Preprint
UCRL-CONF-201180

Compression and Occlusion for Fast Isosurface Extraction from Massive Datasets

Benjamin Gregorski, Joshua Senecal, Mark Duchaineau, and Kenneth I. Joy

This article was submitted to Submitted to IEEE Visualization 2004

Compression and Occlusion for Fast Isosurface Extraction from Massive Datasets

Benjamin Gregorski,[1,2*] Joshua Senecal[1,2] Mark Duchaineau[1]
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

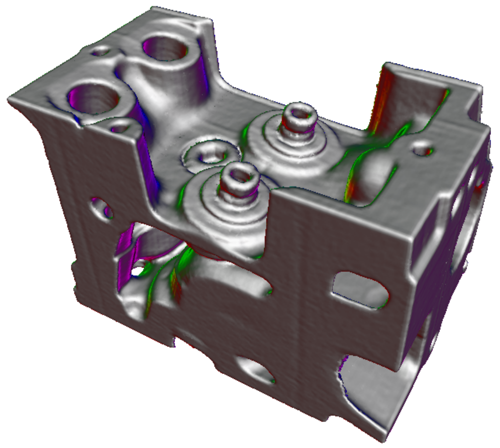
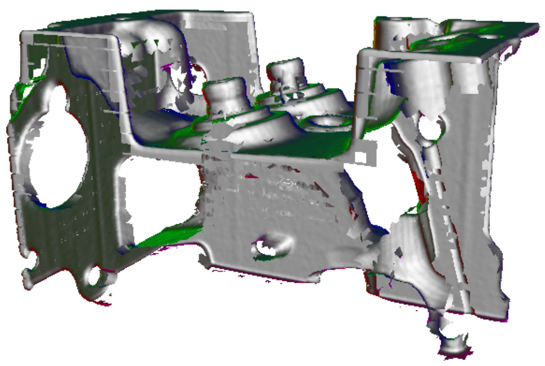



Figure 1: Left: high resolution rendering of an isosurface from the engine dataset: Isovalue = 85, Error = 0.42. Without occlusion culling the mesh contains 98K diamonds, the isosurface contains 1.90M triangles and it renders at 6 FPS. Right: A backside view shows the regions removed using occlusion culling; Occlusion culling reduces the mesh to 51K diamonds, the isosurface to 553K triangles, and increases the rendering speed to 10 FPS. At each frame, 7-8K occlusion tests are performed and 2-3K diamonds are found to be occluded.

Abstract

We present two algorithms for data compression and occlusion culling that improve 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 at interactive rates from losslessly compressed volumes where the region of interest, determined at runtime by user interaction, is decompressed *on-the-fly*. For interactive applications, we exploit frame-to-frame coherence between consecutive views to simplify the mesh structure in occluded regions and eliminate occluded triangles significantly reducing the complexity of the visualized surface and the underlying multiresolution volume representation. 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.

CR Categories: I.3.3 [Computer Graphics]: Line and Curve Generation— [I.3.3]: Computer Graphics—Viewing Algorithms

Keywords: ROAM, Large Data Visualization, Interactive Multiresolution Techniques, Isosurface Extraction

[*]gregorski1,senecal1,duchaineau1@llnl.gov,kijoy@ucdavis.edu

1 Introduction

Isosurface extraction from rectilinear datasets is a well-known problem and significant research has been done to improve the quality of the extracted isosurface, see [30, 25]. Unfortunately, the complexity of these isosurfaces varies considerably and somewhat independently of the original volume's size. In Duchaineau et al. [2], isosurfaces extracted from a single timestep of a large fluid dynamics dataset contained 460 million triangles and exceeded the original volume's storage requirements. Additionally, the visual complexity of the isosurface was very high with as many as 50 occluded surfaces per screen space pixel. The size and complexity of this surface makes it virtually impossible to interactively visualize by first extracting the isosurface and then rendering the entire triangle mesh. Furthermore, most volume datasets contain several isosurfaces of interest and extracting the triangle mesh for each isosurface and then rendering it is simply not practical.

One solution to this visualization problem is to directly extract the isosurfaces from a volume in an *on-demand* fashion using multiresolution volumetric techniques that allow visualization quality and interactivity to be balanced efficiently. Gregorski et al. [10] developed a view-dependent isosurface extraction algorithm based on longest-edge bisection (LEB) of tetrahedral meshes that allows the extraction of arbitrary isosurfaces. These LEB meshes have been used extensively in scientific visualization for adaptively reconstructing ultrasound data [32], performing isocontouring with controlled topology simplification [8] and rendering multiple transparent and opaque isosurfaces [7]. One can effectively deal with large datasets through the use of a hierarchical data layout based on space filling curves [10, 21, 31]. This method, however, provides only a partial solution because it does not address the problems of storing these large volumes and effectively rendering extremely complex isosurfaces with high depth complexity.

Visualization of massive isosurfaces is still difficult because of the large storage requirements of the volumes from which they are extracted. Widely available compression libraries such as gzip and bzip are computationally too expensive to be used for these interactive applications. Furthermore, traditional hierarchical compression algorithms based on wavelet decompositions update data values as the level of detail changes forcing the isosurface to be reextracted from a larger number of elements, thus increasing the cost of the visualization. In order to be effective for interactive applications, we need a decompression algorithm that is fast, follows the hierarchy given by the mesh refinement to leverage memory coherence, and is 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.

In addition, isosurfaces extracted from massive volumes contain millions of triangles and almost always have a depth complexity far greater than one, i.e., a large number of occluded triangles, triangles within the view frustum that do not affect the final image, are extracted and rendered. In general, a pixel on the screen will have a large number of fragments rendered to it even though many of those fragments are invisible. 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. It is very wasteful of CPU resources because the tetrahedral mesh is refined in the occluded regions, increasing the size of the runtime data structures and the amount of data that must be loaded from disk. It is also very wasteful of GPU resources because more triangles must be stored in GPU memory for faster rendering and a larger number of invisible triangles are sent through the graphics pipeline. In order to effectively visualize these large surfaces, the occluded regions must be detected and culled from the visualization process at a very coarse level allowing the mesh structure to be simplified in these regions. Occlusion culling prevents these invisible triangles from being extracted and rendered. 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.

Compression techniques for interactive visualization have been utilized in the context of volume rendering by Guthe et al. [13], where wavelets and mpeg style encoding are used to compress time-varying volumes, and by Lum et al. [26] where quantization of DCT coefficients is used. Westermann et al. [35] use octrees to perform adaptive isosurface extraction fixing cracks between transition levels with triangle fans. Guthe and Strasser [14] utilize wavelet based compression to adaptively render volume datasets using texture hardware. In their method, 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 [17] utilize a hierarchical vector quantization scheme for compressing static and time-varying volumes. The dataset is divided into blocks of 4^3 elements and each block is encoded hierarchically. A multiresolution representation is created by performing this recursively on groups of downsampled blocks and producing codebooks for the different levels of resolution. Their algorithm can reconstruct the data on the graphics card removing the overhead of CPU decompression.

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. El-Sana et al. [4], divided a dataset against a grid and a determine a solidity value 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. The hierarchical occlusion map developed by Zhang et al. [38] 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 feature has been used extensively for occlusion culling in large polygonal environments, see [9, 37, 16, 15].

Occlusion culling methods have also been extended to isosurface extraction from volume datasets and to volume rendering applications. Livnat and Hansen [24] traverse an 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 Zhang et al. [39], 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. Isosurfaces can also be visualized directly from volume datasets using direct volume rendering techniques such as those by Lum et al. [27] and [28]. The advantage of these algorithms is that they do not require the overhead of multiresolution data structures and space for storing explicit isosurface geometry. Their disadvantage is that they are fill-limited and their performance is directly related to the size of the rendered image and available texture memory. We extract isosurfaces directly from the tetrahedral mesh and render explict geometry. This can provide a better balance between the geometry and fill sections of the graphics pipeline. Recent work by Li et al. [20] and Gao et al.[5] has extended occlusion culling techniques to volume rendering applications where transparent and opaque surfaces are rendered.

This paper presents a fast and efficient data compression algorithm that follows the tetrahedral mesh hierarchy defined by LEB. The cubical blocks of an octree decomposition frequently used in the data compression algorithms, are replaced with the diamond shapes developed in Gregorski et al. [10] and used by Linsen et al. [22] for hierarchical representation of large datasets. These meshes have the advantage that the number of elements is increased by a factor of 2 with each refinement rather that the factor of 8 associated with octree refinement allowing more levels of detail in the multiresolution representation. A top-down traversal of the mesh hierarchy uses the data values and gradient components at the vertices of a refinement edge to predict the data and gradient components at the newly inserted vertex for lossless compression of large volume datasets. In general, the magnitudes of these predicted values are smaller that the original magnitudes which means that the resulting data has reduced entropy and is easier to compress. This prediction scheme follows the data access pattern dictated by LEB refinement. As new vertices are introduced by mesh refinement, the associated delta values are loaded from disk and the vertices' data is reconstructed. Combined with the z-order data layout scheme of Pascucci et al. [31], it ensures that 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 massive datasets. The delta values used for runtime reconstruction are compressed in chunks and stored in z-order. Hierarchy levels 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).

The occlusion culling algorithm (Section 4) uses the diamonds of the mesh hierarchy to determine occluded regions. This allows occlusion culling to be utilized regardless of the isosurface being visualized. Since diamonds are also the unit of refinement, it is straightforward to determine how the mesh must be refined or simplified as regions become visible or occluded. As the diamond hierarchy does not have a nested parent-child relationship that allows for front-to-back rendering, we utilize frame-to-frame coherence between successive view-points to create an approximate depth buffer against which hardware occlusion tests are performed. Once the visible and occluded regions have been determined, the mesh is dynamically refined and coarsened accordingly. Results for compression and occlusion culling are given in Section 5.

2 Mesh Refinement and Preprocessing

In this section we review LEB mesh refinement [10] and its relationship with the z-order space filling curve [31]. The mesh refinement begins with the root diamond: a cube divided into six tetrahedra around its major diagonal.[1] Refinement occurs around cube diagonals, face diagonals, and edges of the mesh. Figure 2 shows the three types of polyhedral shapes or diamonds that occur around the diagonals and edges of the mesh. The basic unit of refinement is the diamond. All tetrahedra in a diamond are refined at the same time by inserting a new vertex call the *split vertex* at the midpoint of the diamond's *split edge* which is the edge shared by all tetrahedra in the diamond. See [10] for a complete description of this procedure.

Split and *merge* operations are used to refine and coarsen the mesh. When a diamond is split, the split vertex is inserted and each tetrahedron in the diamond is split into two child tetrahedra. Splitting and merging diamonds ensures that the mesh is always crack-free. The split/merge process is implemented with two error-based priority queues [3, 10], 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 LEB refinement and the z-order space filling curve. Each (i, j, k) index of point P in the dataset corresponds to a $z-order$ index on the one dimensional z-order curve. Conversion between (i, j, k) and $z-order$ indices is straightforward and given in Pascucci et al. [31]. Note that each data point, except for the points at the vertices of the root diamond, is the split vertex of a diamond. The order of the points introduced by each level of mesh refinement is essentially the same as that 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. This is not a problem because the z-order curve tiles the dataset in all spatial dimensions causing an index I to map to I mod 2^k. Thus index 2^k on the mesh boundary maps to index 0. The correspondence between the mesh hierarchy and the z-order curve gives excellent disk and memory coherence making them well suited for adaptive, out-of-core visualization of large datasets.

The following information Gregorski et al. [10] compute in a preprocessing step:

[1]In this paper we deal only with rectilinear meshes, however, the actual physical positions of the vertices does not affect the order of refinement.

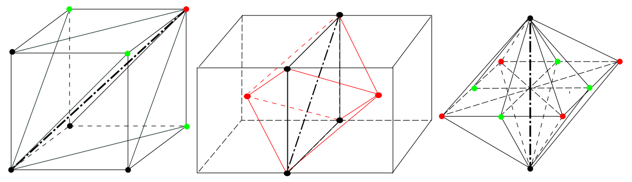


Figure 2: From left to right: diamond shapes for phases two, one, and zero of the mesh refinement. The split or refinement edge is the bold, circle-segment dashed line. The diamonds occur around cube diagonals, face diagonals and edges of the mesh, respectively.

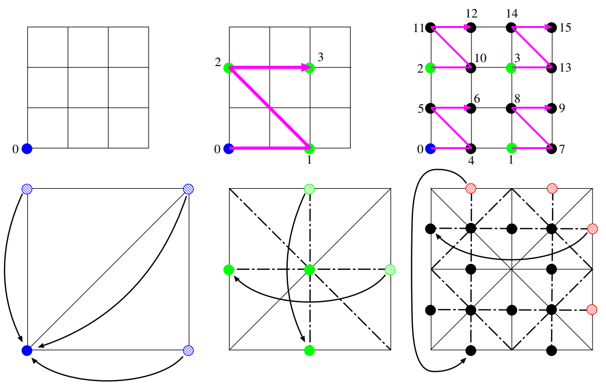


Figure 3: Two-dimensional example of hierarchical z-order and mesh refinement for a 4×4 dataset. Top, from left to right: levels 0(blue), 1(green), and 2(black) of the hierarchical z-order curve showing the points introduced at each level and their order on the one-dimensional curve. Bottom, left to right: levels 0, 1, and 2 of the LEB mesh. The solid circles indicate the data points introduced by the LEB refinement at each hierarchy level. The dashed circles and arrows show how mesh points with at least one index equal to (2^k) are wrapped around the dataset to where the index equals 0. The data points added at each level of the z-order curve are the same points added at each level of the mesh refinement.

1. The isosurface approximation error, minimum data value, and maximum data value of the region enclosed by each diamond in the mesh.

2. The normalized gradient vector at each data point. The gradient is used as a texture coordinate to perform diffuse lighting and to highlight the regions orthogonal to the viewing direction.

For a tetrahedron T, the isosurface approximation error is the maximum difference between the isosurface generated using the scalar values at T's vertices from the true isosurface passing through T. The storage required for the minimum, maximum, and error values is reduced by grouping sets of tetrahedra into chunks of 64 tetrahedra called subtrees, see [12]. This reduces the overall error of a tetrahedron, the granularity of the mesh refinement and the size of the runtime data structures. All tetrahedra in a subtree are contoured at the same time. Grouping tetrahedra into subtrees of 64 reduces the storage requirements of the minimum, maximum, and error values are reduced by a factor of 64 and allows them to be stored uncompressed. For a 1024^3 byte dataset, the size of these values is reduced from 3 GB to 50 MB or about 1.6% of the size. This reduced storage comes at the cost of an increased number of tetrahedra that do not contain the isosurface being contoured. Alternatively, these precomputed values can be compressed by the algorithm of Bordoloi et al. [1], where transform coding is used to compress cell minimum and maximum values. Their algorithm, however, still extracts the finest level isosurface and does not compress the original data or the cell id numbers needed for their algorithm, both of which require significant storage. The data compression algorithm described in Section 3 is used for the data values and gradient components. Gradients are precomputed because it is expensive to compute gradients via differencing at runtime due to adjacent data points being far away on the z-order curve.

3 Data Compression

Hierarchical prediction algorithms, such as wavelet transforms based on octrees [14], the $\sqrt{2}$ meshes [22] or the **JPEG 2000** algorithm, traverse a dataset in a fine-to-coarse manner building successive, filtered approximations of the original data. One problem with using these wavelet based techniques in multiresolution visualization is that the function values at vertices change as the vertex moves between levels in the wavelet hierarchy. When a vertex V is added, function values at the vertices needed to reconstruct the value at V must be updated before the value at V is computed. The isosurface must be reextracted from the elements that use updated values otherwise cracks will occur. Similarly, as regions of the mesh are coarsened, function values must be updated to the appropriate level of the hierarchy. This increases the amount of work that must be done in each frame and makes it more difficult to maintain consistent frame rates as the level-of-detail is changed. The alternative is subsampled hierarchies. Here the values do not change because the coarser resolutions are created by selecting a reduced set of individual data points from the finer resolution without considering neighboring values. We use hierarchies based on subsampling and exploit their static properties so that the error bound can be quickly tightened and the isosurface can be extracted from regions of interest at a high resolution.

Our data compression algorithm is divided into three phases. In the first phase, the original volume is traversed along the LEB hierarchy and the data and gradient component values at a diamond's split vertex are predicted using difference from linear along a diamond's split edge. In the second phase, these delta values are passed to an encoder which builds a set of codes used for compression. In the final phase, the stream of delta values is divided into pages and compressed using the codes generated in phase two.

For multiresolution visualization, it is important to use hierarchical prediction that follows the refinement of the multiresolution mesh. This ensures that the data needed for reconstruction can be efficiently located at runtime. Decompression is localized across the space and scale of the volume hierarchy. Decompressing a region at a high resolution occurs in a small localized space around the region and in similarly localized spaces at coarser levels of the hierarchy. In this paper we consider selective, local refinement applications, such as view-dependent refinement, where the important and needed regions change at the user's discretion and must be located immediately without having to reconstruct the whole volume. Thus, the encoded data stream must not only be progressive with respect to the overall quality of the dataset but also must allow random access to spatially coherent blocks located across the levels of the hierarchy.

Embedded coders and bit-plane coders such as zero-tree coding [34] and SPIHT [33] provide good compression and excellent bitrates and allow for progressive transmission, but they are not well suited for selective, local refinements where only a subregion of the original data is needed. This is because they encode the most important wavelet coefficients first and the least important ones last; the goal being to represent the whole dataset as accurately as possible with the fewest number of bits. However, in view-dependent and region-of-interest refinement, the most important coefficients and the needed coefficients change and are not known until runtime. Thus, we cannot use an algorithm that requires a globally fixed ordering and must instead use an algorithm that follows the mesh hierarchy's local refinement. Data prediction using data values along the split edges accomplishes this while maintaining small reconstruction masks necessary for good performance.

3.1 Compression and Decompression

Our data prediction algorithm, based on difference from linear prediction along a diamond's split edge, is illustrated for two-dimensional refinement in Figure 4. If D_{SV_0} and D_{SV_1} are the data values at the vertices of the split edge, then we define the predicted value to be

$$D_p = \frac{(D_{SV_0} + D_{SV_1})}{2}, \qquad (1)$$

and encoded delta value is

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

In three dimensions, the predicted value is also computed by averaging the values at the split edge's vertices. 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. Thus, there are no special cases. Figure 8 shows histograms of the data before and after prediction.

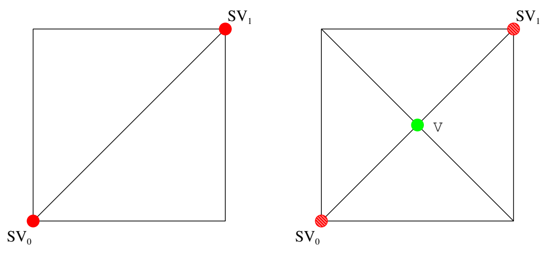


Figure 4: The value at a vertex V, which is also the split vertex of a diamond, is predicted as the average of the values at its split edge vertices SV_0 and SV_1.

After the initial data prediction transformation, the original data has been converted to a representation that is easier to compress since the transformed data's coefficient magnitudes are clustered toward smaller values than those of the original data. Smaller magnitudes indicate that the position of the leading 1, in the binary representation of the data, is in a lower order bit. Thus, there are a larger runs of zeros in the most significant bits of the coefficients indicating that the transformed data has more redundancy. For example, in an 8-bit transform coefficient, such as 00000101, the high-order bits 000001 are more compressible than the remaining bits 01. [2]

To exploit this property of the transformed data, we utilize *lead-1 encoding*. If the transformed data values require 8 bits for representation (signed data), there are 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

[2]Only lossless compression is considered since the datasets we are dealing with have already been quantized in some manner.

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 other types of coders such as arithmetic and arithmetic-type encoders, lead-1 encoding offers an acceptable compression ratio with superior speed and throughput.

The data being compressed is the data value and quantized gradient at each vertex. The gradients are quantized in three bytes with one byte per component. The transformed coefficients are stored in hierarchical z-order. To allow for local decompression, the transformed coefficients are divided into pages and each page is encoded 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.

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. Accessing the data at the split edge vertices needed for reconstruction, from memory and disk, exploits these coherence properties of the storage order. Compression results for test datasets are given in Table 1.

4 Occlusion Culling

Complex isosurfaces extracted from volume datasets have high depth complexity and millions of occluded triangles. This makes interactive visualization more difficult because a significant amount of time is spent processing data that does not contribute to the final image. Since a dataset usually contains a large number of interesting isosurfaces whose topology and geometry can vary greatly with level-of-detail and isovalue, our occlusion culling algorithm is based upon the mesh hierarchy and not the extracted isosurface geometry.

Occlusion queries on modern graphics hardware allow one to render geometry and determine the number of samples that pass the depth test. 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 will be modified if the given geometry is rendered. There is a noticeable delay between the time when an occlusion query is issued and the time when the results are available since the query geometry must be rendered and the query results read back from the graphics card. One of the keys to efficiently utilizing hardware occlusion queries is to fill this time gap with useful computations. In order to minimize this delay, we issue several queries and perform speculative error calculations for the diamonds in the split and merge queues before reading the occlusion query results.

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 regions with a single occlusion test. Since a diamond's children are not contained within its convex hull, the occlusion queries cannot be performed top-down. Furthermore, since we only maintain a hierarchy of diamonds and do not maintain a hierarchy of the tetrahedra in the mesh, we cannot render the isosurface front-to-back to generate the depth buffer for the occlusion queries. Instead we exploit the frame-to-frame coherence of the isosurface between successive view points to generate a good set of occluders from which a depth buffer for the occlusion queries is created.

Our occlusion culling algorithm works as follows:

1. To exploit frame-to-frame coherence between successive views, at the start of a frame f_i, we render the isosurface from frame f_{i-1} to generate the initial depth buffer for the occlusion queries. This provides a good initial guess for the set of occluders and gives the correct depth buffer for them. The occlusion queries in frame f_i are performed against this depth buffer.

2. An occlusion query is then initiated for all diamonds in the split and merge queues 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.

3. After all of the queries have been issued, we read back the results of the occlusion queries in the order that they were executed. 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.

4. The triangles extracted from the tetrahedra added to the mesh in frame f_i are rendered to fill the holes in the image caused by rendering f_{i-1}'s isosurface at f_i's viewpoint. Since a limited amount of time is allocated to each frame, it is possible that

Dataset	Uncomp	Comp	Data+G	Data	Grad
Bucky 1024^3	4096	1334	3.071	5.51	2.6
PPM 512^3	512	284.9	1.797	2.43	1.65
Engine 256^3	64	24.6	2.6	4.35	2.22

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 and Grad columns shows the compression ratios for the data and gradients.

cracks will appear in the isosurface for frame f_i if there is not enough time to process all of the newly visible diamonds.

Our occlusion method allows visible regions to remain visible for several frames. This 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. Our occlusion culling method is conservative because diamonds marked as visible can become occluded for a few frames before an occlusion test is performed and because we render bounding boxes to determine a diamonds's visibility. Thus some occluded triangles will be extracted and rendered.

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.

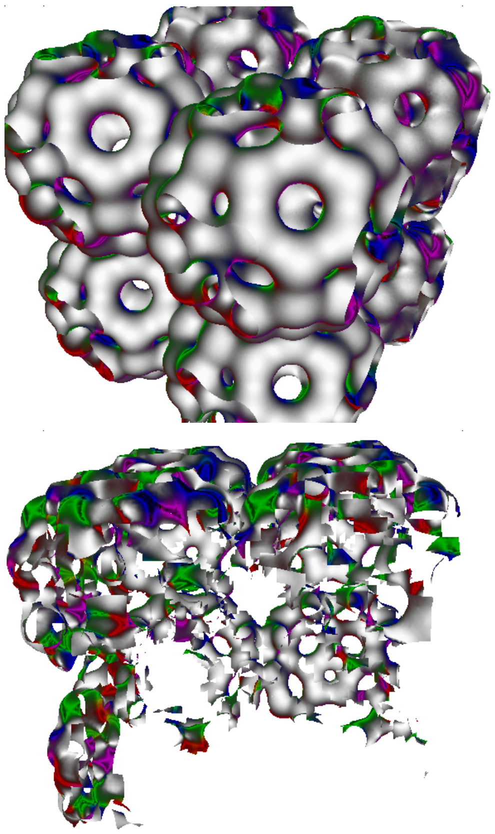


Figure 5: High resolution rendering of the 1024^3 synthetic buckball dataset, Isovalue = 80, Error = 0.5. Top: The isosurface extracted with occlusion culling contains 1.1M triangles, 100K diamonds and renders at 4.2 fps. In each frame, 15-18K occlusion tests are performed and 4-6K diamonds are found to be occluded. Bottom: A backside view shows the regions removed by occlusion culling. Without occlusion culling, the surface contains 2.99M triangles, 156K diamonds and renders at 3.3 fps.

5 Results

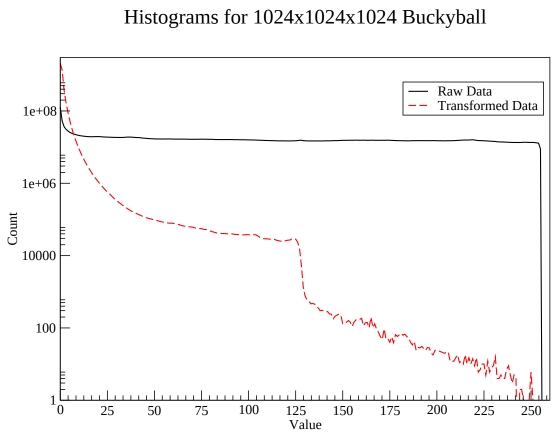
Our test machine is a 2 Ghz Pentium with 1 GB of main memory and a GeForce4 Ti 4600 graphics card. The amount of main memory available for uncompressed data was limited to the lesser of 512 MB or $(1/2)$ of the uncompressed dataset size. We have tested our algorithms on several large volume datasets. 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 512^3 chunk from timestep 273 of the Richtmyer-Meshkov simulation dataset from Lawrence Livermore National Laboratory [29]. The engine dataset is originally $256 \times 256 \times 128$ and has been resampled to 256^3 with the cell aspect ratios appropriately adjusted at runtime. 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 8 shows histograms of the raw and transformed data. 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. In general, the data compresses much better than the gradients. For several test datasets, we have lossless compression ratios for data and gradients between 1.8:1 to 3.1:1. This is comparable to the lossless compression rates given in [14].

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 times to decompress a single page. This takes n input bytes, where n is the size of the compressed page, and produces 4×2^k deltas; the time to load data from disk is not considered. Over several test runs the decompression speed ranged from $10-110$ 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* is measured as decompression time plus the time to compute the actual data and gradients using the deltas.



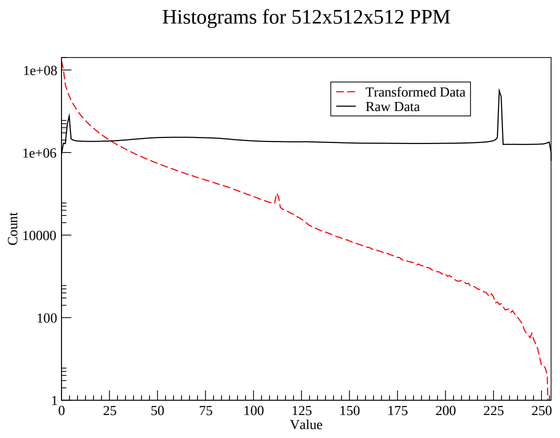


Figure 8: Histograms of the magnitudes of raw and transformed values (data and gradients) for the Buckball and PPM datasets (8-bit). As the dashed graphs indicate, the magnitudes of the transformed values are clustered much more closely toward 0. This indicates that the entropy of the data has been reduced and that it should compress better.

This includes the time to invert the z-order indices, compute the vertices needed for reconstruction and fetch their associated data and gradients. It also includes the time to load data from disk since this significantly affects the performance of out-of-core algorithms. The reconstruction speed is affected most by the time to load data from disk and the cost of fetching the needed data for the split edge vertices. Reconstructing a single point requires two memory lookups or a total of 2^{k+1} accesses to reconstruct one page. For the buckyball dataset $k = 12$, it contains 262144 pages and each uncompressed disk page contains $4 * 4096$ or 16384 bytes. The isosurface shown in Figure 6 is extracted from a 512^3 PPM volume compressed with $k = 12$. The dataset is divided into 32768 pages each containing 2^{12} data elements requiring 2^{13} lookups to reconstruct one page. Similarly, the isosurface in Figure 7 is extracted from the same volume compressed with $k = 11$. This volume is divided into 65536 pages each containing 2^{11} elements and requiring 2^{12} lookups to reconstruct one page.

The isosurface from the PPM dataset shown on the left of Figure 6, contains 4.4M triangles without occlusion culling and 1.38M triangles with occlusion culling. The reconstruction speed ranged from $1.38-12.66$ MB/sec ($362K-3.32M$ data points per second) with an average speed of 5.8 MB/sec (1.52M data points per second). Figure 7 shows another isosurface from this dataset. The isosurface contains 1.22M triangles with occlusion culling and 3.98M triangles without occlusion culling. Extracting the isosurface from the compressed volume required 22 seconds and 80 iterations of the split/merge refinement algorithm or 0.275 seconds/frame; extraction from the uncompressed volume required 20 seconds and 87 iterations or 0.23 seconds/frame. Compared to isosurface extraction from the uncompressed volume, the isosurface extraction from the compressed volume is nearly as fast and uses only 56% of the storage space. When extracting the isosurface from the compressed volume, the reconstruction speed ranged from $0.25-12.35$ MB/sec ($66K-3.24M$ data points per second) with an average speed of 6.7 MB/sec ($1.75M$ data points per second). Over the course of the isosurface extraction, a total of 19.85 MB (6.96% of the total compressed data) in 3153 pages was loaded from disk, and a total of 7.4 seconds was spent reconstructing data the compressed pages. Extracting the buckyball isosurface in Figure 5 requires 23.4 and 20.1 seconds from the compressed and uncompressed volumes respectively The reconstruction speed was between $0.66-11.9$ MB/sec with the average of 10.1 MB/sec. A total of 9.4 MB, or 0.7% of the total compressed data, in 2369 pages was loaded and reconstructed. Compared to isosurface extraction from the uncompressed data, the isosurface extraction from the compressed data is nearly as fast and uses a much smaller amount of the storage space showing that interactive frames can be achieved using highly compressed volumes.

The interactivity of the visualization, with respect to frames-per-second, is not significantly affected because the cost of data reconstruction and occlusion culling is amortized over the whole extraction process. Furthermore, since our algorithm progressively refines the isosurface over time, the user is provided with immediate visual feedback in the form of a coarse isosurface that is continuously being refined and can be interacted with while the full isosurface is extracted.

Occlusion culling results for the images shown in Figures 1, 5, 6 and 7 are summarized in Table 2. For surfaces extracted with and without occlusion culling, the table shows the number of triangles in the isosurface, the number of diamonds in the tetrahedral mesh. The number of occlusion queries performed in each frame 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. Occluded diamonds are not contoured. The extracted isosurface geometry is stored as indexed face lists in vertex arrays. This allows both the surface geometry (vertices and normals) and triangle indices to be cached on the graphics card. Since a large number of triangles are kept between frames, this allows for fast, asynchronous rendering.

The isosurfaces extracted from the engine and buckball datasets 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 and diamonds. This reduces the rendering time and memory usage. 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. Currently, the benefits of occlusion culling are limited by the time to perform the queries and the readback speed from the graphics card which is currently much slower than the download speed. However, in future graphics systems this should improve. Compared to the engine and buckball datasets,

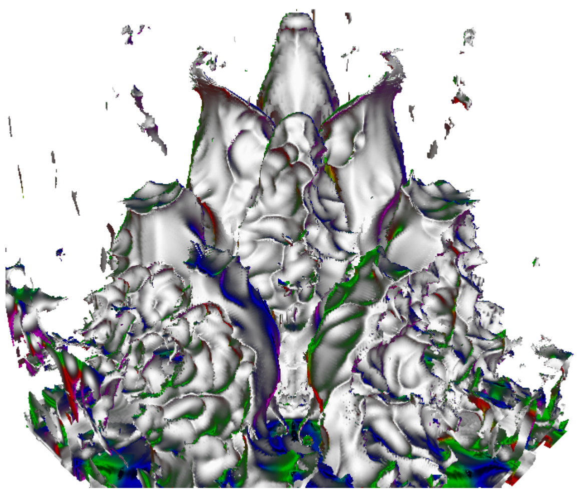
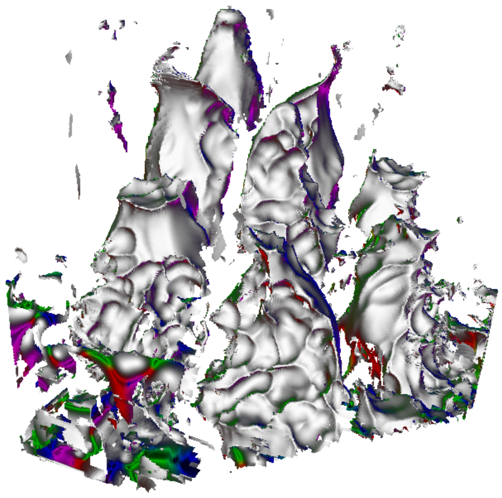



Figure 6: Top : Isosurface from the PPM dataset: Isovalue = 228, Error = 0.50, 4.4M triangles, 215K diamonds, rendered at 2.5 fps. Bottom: A view from the side shows the sections of the surface removed by occlusion culling. The isosurface with occlusion culling contains 1.4M triangles, 149K diamonds and renders at 4.6 frames per second. 15-19K occlusion tests are performed each frame, and 6-7K are found to be occluded. Results with and without occlusion culling are given in Table 2.

isosurfaces from the PPM dataset are much more complex; they have high depth complexity, and they contain a large number components and small features. These isosurfaces generally contain a larger number of triangles when compared to isosurfaces from similarly sized volumes at the same error bound. For this dataset, occlusion culling reduces the number of extracted triangles by about 70% and the size of the LEB mesh by 30% leading to significant frame rate improvements even when high resolution surfaces containing millions of triangles are being extracted.

Movie Summary: The movie included with the paper submission was created using Adobe Premier and probably is best viewed using Windows Media Player.

The movie shows an isosurface from the engine dataset extracted from a compressed volume. The first clip shows the isosurface extracted without occlusion culling. It starts with the isosurface already extracted at a high resolution and thus very little decompression occurs as the volume is rotated. The information at the bottom of the movie frames shows the frame rate, number of triangles in

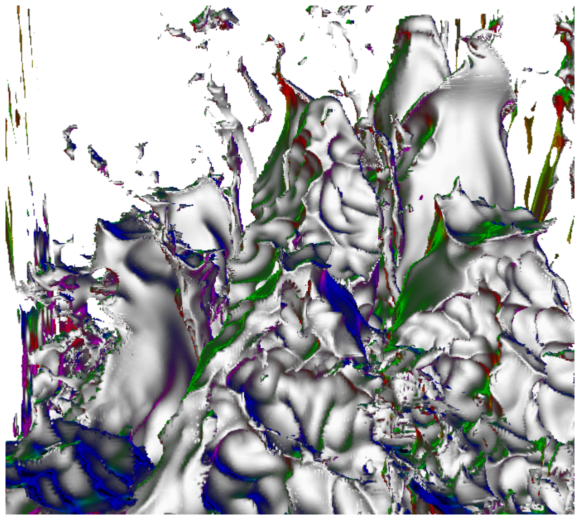


Figure 7: Isosurface extracted from the PPM dataset with occlusion culling: Isovalue = 225, Error = 0.55, 1.22M triangles, 123K diamonds, 15-19K occlusion test per frame, rendered at 5.0 fps. Extracting the isosurface from the compressed volume required 22 seconds and 80 split/merge iterations and 20 seconds and 87 iterations from the uncompressed volume. Occlusion culling results are given in Table 2.

the isosurface and the number of diamonds in the tetrahedral mesh.

The second clip shows the isosurface rotate along the same path as before extracted with occlusion culling. The volume is decompressed as the isosurface is rotated since the isosurface is being extracted from previously occluded regions. The new line of printed information shows the number of occlusion tests being performed in each frame, the number of diamonds determined to be occluded and the time in seconds to perform the occlusion queries. Cracks that may appear in the isosurface rendered with occlusion culling occur because a limited amount of processing time is allocated per frame and there may not be enough time to extract the isosurface from all of the newly visible regions.

The last clip shows an isosurface rendered with occlusion culling that is rotated to show the regions of the surface that have been removed.

6 Conclusions and Future Work

We have presented new data compression and occlusion culling algorithms for fast isosurface extraction from massive datasets. Our fast and efficient compression algorithm reduces the storage requirements of these massive volume datasets and our occlusion culling algorithm reduces the number of triangles in the extracted isosurfaces. The synergistic combination of these two algorithms allows large isosurfaces consisting of millions of triangles to be visualized at interactive rates from massive, compressed 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 1.8:1 to 3.1:1 compression with 2.4:1 to 5:1 compression for the data and 1.65:1 to 2.6:1 compression for 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. Frame-to-frame coherence is

Dataset	Error	Triangles	Diamonds	Occlusion Tests Per Frame	FPS
Engine (256^3)	0.42	1.9M	98K	–	6.1
	0.42	553K(29%)	50K(51%)	7-8K	10.5
PPM (Figure 6)	0.50	4.4M	215K	–	2.5
	0.50	1.38M(31%)	148K (69%)	15-19K	4.6
PPM (Figure 7)	0.55	3.98M	170K	–	3.0
	0.55	1.22M(31%)	122K(72%)	13-18K	5.0
Buckyball (1024^3)	0.50	2.99M	156K	–	3.3
	0.50	1.1M(37%)	100K (64%)	15-18K	4.2

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.

utilized to create a good initial set of occluders from which an approximate depth buffer is created. Hardware occlusion queries are performed against this set of occluders and the mesh is quickly refined and coarsened in the visible and occluded regions.

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.

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

Acknowledgements

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. Joshua Senecal's work was supported in part by a United States Department of Education Government Assistance in Areas of National Need (DOE–GAANN) grant #P200A980307.

The engine dataset was obtained from http://www.volvis.org. The buckyball dataset was obtained from Oliver Kreylos at UC Davis.

References

[1] Udeepta D. Bordoloi and Han-Wei Shen. Space efficient fast isosurface extraction for large datasets. In *Proceedings of IEEE Visualization*, pages 201–208, 2003.

[2] Mark A. Duchaineau, Serban Porumbescu, Martin Bertram, Bernd Hamann, and Kenneth I. Joy. Dataflow And Re-Mapping For Wavelet Compression And View-Dependent Optimization Of Billion-Triangle Isosurfaces. In G. Farin, H. Hagen, and B. Hamann, editors, *Hierarchical Approximation and Geometrical Methods for Scientific Visualization*. Springer-Verlag, Berlin, Germany, (to appear)., 2002.

[3] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing Terrain: Real-time Optimally Adapting Meshes. In *Proceedings of IEEE Visualization 1997*, pages 81–88, 1997.

[4] J. El-Sana, N. Sokolovsky, and C. Silva. Integrating occlusion culling with view-dependent rendering. *Proc. of IEEE Visualization*, 2001.

[5] Jinzhu Gao, Jian Huang, Han-Wei Shen, and James Arthur Kohl. Visibility culling using plenoptic opacity functions for large volume visualization. In *Proceedings of IEEE Visualization 2003*, pages 341–348, 2003.

[6] T. Gerstner and M. Rumpf. Multiresolution Parallel Isosurface Extraction Based On Tetrahedral Bisection. In M. Chen, A. Kaufman, and R. Yagel, editors, *Volume Graphics*, pages 267–278. Springer-Verlag, 2000.

[7] Thomas Gerstner. Fast Multiresolution Extraction Of Multiple Transparent Isosurfaces. In Ronald Peikert David S. Ebert, Jean M. Favre, editor, *In Data Visualization 2001 Proceedings of VisSim 2001*, Annual Conference Series. Springer-Verlag, 2001.

[8] Thomas Gerstner and Renato Pajarola. Topology Preserving And Controlled Topology Simplifying Multiresolution Isosurface Extraction. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings of IEEE Visualization 2000*, pages 259–266. IEEE Computer Society Press, 2000.

[9] Naga K. Govindaraju, Brandon Lloyd, Sung-Eui Yoon, Avneesh Sud, and Dinesh Manocha. Interactive shadow generation in complex environments. In *Proceedings of SIGGRAPH 2003*, pages 501–510, 2003.

[10] Benjamin Gregorski, Mark A. Duchaineau, Peter Lindstrom, Valerio Pascucci, and Kenneth I. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings of the IEEE Visualization 2002*, 2002.

[11] Benjamin Gregorski, Mark A. Duchaineau, Peter Lindstrom, Valerio Pascucci, and Kenneth I. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings of the IEEE Visualization 2002*, 2002.

[12] Benjamin Gregorski, Joshua Senecal, Mark Duchaineau, and Kenneth I. Joy. Adaptive extraction of time-varying isosurfaces. In *To be published in IEEE Transactions on Visualization and Computer Graphics, available as LLNL UCRL UCRL-JP-200087*, 2004.

[13] Stefan Guthe and Wolfgang Staser. Real-time decompression and visualization of animated volume data. In *Proceedings of IEEE Visualization 2001*, pages 349–358, 2001.

[14] Stefan Guthe, Michael Wand, Julius Gonser, and Wolfgang Staser. Interactive rendering of large volume data sets. In *Proceedings of IEEE Visualization 2002*, pages 53–60, 2002.

[15] Haeyoung Ha. 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.

[16] K. Hillesland, B. Salomon, A. Lastra, and D. Manocha. Fast and simple occlusion culling using hardware-based depth queries. Technical Report UNC-CH-TR02-039, Computer Science Department, Univer-

sity of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 2002.

[17] Schneider J. and Rudiger Westermann. Compression domain volume rendering. In *Proceedings of IEEE Visualization 2003*, pages 293–300, 2003.

[18] Armin Kanitsar, Thomas Theußl, Lukas Mroz, Milos Srámek, Anna Vilanova Bartrolí, Balázs Csébfalvi, Jirí Hladùvka, Dominik Fleischmann, Michael Knapp, Rainer Wegenkittl, Petr Felkel, Stefan Guthe, Werner Purgathofer, and Meister Eduard Gröller. Christmas tree case study: Computed tomography as a tool for mastering complex real world objects with applications in computer graphics. In Robert Moorhead, Markus Gross, and Kenneth I. Joy, editors, *Proceedings of the 13th IEEE Visualization 2002 Conference (VIS-02)*, pages 489–492, Piscataway, NJ, October 27– November 1 2002. IEEE Computer Society.

[19] Haeyoung Lee, Mathieu Desbrun, and Peter Schroder. Progressive encoding of complex isosurfaces. In *Proceedings of SIGGRAPH 2003*, pages 471–476, 2003.

[20] Wei Li, Klaus Mueller, and Ari Kaufman. Empty space skippping and occlusion clippinf for texture-based volume rendering. In *Proceedings of IEEE Visualization 2003*, pages 317–326, 2003.

[21] Peter Lindstrom and Valerio Pascucci. Visualization of large terrains made easy. In T. Ertl, K. Joy, and A. Varshney, editors, *Proceedings of IEEE Visualization 2001*, pages 363–370. IEEE Computer Society Press, 2001.

[22] Lars Linsen, Jevan T. Gray, Valerio Pascucci, Mark A. Duchaineau, Bernd Hamann, and Kenneth I. Joy. 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.

[23] Lars Linsen, Bernd Hamann, , and Kenneth I. Joy. Wavelets for adaptively refined '3rd-root-of-2'-subdivision meshes. In *Proceedings of Sixth IASTED International Conference on Computers, Graphics, And Imaging (CGIM 2003)*, 2003.

[24] Y. Livnat and C. Hansen. View Dependent Isosurface Extraction. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of IEEE Visualization 1998*, pages 175–180. IEEE Computer Society Press, 1998.

[25] Adriano Loes and Ken Brodlie. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. In *IEEE Transactions on Visualization and Computer Graphics*, volume 9-1, January-March 2003.

[26] Eric Lum, Kwan-Liu Ma, and John Clyne. Texture hardware assisted rendering of time-varying volume data. In *Proceedings of IEEE Visualization 2001*, pages 263–270, 2001.

[27] Eric B. Lum and Kwan-Liu Ma. Rendering isosurfaces directly from 3d textures. Technical Report CSE-2003-10, Computer Science Department, University of California Davis, 2003.

[28] Eric B. Lum, Brett Wilson, and Kwan-Liu Ma. High quality lighting and efficient pre-integration for volume rendering. In *Proceedings of IEEE TVCG Symposium on Visualization*, 2004.

[29] Arthur A. Mirin, Ron H. Cohen, Bruce C. Curtis, William P. Dannevik, Andris M. Dimits, Mark A. Duchaineau, D. E. Eliason, Daniel R. Schikore, S. E. Anderson, D. H. Porter, , and Paul R. Woodward. Very High Resolution Simulation Of Compressible Turbulence On The IBM-SP System. In *Proceedings of SuperComputing 1999*. (Also available as Lawrence Livermore National Laboratory technical report UCRL-MI-134237), 1999.

[30] Gregory M. Nielson. Mc*: Star functions for marching cubes. In *Proceedings of IEEE Visualization*, pages 59–66, 2003.

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

[32] Tom Roxborough and Gregory M. Nielson. Tetrahedron Based, Least Squares, Progressive Volume Models With Application To Freehand Ultrasound Data. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings Visualization 2000*, pages 93–100. IEEE Computer Society Press, 2000.

[33] Amir Said and William A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. In *IEEE Transactions on Circuits and Systems for Video Technology*, volume 6, pages 243–250, June 1996.

[34] J. M. Shapiro. Embedded image coding using zerotrees of wavelets coeefficients. In *IEEE Transactions on Signal Processing*, volume 41, pages 3445–3462, Dec 1993.

[35] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. In *The Visual Computer*, pages 100–111, 1999.

[36] Rudiger Westermann. Compression domain rendering of time-resolved volume data. In *Proceedings of IEEE Visualization 1995*, pages 168–175, 1995.

[37] Sung-Eui Yoon, Brian Salomon, and Dinesh Manocha. Interactive view-dependent rendering with conservative occlusion culling in complex environments. In *Proceedings of IEEE Visualization 2003*, 2003.

[38] H. Zhang, D. Manocha, T. Hudson, and K. Hoff III. Visibility culling using hierarchical occlusion maps. *Proceedings of SIGGRAPH*, pages 77–88, August 1997.

[39] Xiaoyu Zhang, Chandrajit Bajaj, and Vijaya Ramachandran. Parallel And Out-Of-Core View-Dependent Isocontour Visualization. In David Ebert, Pere Brunet, and Isabel Navaz, editors, *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualizatation (VisSym-02)*, Vienna, Austria, May 27–29 2002. Springer-Verlag.

[40] Yong Zhou, Baoquan Chen, and Arie Kaufman. Multiresolution Tetrahedral Framework For Visualizing Regular Volume Data. In *Proceedings of IEEE Visualization 1997*, pages 135–142. IEEE Computer Society Press, 1997.