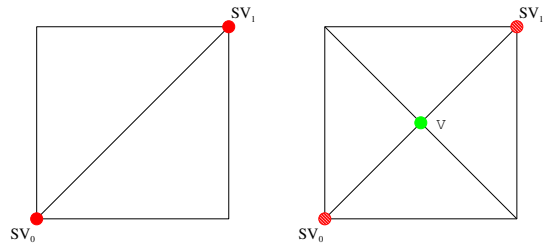

Figure 4: *In difference from linear prediction, the value at a point V, which is also the split vertex of a diamond, is predicted as the midpoint of its two split edge values. The red, dashed circles, SV_0 and SV_1, indicate the split edge vertices.*

3.2. Compression and Decompression

After the transformation described in Section 3.1, the original data has been converted to a representation that is easier to compress. The transformed data's coefficient magnitudes are clustered toward smaller values than those of the original data. The magnitudes show that the position of the leading 1, in the binary representation of the data, is highly redundant. Consider an 8-bit transform coefficient, such as 00000101, the high-order bits 000001 are more compressible than the remaining bits 01.

To exploit this property of the transformed data we utilize *lead-1 encoding*. If the transformed data values (signed data) require 8 bits for representation, there 128 different coefficient magnitudes and 256 possible values. Instead of encoding all possible coefficient magnitudes, we encode the position of the magnitude's leading 1. The remaining bits of the magnitude and the sign bit are passed through uncompressed. For 8-bit transform coefficients, the number of symbols that need to be represented is reduced to 9 from 256 (i.e. the 8 possible positions of the leading 1 and the zero coefficient versus the 256 possible values that can be represented with 8 bits). The codes for the leading 1 position are generated using the Huffman algorithm. The compressed output consists of the codeword followed by the sign bit if the magnitude is non-zero and the remaining low-order bits after the leading one.

The encoding process utilizes one value of context to generate the codes. Given the sequence of transformed values generated by the prediction process, the encoder creates a table of leading-1 counts based upon the previous and current lead-1 positions. This 2-D table of counts is used by the encoder to build the Huffman codes for the leading 1 positions. An encoding and decoding table is required for each previous leading 1 position.

Lead-1 encoding and Huffman codes were chosen for several reasons. First, encoding and decoding can be done with direct table lookups which is very fast. Second, the number of symbols that need to be encoded is greatly reduced, minimizing the size of the encoding and decoding tables. The generated Huffman codes are limited in length by the number of input symbols. If there are n possible symbols to be encoded, the longest code length will be at most $n-1$ bits. Lead-1 encoding reduces the number of symbols that need to be encoded thus reducing the size of the codes and the table size. An entry in the decoding table stores the length of the codeword that indexes to that entry and the leading 1 position indicated by the code. From this information the decoder can determine the number of bits after the leading 1 and how far to shift the encoded bit stream during decompression. This information is packed into a single byte which ensures that the decoding tables can fit in cache memory. Lastly, when compared with arithmetic and arithmetic-type encoders, lead-1 encoding offers an acceptable compression ratio with superior speed and throughput.

Since our datasets are stored in hierarchical z-order (Section 2), the transformed coefficients are also stored in z-order. To allow for local decompression, we divide the transformed coefficients into pages of 2^{12} data points and encode each page separately. A lookup table is used at runtime to find the disk offset for loading a particular page.

Given an index $P_{(i,j,k)}$ for a point P, the data value and gradient at P are reconstructed as follows:

1. Convert the (i,j,k) index $P_{(i,j,k)}$ to its z-order index P_z. Using P_z, locate the disk page DP that contains P, and decompress DP to get the delta values associated with DP's data points.
2. Since the deltas are stored in z-order, we know the z-order index P_z for all points in DP. For each point in DP, compute its (i,j,k) index $P_{(i,j,k)}$ from P_z.

3. Given that this (i, j, k) index, $P_{(i,j,k)}$, is the split vertex of a diamond, compute the (i, j, k) indices of the vertices necessary to reconstruct the value at P.
4. Fetch the surrounding values needed for reconstruction and compute the original data value. This may require recursive decompression as necessary to obtain all of the surrounding values.

Since our data is stored in hierarchical z-order, the data values d_i needed to reconstruct the data at a point $P_{(i,j,k)}$ all have z-order indices z_i such that $z_i < P_z$. This means that they come from coarser levels of the mesh and from earlier positions (relative to the file offset) in the data file than P. Furthermore, z-order stores the data points first by level-of-detail and then by spatial proximity within each level-of-detail, and it has been shown to have good memory and disk coherence properties. Accessing the coarse and fine data points, from memory and disk, needed for reconstruction, exploits these coherence properties of the storage order. Compression results for test datasets are given in Table 1.

Dataset	Uncomp	Comp	Data+G	Data
Bucky 1024^3	4096	1334	3.071	5.51
PPM 512^3	512	284.9	1.797	2.43
XMasTree 512^3	640	378.1	1.707	4.06

Table 1: *Compressed and Uncompressed sizes of test datasets in MB. The Uncomp and Comp columns show the uncompressed and compressed sizes of the combined data and gradients file. The Data+G column shows the compression ratio for the data and gradients combined and the Data column shows the compression ratio for the data only.*

4. Occlusion Culling

Occlusion queries on modern graphics hardware allow one to render geometry and determine the number of samples that pass the depth and stencil tests. This indicates the number of screen pixels affected by the rendered geometry. By disabling writes to the depth and color buffers, one can determine the number of screen pixels that would have been modified by the given geometry had it actually been rendered. Since the results must be read back from the graphics hardware, there is a noticeable delay between the time when the query is issued and the time when the results are available. One of the keys to efficiently utilize hardware occlusion queries is to fill this time gap with useful computations that can be used later on. In order to minimize this delay, we issue several queries before reading the results of the first query.

Occlusion culling is performed on the diamonds which drive the mesh refinement and coarsening process. The advantage of using diamonds instead of tetrahedra for occlusion culling is that diamonds, being the unit of refinement in our hierarchy, allow us to quickly eliminate whole regions of refinement with a single occlusion test. Unlike a tetrahedron, a diamond's children are not contained within its convex hull. This means that the occlusion queries cannot be performed top-down. Performing occlusion culling on the diamonds allows the mesh to be coarsened enough so that the number of occlusion tests performed is minimized.

Our occlusion culling algorithm works as follows:

1. At the start of a frame f_i, we render the isosurface from frame f_{i-1} and then initiate an occlusion query for all diamonds in the split and merge queues (Section 2) within the current view frustum that were occluded in frame f_{i-1}. This is done by rendering the diamond's bounding box. Visible diamonds are allowed to remain visible for several frames before occlusion queries are performed on them. Queries are performed on occluded diamonds every frame.
2. After all of the queries have been issued, we read back the results of the occlusion queries in the order that they were executed and recompute the error for diamonds in the split and merge queues. Occluded diamonds are given an error of zero. Thus, occluded diamonds in the split queue are never split, and occluded diamonds in the merge queue can be merged as necessary to simplify the mesh.
3. To exploit the frame-to-frame coherence present in view-dependent refinement, the occlusion queries in frame f_i are performed against the depth buffer created when the isosurface from frame f_{i-1} is rendered from f_i's view point. This can result in visible regions of f_i not being rendered until f_{i+1}. However, it provides a good initial guess for the occluders, gives the correct depth buffer for those occluders, and delays the set of visible diamonds by only one frame.

4.1. Changing Isovalues

When the isovalue is changed, the currently extracted isosurface can no longer be used as a valid set of occluders against which occlusion queries can be performed. This is because the new isosurface can be dramatically different from the previous one. When the isovalue is changed by the user, all triangles for the diamonds in the split queue and outstanding occlusion queries for the diamonds in the split and merge queues are invalidated. All diamonds in the split and merge queues are marked as visible and the new isosurface is extracted from the diamonds in the split queue. The depth buffer given by the new isosurface can now be used for occlusion queries, and the refinement continues as described above in Section 4. As new error values are computed and new occluded regions are discovered, the mesh structure refines around the visible region of the new isosurface and coarsens everywhere else.

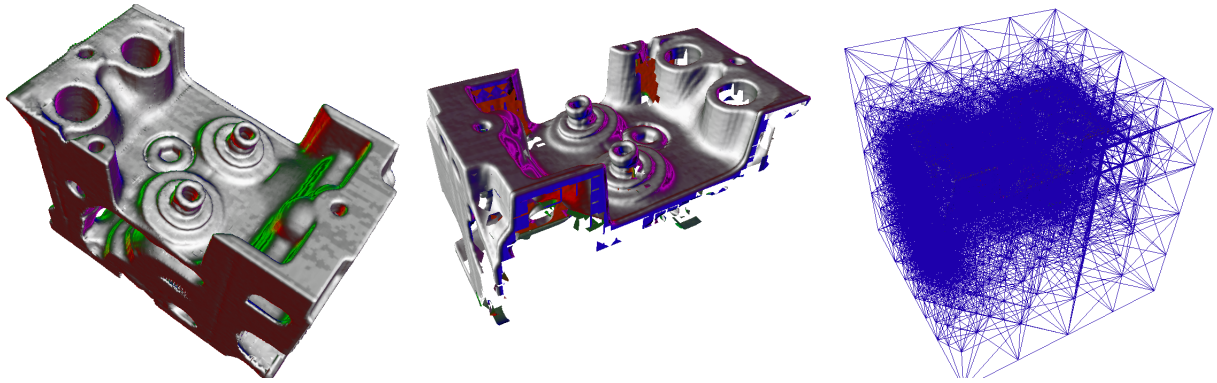

Figure 5: *Left: high resolution rendering of the engine dataset, Isovalue = 100, Error = 0.6. The mesh contains 1.23 million triangles. Middle: Backside view showing the parts of the dataset removed by occlusion culling; the mesh contains 680K triangles. Right: The tetrahedral mesh showing that the occluded areas (lower right) are at a lower resolution.*

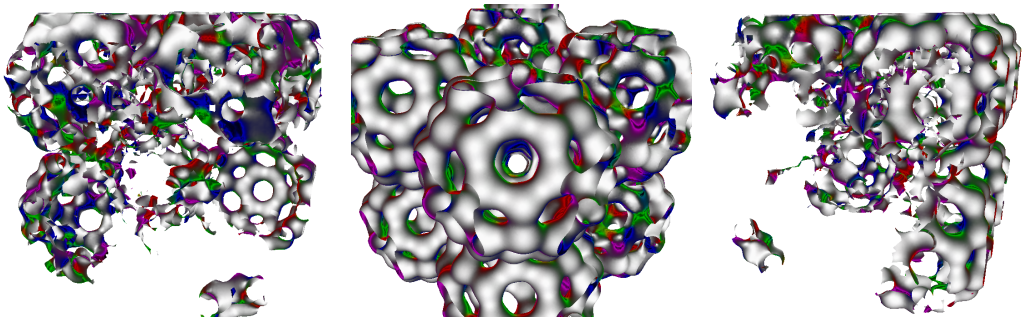

Figure 6: *High resolution rendering of the* 1024^3 *synthetic buckball dataset. Left and Right: Backside views showing that the occluded areas have been removed. Middle: The surface rendered with occlusion culling, Isovalue = 104, Error = 0.71. Occlusion culling results are given in Table 2.*

5. Results

Our test machine is a 2 Ghz Pentium with 1 GB of main memory and a GeForce4 Ti 4600 graphics card. The amount of main memory available for uncompressed data was limited to the lesser of 512 MB or $(1/2)$ of the uncompressed dataset size. We have tested our algorithms on several volume datasets. The buckyball dataset is a synthetic dataset made from Gaussian functions. The 1024^3 dataset was made by a $2 \times 2 \times 2$ tiling of a 512^3 dataset. The PPM (Piecewise Parabolic Method) dataset is a Richtmyer-Meshkov simulation dataset [MCC*99]. For the Christmas Tree CT dataset [KTM*02], we are using the high resolution version of the near isotropic and rotated dataset. Its initial dimensions are $512 \times 512 \times 499$, and it has been padded with zeros to make it 512^3. As stated by Lee et al. in [LDS03], the extension and padding of the dataset comes with a small increase in entropy. All isosurfaces were extracted from the compressed volumes.

Each dataset includes the actual data and three bytes for a quantized gradient, one byte for each component. The four values are predicted independently, and one encoding table is used for compression. Tests using separate encoding tables for each component or separate tables for the data and gradients did not yield significant improvements in compression. Before compression, the raw floating point gradients are run through a diffusion based smoothing process to ensure that there are no degenerate gradients that can cause problems for shading. Figure 9 shows histograms of the raw and transformed data for several test datasets. The transformed data is the set of delta values generated by the prediction step (Section 3.1). As the histograms show, the data transformation greatly increases the number of values with small magnitudes and reduces the number of values with large magnitudes, indicating that the transformed data will compress better.

Table 1 summarizes the performance of our compression algorithm. For each dataset, we show the compression of the data and gradients as well as the compression of just the data. In general, the data compresses much better than the gradients since the gradients, even after our smoothing pro-

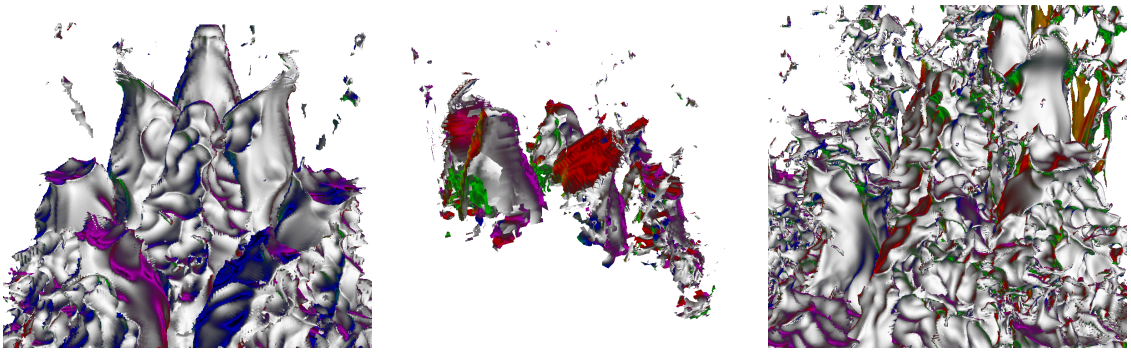

Figure 7: *Left: High resolution isosurface from the PPM dataset. Isovalue = 228, Error = 0.78. Middle: A backside view of the left image showing the occluded regions. Right: Another isosurface from the PPM dataset. Isovalue = 200, Error = 0.638. Results with and without occlusion culling are given in Table 2.*

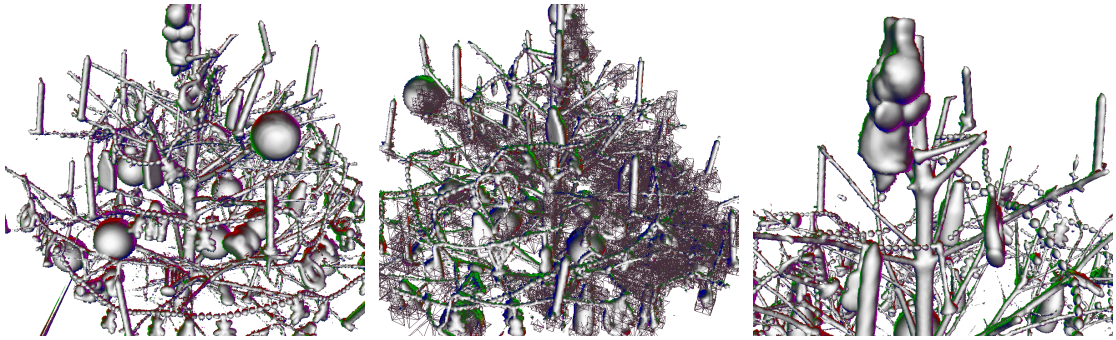

Figure 8: *Isosurfaces from the Christmas Tree dataset. The large number of thin features in the surface, caused by the needles, branches, and tinsel, make occlusion culling hard for these isosurfaces. Left: Isovalue = 175, Error = 0.50. Middle: A backside view of the left image showing the occluded diamonds in the split queue. This indicates where occluded triangles have been removed. Right: Isovalue = 300, Error = 0.496. Results with and without occlusion culling are given in Table 2.*

cess, are still very noisy. For several test datasets, we are able to achieve lossless compression ratios between 2:1 and 5:1 and 50% compression of the gradients using the simple split-edge predictor and lead-1 encoding with Huffman codes. To test the speed of our runtime decompression and reconstruction algorithms, we recorded measurements over several interaction sessions in which the viewing position, isovalue, and error level were all changed at some point. To measure the decompresion speed, we recorded the time to decompress a single page. This takes k input bytes, where k is the size of the compressed page, and produces 4×2^{12} deltas or 32768 bytes. This does not included the time to load data from disk. Over several test runs the decompression speed ranged from $10-100$ MB/sec with the average decompression speed between $50-70$ MB/sec. The decoder's speed is $O(n)$ where n is the number of output values. It is affected mainly by memory coherence of the input and output data arrays. The reconstruction rate was measured as the time to compute the actual data and gradients using the deltas. This includes the time to invert the z-order indices, compute the vertices needed for reconstruction and fetch their associated data and gradients. The reconstruction speed varied from $1-12$ MB/sec with the average rate between $8-10$ MB/sec. The reconstruction speed is affected by the index conversion and the cost of fetching the needed data needed. This requires two memory accesses per data point or a total of 2^{13} accesses to reconstruct one page. Furthermore, it can cause recursive decompression when needed values must be reconstructed themselves.

Results for occlusion culling are summarized in Table 2. It shows the number of triangles, tetrahedra, and diamonds in the mesh for surfaces extracted with and without occlusion culling. It also shows the number of occluded diamonds in the split queue, and the total number of occlusion queries performed per frame. This includes queries on diamonds in the merge and split queues. Occluded diamonds exist in the split queue because of splits necessary to maintain the mesh continuity and to meet the given error threshold. However, contouring is never performed on these occluded diamonds. Our occlusion method is conservative because we allow vis-

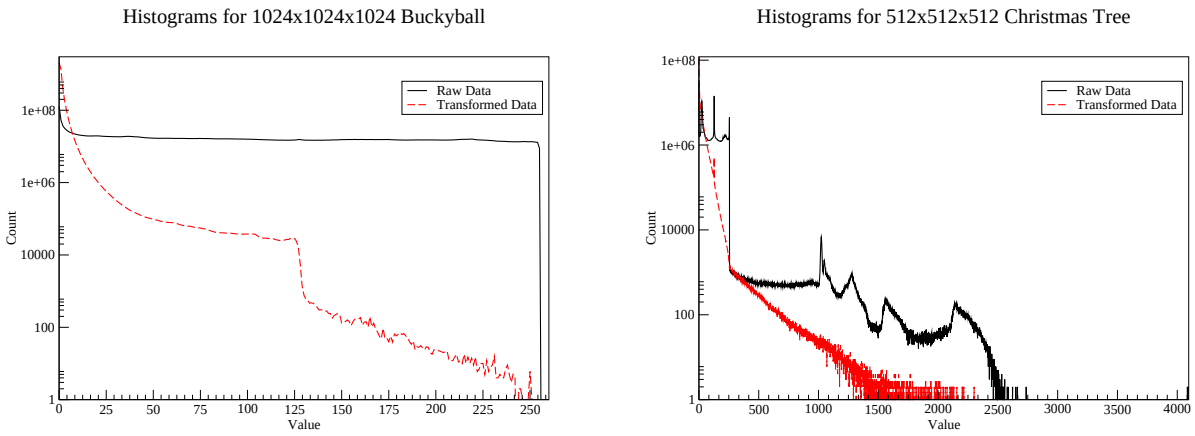

Figure 9: *Histograms of raw and transformed datasets (data and gradients). Left: Synthetic buckyball dataset. Right: Christmas Tree dataset. The buckball dataset is 8-bit data, and the Christmas Tree dataset is 12-bit data represented in 16-bit unsigned shorts.*

ible regions to remain visible for several frames. Conservative occlusion culling greatly reduces the number of occlusion tests performed per frame and randomizes at which frame the queries are performed, allowing us to maintain interactive frame rates.

Figures 5 and 6 show the benefits of occlusion culling on the engine dataset and the buckball dataset. The isosurfaces are extracted at a high resolution and thus contain a large number of occluded triangles. In most areas, the engine has a depth complexity of 2-3, and the buckball has depth complexity of 4-6. For these datasets, occlusion culling greatly reduces the number of triangles, tetrahedra, and diamonds. A reduction in triangle count yields a reduction in rendering time and storage space for the triangles. The reduction in mesh size reduces the number of error calculations that need to be performed and the number of data pages that need to be decompressed; both of which improve performance.

Figure 7 shows two high resolution isosurfaces extracted from different view points and isovalues from a 512^3 chunk of the PPM dataset. These isosurfaces are much more complex than the engine and buckball isosurfaces, they have high depth complexity, and they contain a large number of very small features. As shown in Table 2, for the PPM dataset occlusion culling reduces the number of extracted triangles from 30-50% and the size of the mesh by 20-30%. For the Christmas Tree dataset shown in Figure 8, occlusion culling is not as useful because the isosurfaces contain a large number of very thin features and fewer large features which are good occluders. As the results indicate, occlusion culling is able to remove some triangles, but it is not nearly as effective as it is for the other datasets.

6. Conclusions and Future Work

In conclusion, we have presented new data compression and occlusion culling algorithms for fast isosurface extraction from massive datasets. We utilize a hierarchical tetrahedral mesh structure based on longest-edge bisection to form an adaptive volume representation. The adaptive representation is the basis for a hierarchical prediction technique used to compress the volume data and its precomputed, quantized gradients. A fast coder based on lead-1 encoding and Huffman codes achieves 2:1 to 5:1 compression of the volume data and 50% compression of the gradients. Occlusion culling, performed using the diamond representation of the mesh, allows unseen portions of the volume to be culled and simplified, thus reducing the number of rendered triangles and runtime computation.

Our future work is focused on developing new predictors, better methods for quantizing and compressing the gradients, and new shading techniques. Since the accuracy of the predictor can greatly improve the compression performance, new predictors that can further reduce the entropy without a large computation cost are desired. While our technique provides good lossless compression for the data, the compression of the quantized gradients is not nearly as good. New techniques for predicting the gradients for compression and quantizing the gradients for accurate prediction are needed to improve the compression performance. Additionally lossy compression techniques can be used for gradient compresion.

The compression of the gradients is necessary because they are needed for shading. New shading techniques, based on other surface and volume properties such as curvature and second derivatives, that do not require gradients or can use

Dataset	Triangles	Tetrahedra	Diamonds	Occluded Diamonds	Occlusion Tests Per Frame
PPM (Left)	2.5M	306K	128K	–	–
	1.02M	206K	96K	16K	10-12K
PPM (Right)	2.9M	400K	175K	–	–
	2.0M	312K	143K	22K	16-17K
XMasTree (Left)	1.6M	322K	153K	–	–
	1.45M	309	146K	5K	8-10K
XMasTree (Right)	1.06M	246K	124K	–	–
	893K	234K	119K	7K	7-9K
Buckyball	2.24M	254K	102K	–	–
	1.23M	183K	82K	17K	10-14K

Table 2: *Occlusion culling performance for various test datasets. Top row shows values without occlusion culling and the bottom row shows values with occlusion culling. For the PPM and Christmas Tree dataset, left and right refer to the two surfaces in Figures 7 and 8.*

lower quality gradients are needed to improve the quality of the visualization and reduce the gradient's storage overhead.

Acknowledgements

The Christmas Tree dataset was obtained from Department of Radiology, University of Vienna and the Institue of Computer Graphics and Algorithms, Vienna University of Technology. The engine dataset was obtained from http://www.volvis.org. The buckyball dataset was obtained from Oliver Kreylos at UC Davis.

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. This work was supported by the National Science Foundation under contracts ACR 9982251 and ACR 0222909, through the National Partnership for Advanced Computing Infrastructure (NPACI) and by Lawrence Livermore National Laboratory under contract B523818. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

[DWS*97] DUCHAINEAU M. A., WOLINSKY M., SIGETI D. E., MILLER M. C., ALDRICH C., MINEEV-WEINSTEIN M. B.: ROAMing Terrain: Real-time Optimally Adapting Meshes. In *IEEE Visualization '97* (Oct. 1997), IEEE Computer Society Press, pp. 81–88.

[ESSS01] EL-SANA J., SOKOLOVSKY N., SILVA C.: Integrating occlusion culling with view-dependent rendering. *Proc. of IEEE Visualization* (2001).

[GDL*02] GREGORSKI B., DUCHAINEAU M. A., LINDSTROM P., PASCUCCI V., JOY K. I.: Interactive view-dependent rendering of large isosurfaces. In *Proceedings of the IEEE Visualization 2002* (2002).

[Ger01] GERSTNER T.: Fast Multiresolution Extraction Of Multiple Transparent Isosurfaces. In *In Data Visualization 2001 Proceedings of VisSim 2001* (2001), David S. Ebert Jean M. Favre R. P., (Ed.), Annual Conference Series, Springer-Verlag.

[GHSK03] GAO J., HUANG J., SHEN H.-W., KOHL J. A.: Visibility culling using plenoptic opacity functions for large volume visualization. In *Proceedings of IEEE Visualization 2003* (2003), pp. 341–348.

[GLY*03] GOVINDARAJU N. K., LLOYD B., YOON S.-E., SUD A., MANOCHA D.: Interactive shadow generation in complex environments. In *Proceedings of SIGGRAPH 2003* (2003), pp. 501–510.

[GP00] GERSTNER T., PAJAROLA R.: Topology Preserving And Controlled Topology Simplifying Multiresolution Isosurface Extraction. In *Proceedings of IEEE Visualization 2000* (2000), Ertl T., Hamann B.„ Varshney A., (Eds.), IEEE Computer Society Press, pp. 259–266.

[GS01] GUTHE S., STASER W.: Real-time decompression and visualization of animated volume data. In *Proceedings of IEEE Visualization 2001* (2001), pp. 349–358.

[GWGS02] GUTHE S., WAND M., GONSER J., STASER W.: Interactive rendering of large volume data

sets. In *Proceedings of IEEE Visualization 2002* (2002), pp. 53–60.

[Ha03] HA H.: *Out-of-core Interactive Display of Large Meshes Using an Oriented Bounding Box-based Hardware Depth Query*. Master's thesis, University of California, Davis, September 2003. Available as Department of Computer Science Technical Report CSE-2003-25.

[HSLM02] HILLESLAND K., SALOMON B., LASTRA A., MANOCHA D.: *Fast and Simple Occlusion Culling using Hardware-Based Depth Queries*. Tech. Rep. UNC-CH-TR02-039, Computer Science Department, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 2002.

[KTM*02] KANITSAR A., THEUSSL T., MROZ L., SRÁMEK M., BARTROLÍ A. V., CSÉBFALVI B., HLADÙVKA J., FLEISCHMANN D., KNAPP M., WEGENKITTL R., FELKEL P., GUTHE S., PURGATHOFER W., GRÖLLER M. E.: Christmas tree case study: Computed tomography as a tool for mastering complex real world objects with applications in computer graphics. In *Proceedings of the 13th IEEE Visualization 2002 Conference (VIS-02)* (Piscataway, NJ, Oct. 27– Nov. 1 2002), Moorhead R., Gross M.„ Joy K. I., (Eds.), IEEE Computer Society, pp. 489–492.

[LDS03] LEE H., DESBRUN M., SCHRODER P.: Progressive encoding of complex isosurfaces. In *Proceedings of SIGGRAPH 2003* (2003), pp. 471–476.

[LGP*03] LINSEN L., GRAY J. T., PASCUCCI V., DUCHAINEAU M. A., HAMANN B., JOY K. I.: Hierarchical large-scale volume representation with '3rd-root-of-2' subdivision and trivariate b-spline wavelets. In *Geometric Modeling for Scientific Visualization*. Springer-Verlag, Heidelberg, Germany, 2003.

[LH98] LIVNAT Y., HANSEN C.: View Dependent Isosurface Extraction. In *Proceedings of IEEE Visualization 1998* (1998), Ebert D., Hagen H.„ Rushmeier H., (Eds.), IEEE Computer Society Press, pp. 175–180.

[LMC01] LUM E., MA K.-L., CLYNE J.: Texture hardware assisted rendering of time-varying volume data. In *Proceedings of IEEE Visualization 2001* (2001), pp. 263–270.

[LMK03] LI W., MUELLER K., KAUFMAN A.: Empty space skippping and occlusion clippinf for texture-based volume rendering. In *Proceedings of IEEE Visualization 2003* (2003), pp. 317–326.

[LP01] LINDSTROM P., PASCUCCI V.: Visualization of large terrains made easy. In *Proceedings of IEEE Visualization 2001* (2001), Ertl T., Joy K.„ Varshney A., (Eds.), IEEE Computer Society Press, pp. 363–370.

[MCC*99] MIRIN A. A., COHEN R. H., CURTIS B. C., DANNEVIK W. P., DIMITS A. M., DUCHAINEAU M. A., ELIASON D. E., SCHIKORE D. R., ANDERSON S. E., PORTER D. H., , WOODWARD P. R.: Very High Resolution Simulation Of Compressible Turbulence On The IBM-SP System. In *Proceedings of SuperComputing 1999* (1999), (Also available as Lawrence Livermore National Laboratory technical report UCRL-MI-134237).

[Pas02] PASCUCCI V.: Multi-Resolution Indexing For Out-Of-Core Adaptive Traversal Of Regular Grids. In *Proceedings of the NSF/DoE Lake Tahoe Workshop on Hierarchical Approximation and Geometric Methods for Scientific Visualization* (2002), Farin G., Hagen H.„ Hamann B., (Eds.), Springer-Verlag, Berlin, Germany, (to appear). (Available as LLNL technical report UCRL-JC-140581).

[Wes03] WESTERMANN R.: Compression domain volume rendering. In *Proceedings of IEEE Visualization 2003* (2003), pp. 293–300.

[WKE99] WESTERMANN R., KOBBELT L., ERTL T.: Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. In *The Visual Computer* (1999), pp. 100–111.

[XZ02] XIAOYU ZHANG CHANDRAJIT BAJAJ V. R.: Paralel And Out-Of-Core View-Dependent Isocontour Visualization. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualizatation (VisSym-02)* (Vienna, Austria, May 27–29 2002), Ebert D., Brunet P.„ Navaz I., (Eds.), Springer-Verlag.

[YSM03] YOON S.-E., SALOMON B., MANOCHA D.: Interactive view-dependent rendering with conservative occlusion culling in complex environments. In *Proceedings of IEEE Visualization 2003* (2003).

[ZMHI97] ZHANG H., MANOCHA D., HUDSON T., III K. H.: Visibility culling using hierarchical occlusion maps. *Proceedings of SIGGRAPH* (August 1997), 77–88.