Title
Multiresolution View-Dependent Splat Based Volume Rendering of Large Irregular Data

Authors
Meredith, Jeremy S
Ma, Kwan-Liu

Publication Date
2001

Peer reviewed

Multiresolution View-Dependent Splat Based Volume Rendering of Large Irregular Data

Jeremy Meredith*
Lawrence Livermore National Laboratory

Kwan-Liu Ma†
University of California, Davis

Abstract

We present techniques for multiresolution approximation and hardware-assisted splat based rendering to achieve interactive volume visualization of large irregular data sets. We examine two methods of generating multiple resolutions of irregular volumetric grids and a data structure supporting the splatting approach for volume rendering. These techniques are implemented in combination with a view-dependent error based resolution selection to maintain accuracy at both low and high zoom levels. In addition, the error tolerance may be adjusted at run time to obtain the desired balance between high frame rates and accurate rendering. Along with an effective way to compute gradients for lighting, we offer an integrated solution for interactive volume rendering of irregular-mesh or meshless data, and we demonstrate our technique on unstructured-grid data sets from aerodynamic flow simulations.
Keywords: Hardware-assisted rendering, irregular-grid data, lighting, multiresolution representation, splatting, volume rendering.

1 Introduction

Scientists today make use of parallel computers consisting of hundreds to thousands of processors to conduct large scale simulations. They increasingly use irregular computational meshes to better allocate computing resources for greater accuracy. Visualization of large scale data from these simulations presents a number of challenges, especially volume visualization which requires rendering the contents of every cell in the data set. There are many solutions to tackling this problem on regular grids, where connectivity is simple, cell size is constant, partitioning for parallel computation is straightforward, and the data lends itself well to a hierarchical representation. However, none of these assumptions carry over to irregular data sets, and novel approaches must be constructed to deal with the complexity.

We consider irregular data to be those on either non-rectilinear grids or a collection of scattered data points. Several novel algorithm designs have been developed for software rendering of irregular data [3, 5, 6, 7, 12, 23], among which [23] can handle multiple intersecting grids commonly found in CFD data sets, and [3] demonstrated high rendering efficiency. To make possible interactive rendering of very large scale data, Ma and Crockett developed a highly scalable distributed-memory parallel algorithm for unstructured-grid data [12].

To speed up rendering with graphics hardware, Shirley and Tuchman introduced the Projected Tetrahedra (PT) algorithm which converts tetrahedral cells into sets of overlapping triangles that can be efficiently rendered by polygon graphics hardware [18]. More works followed to improve the accuracy of the PT algorithm [19, 24]. Recently, Rottger, Kraus and Ertl have extended the PT algorithm by employing 2-d and 3-d hardware texture mapping [17]. Other hardware-assisted algorithms include the incremental slicing approach by Yagel, et al. [26], the multiresolution slicing approach by Kreylos, Ma and Hamann [8], and the two-pass approach by Westermann and Ertl which reduces the cost of depth sorting polyhedra [21].

A few other unique approaches worth mentioning are the stochastic resampling technique presented by Mao for using splatting [15], the integrated tetrahedral mesh compression and rendering technique demonstrated by Yang, Mitra and Chiueh [27], and the out-of-core strategy proposed by Farias and Silva for rendering data of arbitrary sizes [4].

The aforementioned techniques have addressed many different aspects of the irregular data visualization problem. In this paper, we describe the design and experimental results of a multiresolution, hardware-assisted approach. We present different techniques for implementing multiresolution approximation of irregular data, coupled with a hardware-assisted splatting approach, to achieve interactive visualization and exploration of large scale data. The connectivity of the original data set is discarded and the final representation for each level of the multiresolution data set is a point cloud. The data set is then stored within an octree data structure, with each leaf node in the tree containing approximately the same number of data points.

The rendering phase involves traversing the octree structure in view dependent order. At each node in the traversal, the approximate error is calculated for each of the resolutions contained within that subtree. This information is then used to determine whether to stop and render the selected resolution, or to descend to the children and find the appropriate resolution for each child of the node.

While hardware-assisted rendering helps us achieve the desired interactivity, the resulting image quality, as shown in subsequent sections, seems close to those of the previously published results using software rendering. In fact, because of the ability to explore the data at different resolutions and at high interactivity, we are often able to derive strikingly powerful transfer functions to reveal important features in the data set. Our approach to multiresolution rendering may be applied to almost any large-scale irregular-grid or meshless data due to its simplicity and flexibility. In addition, we show how gradient values may be reasonably approximated for the

*Lawrence Livermore National Laboratory, 7000 East Ave, Livermore, CA 94550, meredith6@llnl.gov

†CIPIC & Department of Computer Science, University of California, One Shields Avenue, Davis 95616, ma@cs.ucdavis.edu

resulting point data, and how more informative visualizations may be produced with gradient-based shading.

The rest of the paper is organized as follows. Section 2 contains the discussion of the multiresolution approximation methods. Section 3 addresses the data structure and techniques used for rendering the data. Section 4 contains results of these techniques regarding image quality and performance, and Section 5 concludes our study and suggests directions for future research.

2 Multiresolution Representations

In its most basic sense, a multiresolution data set is a sequence of data sets, including the original full resolution data and a series of successively lower resolution approximations of the original. A mipmap of textures is a common example of this: it is a sequence of textures, each one half the size in each dimension of the previous texture in the hierarchy [25].

A common approach to generating a multiresolution representation of a data set is to analyze the error of the variable one is interested in. For areas where there is little or no change in the value of the variable, more information can be discarded. This keeps the most information at the areas where it matters most and thus minimizes the error. One example of this technique is wavelet based compression. The detriment to using this technique is directly due to the benefit: the lower resolution data sets were generated because of the values of a single variable. If one wants to change variables, one needs to regenerate the multiresolution approximations.

It may be that a multiresolution representation could be generated based upon the error across all values in the original data set, but in general the error of one variable may not correspond to the error of another variable. We have the benefit with unstructured data that the data points themselves are not evenly distributed. A higher concentration of points in an area of space implicitly indicates the area where more points should go in a lower resolution approximation. Instead of a data-based multiresolution scheme, therefore, we generated the approximations from geometry-based schemes.

Previous works on multiresolution representations have largely been focused on simplification of surface meshes or regular volume data. Study of irregular volume data problems has been rather sparse. Leutenegger and Ma [11] proposed a multiresolution framework for interactive visualization of large unstructured-grid data but the focus of the study was on the underlying external memory organization using an R-tree. Trotts, et al. presented a tetrahedral collapse algorithm base on a local error controlling criterion but it was designed for tetrahedralized rectilinear-grid data [20]. Cignoni, et al. used a Delaunay refinement strategy that is able to generate finer resolutions for non-convex complexes [2]. More recently, they also developed a systematic, accurate error measure mechanism for simplifying irregular volume data based on edge collapse [1], and this technique also ensures the geometric or topological correctness of the simplified data. Our proposed methods will work on all mesh types, or even scattered data with no connectivity at all.

2.1 Maximum Independent Set Method

In our study, the first method used was based upon the maximum independent set of the previous resolution's vertices. The maximum independent set (MIS) over a graph $G = (V, E)$ is the largest subset V' of the original vertices that are not connected by an edge in E. Generation of the MIS is an *NP*-complete problem, but heuristics exist to quickly generate a maximal independent set, where no vertex can be added to V' and still have the vertices disjoint by E. The method we used involved taking the lowest degree vertex in V, adding it to V', removing that vertex and its neighbors, and repeating for all of V. The connectivity for V' is then generated using a Delaunay tetrahedralization so that further resolutions may be generated. Since the edges between vertices tend to occur with the same distribution as the vertices themselves, this means that the coarser resolution will have a spatial distribution similar to the finer resolution.

2.2 Direct Octree Method

The second method we used was derived directly from the octree structure we used to store the points. To begin, assume that the data points are stored in the leaf nodes in an octree, and that each leaf node contains roughly the same number of points. To generate a coarser resolution data set, we take exactly one point from each leaf node based upon some distribution criteria. In this case, our criterion is to choose the data point closest to the center of the octree node, with the justification that it will allow for a slight smoothing of the distribution of points. Since the octree structure was created explicitly so that there will be more leaf nodes in areas of higher point concentration, the next resolution will be generated with a similar distribution. Note that this requires that each octree node be permitted to hold at least eight data points, otherwise progress on generating coarser resolutions will quickly halt.

3 Rendering

3.1 Splat Based Rendering

Splatting [22] can be a reasonably accurate approximation of the volume rendering integral. Its chief benefit, however, is its efficient use of desktop graphics hardware. With two-dimensional texturing not only commonplace but free in terms of rasterization time on modern hardware, rectilinear grids can be quickly rendered with a single polygon per voxel and a single Gaussian kernel filter for all renderings. Applying this technique to unstructured data is not straightforward, though, since the appropriate kernel for an unstructured cell is not easy to calculate.

We have chosen to work around this problem by using a simple data structure itself supportive of calculating the kernel. The data structure is essentially an octree with roughly the same number of data points stored at each leaf node, and no connectivity information is stored for the data points.

To create this structure, we first choose some number N designating the maximum number of data points that may be stored in any node. We start by creating a bounding box with equal length sides around the original data set and making this the root of the octree. Recursively, then, for any node in the octree containing more than N points, we subdivide that node into eight octants and move the points contained in the given node into its appropriate children. The selection of the value of N is discussed in Section 3.4.

Once this structure is in place, the determination of the kernel becomes more straightforward. For any given viewing parameters, we can calculate the projected size of any octree node on the screen. Since we know how many points will be rendered within that octant, we can divide the screen area among the data points to calculate the approximate kernel size. For example, let there be n points in an octant and let s be the projected one-dimensional size of that octant. We know that there are roughly $\sqrt[3]{n}$ points along each of the three dimensions of that octant. The average distance between splats is then $(s/\sqrt[3]{n})$, and we use this value for the size of the splat kernel.

It follows that the calculation for the average z-distance between splats in view space is identical. This number then not only determines the splat size, but it is used to calculate opacity; the alpha value of each splat is based on the integral over this distance of its corresponding density in the transfer function.

We have thus performed a single calculation for each octant to determine splat size and alpha value for every point in the octant.

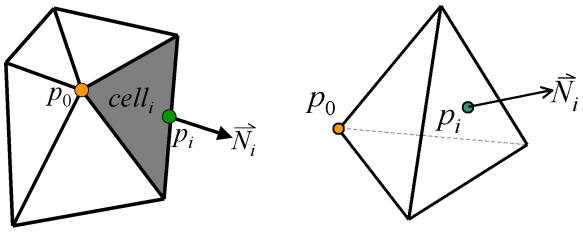

Figure 1: Setup for the gradient calculation.

This is a minimal amount of computation to determine these parameters for the data points, especially since $\sqrt[3]{n}$ can be stored in a lookup table because of the small range of n. It also prevents the large storage overhead associated with saving kernel information for every data point.

Finally, given the splat size and alpha value, we draw each data point in the octant with one square polygon and use the same Gaussian kernel as a texture map in the alpha channel for all points. A sharp dropoff in alpha at the edges of the splat kernel will result in less fuzziness in the final images, but it will not help smooth over the approximations we made to generate our lower resolution data. The splat shape is discussed further in Section 3.4.

3.2 Gradient Calculation

The gradient of a variable is commonly used to apply lighting calculations to the surfaces in volume renderings. This can enhance picture quality and give the viewer impressions of detail and shape which would not otherwise be apparent.

We used the original topology of the unstructured data set to calculate the gradient. If one is working with a data set which was originally a point cloud, the Delaunay tetrahedralization could be used to create the connectivity.

Ma, Van Rosendale, and Vermeer [14] suggested computing a divergence theorem surface integral at each vertex to approximate the gradient on unstructured data. The approach we take, which is simpler to implement and more accurate, is as follows. For a variable v over which we need the gradient, we perform a weighted average of one-sided difference estimates of the partial derivative of v.

Figure 1 shows a representation of the gradient calculation. The left diagram is a slice of the mesh through the point of interest (p_0), and the right diagram is p_0 with one of the surrounding cells. Each point p_0 with value v_0 for which we need to calculate the gradient is in general surrounded by some number of cells $cell_1$ through $cell_n$. For each of these cells, we calculate the centroid and average data value for the outer face of $cell_i$ with respect to p_0. Let the centroid of that face be p_i and the average value be v_i. If we let $\vec{N}_i$ be the normal in the direction of $(p_i - p_0)$, then the one sided difference estimate for $\partial v/\partial \vec{N}$ is $(v_i - v_0)/\|p_i - p_0\|$. We want to provide more weight to the estimate for those cells with greater volume, and we also want to provide more weight to those estimates for which $\vec{N}_i$ is in the direction of the gradient component we wish to calculate. Therefore, if we let the $(N_x)_i$ be the scalar X component of $\vec{N}_x$, then the formula to calculate the value of the gradient in the direction of X is:

$$grad_x = \frac{1}{W_x}\sum_{cells}\frac{v_i - v_0}{\|p_i - p_0\|}\cdot(N_x)_i\cdot|(N_x)_i|\cdot volume_i$$

$$where \quad W_x = \sum_{cells}|(N_x)_i|\cdot volume_i$$

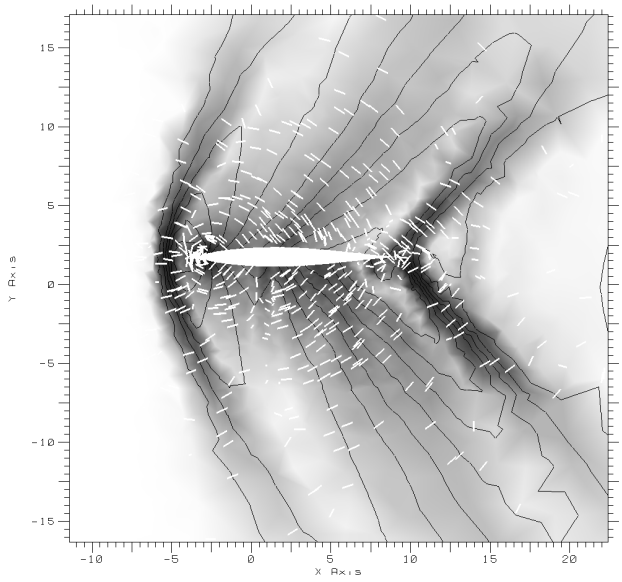

Figure 2: The contour lines are over the variable of interest, the shade shows the gradient magnitude (dark=high, light=low), and the vectors (line segments) show the gradient direction.

The calculations for the other components are derived similarly, and the gradient is then simply $<grad_x, grad_y, grad_z>$. Figure 2 shows an example of the results of this calculation on a 2-d slice of a data set from a simultion of air flow around an airplane wing. The black contour lines in this figure are over the variable of interest, the shade is determined by the magnitude of the gradient, and the white vector plot shows the direction of the gradient. As expected, the gradient magnitude is greatest where the contour lines are closest together and the variable is thus changing quickly. Also as expected, the gradient direction (as shown by the scattered white lines) is perpendicular to the black contour lines and thus to the change in the variable's magnitude.

Figure 3 shows examples with and without use of this gradient calculation for shading. The lighting calculation is a simple ambient and diffuse model with the light source located at the viewpoint. As shown, the lit one conveys the structure of the flow much better. See also figure 6 on the color plate.

3.3 View Dependent Optimization

To this point, we have discussed how to generate multiple resolutions of the same unstructured data set, how to store the data, how to determine the screen space each data point influences, and how to render the data; we have not discussed how to actually perform the multiresolution rendering. The simplest approach would be to allow the user to choose a resolution, to access the octree structure containing that resolution, and to render using that single octree structure. It is possible to do better. Laur and Hanrahan [10] used an octree hierarchy to provide a view-dependent rendering framework for regular grids. LaMar, Hamann, and Joy [9] created a texture hierarchy and performed view-dependent viewing of regular volume data.

We also take a view-dependent approach to optimize rendering and viewing. In the preprocessing step, we combine the octrees containing each resolution into a single octree. This implies that all the data is no longer simply at the leaf nodes but also in many interior nodes for resolutions coarser than the original. In addition,

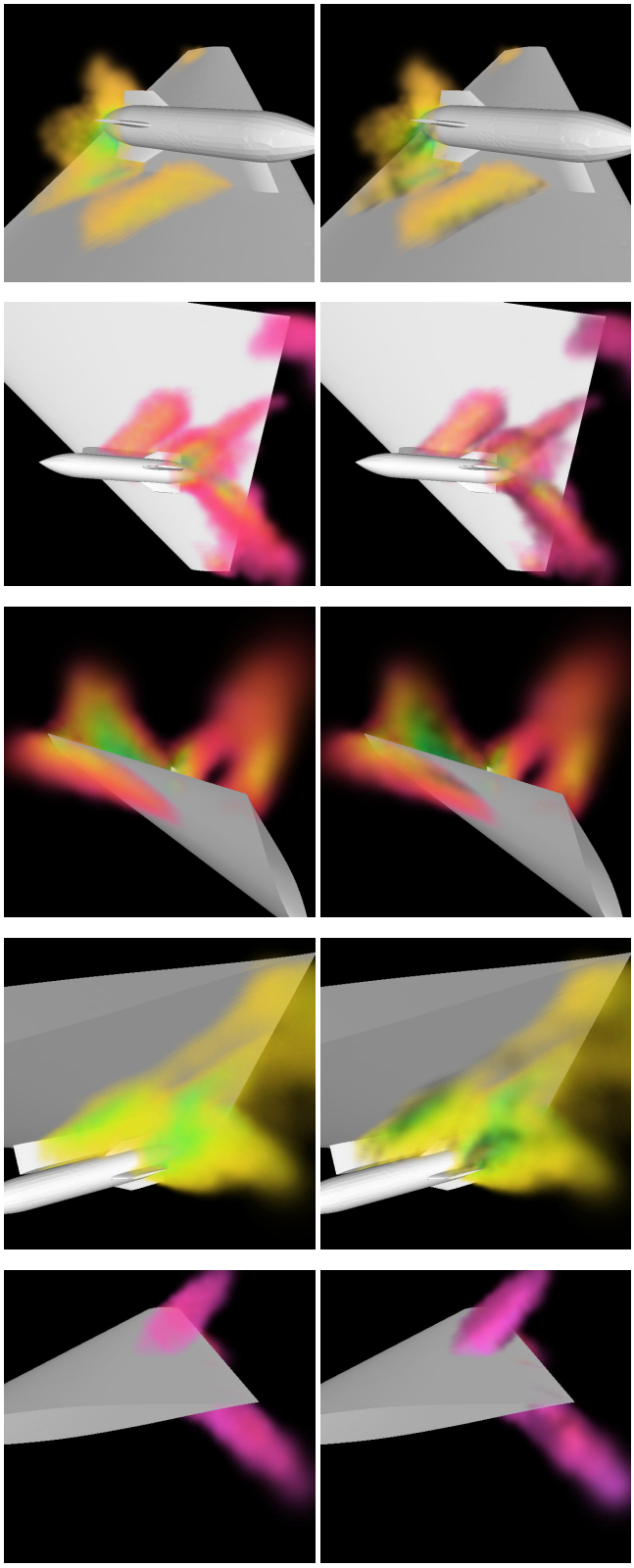

Figure 3: For each pair of images, the image on the left is unshaded, and the right one is lit using the calculated gradient. It is clear that the lit ones provide more information about the flow field.

at each node, we store for each resolution how many data points are contained within this node or its children.

In the rendering phase we use this combined data structure and that small bit of additional information. At each node, we start at the coarsest resolution available, and search toward the higher detail resolutions. For each resolution, since we know the size of this octree node and the number of data points within it or its children, we can determine as usual the approximate splat size for the data points. We choose the first resolution which produces a splat size below a tolerance set by the user. (This tolerance is specifically the maximum percentage of linear screen space which any one splat may cover. For example, if the tolerance is 10% for a 1000x1000 window, then no splat may exceed 100x100 pixels.) If the chosen resolution is stored in the current node, we stop the traversal and render. Otherwise, we descend to the children and recursively repeat the process. This technique allows the user to specify an error tolerance in units of screen space, a parameter which makes intuitive sense.

Note that during the traversal of the octree, we may cull those nodes which lie behind the viewer or are offscreen. Also note that we can arbitrarily render any of the resolutions at any point in the traversal of the tree. This is a direct consequence of discarding the connectivity information and instead calculating the kernel size from the information about the data structure.

3.4 Rendering Issues

Above, it was mentioned that the bounding box must have equal length sides; in other words, it must be a cube. The reason for this is simple. If the bounding box were not a cube, each node in the octree, and thus the kernel for each data point, would also have a non-1:1:1 aspect ratio. When rendering, it is far simpler to assume that the projected area of each node has an equal width and height and that it can be approximated with a circular Gaussian splat. If the nodes had varying shapes, either a large amount of calculation must be done to correctly project the kernels of the data points, or a large lookup table of splat shapes could be created beforehand.

When using a splat based approach, rendering must proceed back to front for the over operator to work as intended. The octree nodes are visited back to front, so the rendered data points are almost completely drawn in correct order just by this traversal. For an orthographic projection, the order of tree traversal can be calculated once, but for a perspective projection, it must be recalculated at each node. Furthermore, we have multiple data points to render within each node, so these data points should be sorted for each node prior to rendering. This could be done as a quicksort for every node and view parameters, or the data points could be pre-sorted along the three major axes to provide a fast approximation at the expense of data size. The speed decrease due to sorting clearly depends upon the number of points per node, but it is significant for almost any quantity. However, the difference between sorting and not sorting at all is nearly impossible to see. There are several reasons for this. Low opacities are commonly used in volume rendering to capture more information from the interior of the volume. There should also be a low number of data points within each octree node, specified by a choice of a smaller N. In addition, since we have discarded the original geometry, by necessity the remaining visual features must be composed of a slightly larger number of similarly valued data points. Since the images are nearly indistinguishable with or without sorting, this makes a good case for skipping the sorting (within octree nodes) for at least previewing purposes.

This brings up the question of choosing N, the maximum number of points within an octree node. A good choice is important because the selection occurs during preprocessing and cannot be changed later. Note that the smaller N is, the deeper the resulting hierarchy and thus the larger the memory footprint for the same original data

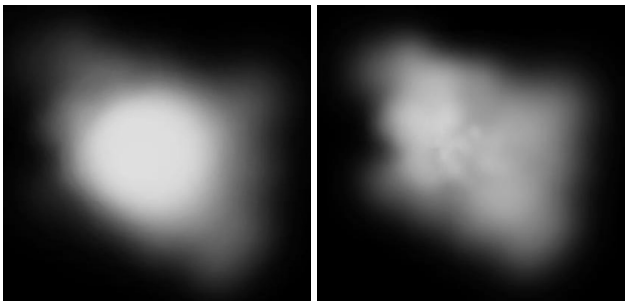

Figure 4: Choice of N for a constant transfer function. The left image was rendered with $N = 25$, the right with $N = 5$.

set. However, there are important rending issues to help guide the choice of N. Figure 4 shows an image with $N = 25$ and an image with $N = 5$. With a higher N, points are too likely to become unevenly distributed within octree nodes and they commonly receive too large an estimated kernel size. This has happened in the image with $N = 25$, where a high concentration of data points in the center of this data set is visible from a distance as a set of large splats. However, the smaller N is, the more likely the octree structure will have empty nodes where there should be none. Take an example of $N = 1$ and an octant with two data points: the octant must subdivide into eight children, six of which must by definition be empty. For this example, one can assert that most of the space should be covered by the kernels for these data points, not empty. The image with $N = 5$ looks much more evenly distributed than the image where $N = 25$, but it appears a bit "splotchy" in places - this is the sign of N being too small. For the remainder of the figures in this paper, we have chosen to use the smallest N allowable by the direct octree multiresolution method: $N = 8$. This number makes a good compromise between the extremes, and it appears to remain a good choice independent of data set size and data point geometric distribution.

4 Results

4.1 Comparison of Multiresolution Generation Techniques

There are several major differences between the MIS method and the direct octree method. First, the generation of the MIS was slow due to the complexity of Delaunay tetrahedralization code. For a 100,000 point data set (corresponding to 500,000 tetrahedra), generating the multiresolution approximation through the MIS approach took a few minutes on a single processor of an SGI Origin 2000. By contrast, the simplicity of coding an $O(n \cdot logn)$ algorithm for the direct octree approach allowed the generation of the multiresolution data to occur in only a few seconds.

Additionally, the data size reduction at each step is about $\frac{1}{6}$ for the MIS method and $\frac{1}{3}$ for the octree method. This means there are more levels of resolution for the octree method. This is both good and bad; it means the total data size is 120% for the MIS method and 150% for the octree method, but there are about twice as many resolutions to choose from for the direct octree method. This could reduce "popping" artifacts because resolutions will change less suddenly.

Figure 7 on the color plate shows a comparison of both techniques at similar resolutions, with gradient shading disabled. The first image shows the MIS technique at 15% of the original data set size, the second shows the octree method at 10% of the original data set size, and the third shows the image rendered at the

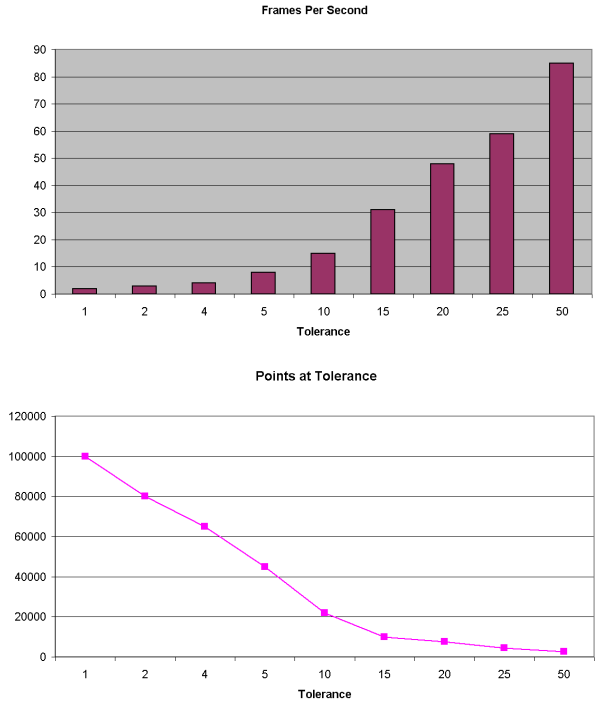

Figure 5: Frames per second (top) and number of visible points (bottom) for varying tolerances using the same view parameters for a 100,000 point dataset.

full resolution. There are a few noteworthy differences in Figure 7. A subjective comparison of the two techniques might indicate that the octree method looks closer to the original, even though the octree image contains only two-thirds the amount of data shown in the MIS image. A more objective comparison would indicate that despite the speed and simplicity of the octree multiresolution technique, it produces pictures of at least similar quality for the same data size as the MIS technique.

4.2 View-Dependent Results

Figure 8 on the color plate shows a series of images with the same view parameters and transfer function. In each successive image, we increase the tolerance, where the given tolerance is specified in terms of the minimum allowable percentage of screen area that a given splat may occupy after projection. This is the action a user would perform to achieve a faster preview at the expense of accuracy. This is also, however, similar to what would happen if the user held the tolerance constant and zoomed out, so that the same points would occupy a smaller region on the screen.

Note that the detail only begins to noticably degrade at the 10% level, but the rendering speed has jumped drastically by then, from 2 to 15 FPS. The image at 20% tolerance is a still a reasonable approximation if the user desired 30 FPS, but by definition, it also depicts what the image would look like at the 1% tolerance if it occupied $\frac{1}{400}$th the screen area.

Figure 9 on the color plate shows images from a much larger data set which contains over 18 million tetrahedral cells. The first two images show our technique at 5% and 1% tolerance. The third image was rendered using a software cell-projection volume renderer. Compared to the first two images in Figure 9, this image is sharper and reveals some fine features in the data. We should be able to improve splatting images by following the techniques suggested by

Mueller, Möller, and Crawfis [16].

In [13], this large data set was rendered in approximately five seconds using 128 Cray T3E processors. In contrast, the first two images generated with our technique were rendered in 0.2 seconds and 1 second, respectively, on a single processor computer. This shows the power of our technique, as it renders the images in similar times to a massively parallel method, preserves most of the information found in the higher fidelity image, and requires only a single processor on a common desktop workstation. By increasing the tolerance, smooth frame rates are achievable even on a data set of this size.

5 Conclusions

We have described a method of performing a multiresolution approximation of irregular data, and for organizing, processing, and rendering this data at interactive frame rates no matter what viewpoint the user chooses. Since the depth of the tree traversal varies with only the logarithm of the data size and the rendering speed is dependent on rendered size, not original data size, this technique should scale well to very large data sets with the only restriction being available memory.

This implies, however, that this technique could be combined with out-of-core or parallel rendering to render even larger data sets. Our multiresolution approach results in data organization which particularly facilitates out-of-core processing. For example, since each coarser resolution is a subset of the previous, the array of data points can simply be reorganized such that the coarsest resolution is first and the additional points needed to generate each successive resolution follow in order. We need only store indices into this array in the octree, and when viewing coarse resolutions most of the data can remain on disk. This enhancement could prove our technique truly scalable to massive data sets.

In addition, different structuring of the data could provide more accurate results. For example, a k-d tree could provide a more consistent subdivision of the point cloud than the octree, and a suitable view-dependent variation of the splat shape would then provide a more accurate rendering.

Acknowledgments

This work has been sponsored in part by the National Science Foundation under contracts ACI 9983641 (PECASE Award) and ACI 9982251 (LSSDSV), and Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159. The authors are grateful to Dr. Paresh Parikh and Dr. Dimitri Maveriplis for providing the test data sets.

References

[1] CIGNONI, P., COSTANZA, D., MONTANI, C., ROCCHINI, C., AND SCOPIGNO, R. Simplification of tetrahedral meshes with error evaluation. In *Proceedings of the IEEE Visualization 2000 Conference* (October 2000), pp. 85–92.

[2] CIGNONI, P., MONTANI, C., PUPPO, E., AND SCOPIGNO, R. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics 3*, 4 (October-December 1997), 352–369.

[3] FARIAS, R., MITCHELL, J., AND SILVA, C. ZSWEEP: An efficient and exact projection algorithm for unstructured volume rendering. In *Proceedings of 2000 Symposium on Volume Visualization* (October 2000), pp. 91–99.

[4] FARIAS, R., AND SILVA, C. Out-of-core rendering of large unstructured grids. *IEEE Computer Graphics & Applications (to appear)* (July-August 2001). A Theme Issue on Large Scale Data Visualization.

[5] GALLAGHER, R., AND NAGTEGAAL, J. An Efficient 3-D Visualization Technique for Finite Element Models and Other Coarse Volume . *Proceedings of SIGGRAPH 89 (Boston,July 31-August 4, 1989). Computer Graphics 23*, 3 (August 1989), 185–193.

[6] GARRITY, M. P. Raytracing Irregular Volume Data. *Proceedings of 1990 Workshop on Volume Visualization (San Diego, December 10-11, 1990). Special issue of Computer Graphics, ACM SIGGRAPH 24*, 5 (November 1990), 35–40.

[7] GIERTSEN, C. Volume Visualization of Sparse Irregular Meshes. *IEEE Computer Graphics & Applications 12*, 2 (March 1992), 40–48.

[8] KREYLOS, O., MA, K.-L., AND HAMANN, B. A multiresolution interactive previewer for volumetric data on arbitrary meshes. In *Proceedings of the Workshop on Computer Graphics and Virtual Reality, 2000 International Computer Symposium* (December 2000).

[9] LAMAR, E., HAMANN, B., AND JOY, K. Multiresolution technique for interactive texture-based volume rendering. In *Proceedings of IEEE Visualization '99 Conference* (October 1999), pp. 355–362.

[10] LAUR, D., AND HANRAHAN, P. Hierarchical splatting: A processive refinement algorithm for volume rendering. In *Proceedings of SIGGRAPH 91* (1991).

[11] LEUTENEGGER, S., AND MA, K.-L. Fast retrieval of disk-resident unstructured volume data for visualization. Tech. rep., May 1998.

[12] MA, K.-L., AND CROCKETT, T. W. A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data. In *Proceedings of the 1997 Parallel Rendering Symposium* (October 1997), ACM SIGGRAPH, pp. 95–104.

[13] MA, K.-L., AND CROCKETT, T. W. Parallel visualization of large-scale aerodynamics calculations: A case study on the cray t3e. In *Proceedings of the 1999 IEEE Parallel Visualization and Graphics Symposium* (October 1999), ACM SIGGRAPH, pp. 15–20.

[14] MA, K.-L., VERMEER, W., AND VAN ROSENDALE, J. 3d shock wave visualization on unstructured grids. In *Proceedings of the 1996 Volume Visualization Symposium* (October 1996), pp. 87–94.

[15] MAO, X. Splatting of non-rectilinear volumes through stochastic resampling. *IEEE Transactions on Visualization and Computer Graphics 2*, 2 (June 1996), 156–170.

[16] MUELLER, K., MÖLLER, T., AND CRAWFIS, R. Splatting without the blur. In *Proceedings of IEEE Visualization '99 Conference* (October 1999), pp. 363–370.

[17] ROTTGER, S., KRAUS, M., AND ERTL, T. Handware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings of the IEEE Visualization 2000 Conference* (October 2000), pp. 109–116.

[18] SHIRLEY, P., AND TUCHMAN, A. A Polygon Approximation to Direct Scalar Volume Rendering. *Proceedings of 1990 Workshop on volume Visulization (San Diego, December 10-11, 1990). Computer Graphics 24*, 5 (November 1990), 63–70.

[19] STEIN, C., BECKER, B., AND MAX, N. Sorting and hardware assisted rendering for volume visualization. In *Proceedings of the 1994 Volume Visualization Symposium* (October 1994), ACM SIGGRAPH, pp. 83–89.

[20] TROTTS, I. J., HAMANN, B., JOY, K. I., AND WILEY, D. F. Simplification of tetrahedral meshes. In *Proceedings of the IEEE Visualization '98 Conference* (October 1998), pp. 287–295.

[21] WESTERMANN, R., AND ERTL, T. The VSBUFFER: Visibility ordering of unstructured volume primitives by polygon drawing. In *Proceedings of the Visualization '97 Conference* (October 1997), pp. 35–42.

[22] WESTOVER, L. Footprint evaluation for volume rendering. In *SIGGRAPH 90 Conference Proceedings* (August 1990), pp. 267–276.

[23] WILHELMS, J., VAN GELDER, A., TARANTINO, P., AND GIBBS, J. Hierarchical and parallelizable diret volume rendering for irregular and multiple grids. In *Proceedings of the Visualization '96 Conference* (October 1996), pp. 57–64.

[24] WILLIAM, P., MAX, N., AND STEIN, C. A high accuracy volume renderer for unstructured data. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (January-March 1998), 37–54.

[25] WILLIAMS, L. Pyramidal parametrics. In *SIGGRAPH 83 Conference Proceedings* (July 1983), pp. 1–11.

[26] YAGEL, R., REED, D. M., LAW, A., SHIH, P., AND SHAREEF, N. Hardware assisted volume rendering of unstructured grids by incremental slicing. In *Proceedings of the 1996 Volume Visualization Symposium* (October 1996), ACM SIGGRAPH, pp. 55–62.

[27] YANG, C.-K., MITRA, T., AND CHIUEH, T.-C. On-the-fly rendering of losslessly compressed irregular volume data. In *Proceedings of the IEEE Visualization 2000 Conference* (October 2000), pp. 101–108.

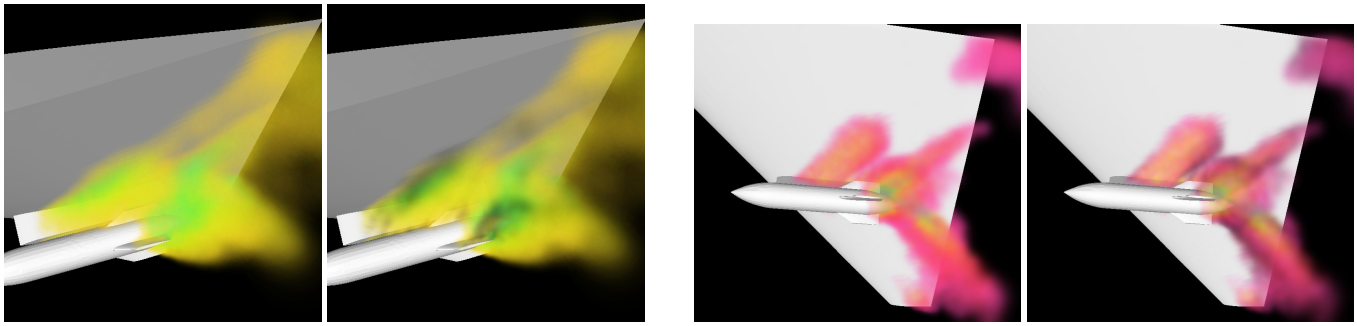

Figure 6: For both pairs of images, the image on the left is unshaded, and the right one is lit using the calculated gradient.

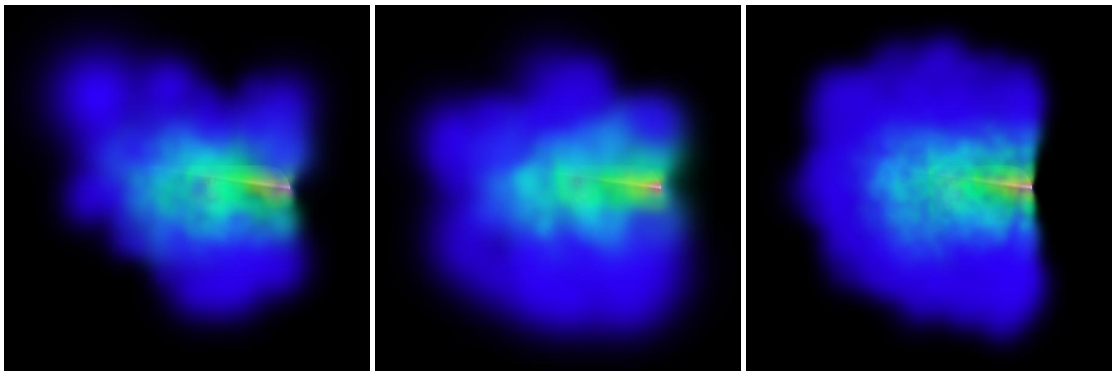

Figure 7: Left: MIS approximation at 15% of original data size. Center: Octree approximation at 10% of original. Right: Original, full resolution data set.

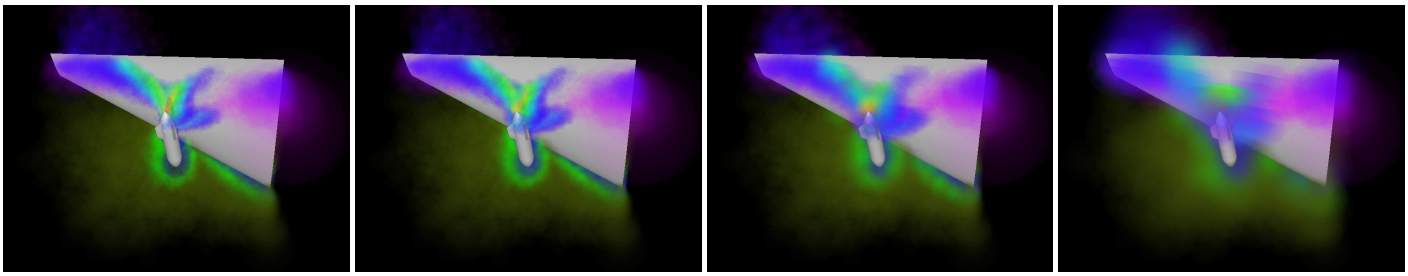

Figure 8: Images showing increasing tolerance at same zoom level. This replicates the effect of zooming out with a constant tolerance. From left to right, tolerances are 1%, 5%, 10%, 20%.

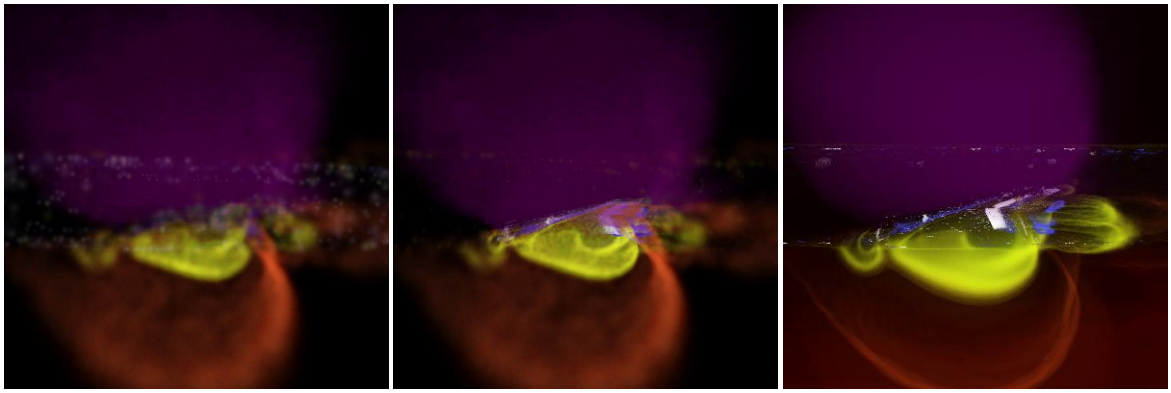

Figure 9: Images showing an 18 million cell data set. The left image was generated with tolerances of 5% (0.2 seconds), the center at 0.5% (2 seconds), and the right image was generated using a cell-projection volume renderer.