Multiresolution and Adaptive Rendering Techniques for Structured, Curvilinear Data

William C. Wynn[1]
Department of Computer Science
University of North Carolina at Chapel Hill

J. Fritz Barnes[2], Bernd Hamann[2,3]
Center for Image Processing and Integrated Computing (CIPIC)
Department of Computer Science
University of California, Davis

Mark Miller[4]
Lawrence Livermore National Laboratory
University of California

Abstract

In many applications one is concerned with techniques for visualizing data sets with real-time interaction. One technique for providing real-time performance is through the use of multiresolution techniques. These techniques provide multiple representations of a data set at different levels of detail. The idea is to select a level of detail that can be rendered within the user's time constraints. We discuss a mechanism which renders finer-detailed representations where the data set has a high frequency, and coarser representations where the data set has lower frequency. We present a new technique for storing curvilinear data sets within a quadtree representation and discuss two rendering schemes: an anti-aliasing scheme and a scheme for maintaining a specified frame rate.

1. Introduction

Novel approaches allowing interactive, real-time visualization are becoming increasingly important due to recent increases in data set sizes. Currently, there is a need to visualize data sets on the order of 10^6 zones and in the near future simulations will be performed for more than 10^9 zones. We are concerned with the development of methods which allow for the rapid analysis of such large data sets. The techniques we present are fundamental, and the hierarchical quadtree data representation we discuss could be used for many common visualization methods. We demonstrate our hierarchy's effectiveness for pseudo-color rendering (i.e., shading zones based on associated function values) and mesh-line rendering (i.e., outlining each zone's boundary segments.)

In the analysis of previous techniques and the synthesis of our technique for multiresolution storage and rendering the objectives considered were:

- **Minimizing storage requirements**. When we are dealing with large data sets on the order of 10^6 zones, doubling the storage requirements is a very serious matter; storage should be minimized.
- **Developing efficient access mechanisms**. It is desirable that utilizing and accessing the multiresolution representation will not degrade speed.
- **Robustness of technique**. Potentially, we could design a separate technique for multiresolution rendering for every visualization algorithm available. Since this requires support of multiple techniques this is not feasible. In our development we considered the need of various visualization techniques.

Considering our goals, it was important that the multiresolution representation minimize the storage requirements

[1]Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-3175 USA. E-mail: `wynn@cs.unc.edu`
[2]Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California, Davis, CA 95616-8562 USA. E-mail: `{barnes,hamann}@cs.ucdavis.edu`
[3]Co-Director of CIPIC and Visualization Thrust Leader
[4]Lawrence Livermore National Laboratory, University of California, Livermore, CA 94551 USA. E-mail: `miller86@llnl.gov`

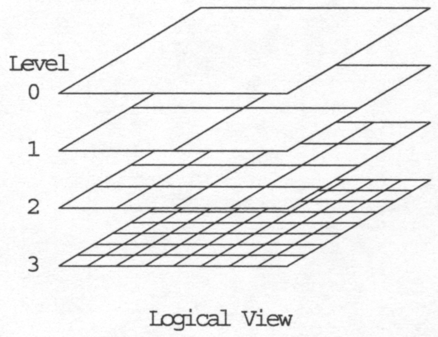

Figure 1. Multiresolution Hierarchy

while maintaining efficient access mechanisms. Our data structure is based on a *quadtree*, which is described in [10, 11, 4]. We use a pointer-less scheme for handling quadtrees. We show how to retrieve zone *siblings*, neighbors of a given zone, from the quadtree representation. Xia and Varshney [16] have developed adaptive surface triangulations to capture view-dependent properties, *e.g.*, highlights. Multiple quadrilaterals may be rendered in the same pixel region. This may result in aliasing effects when determining the actual color for a pixel. Our techniques are used similarly to Xia's and Varshney's to achieve quadrilaterals at roughly pixel sizes.

Our technique is based on the idea of constructing a *multiresolution pyramid*, a data representation hierarchy of triangulations with increasing precision, see [5]. We extend the idea of a hierarchy to quadrilaterals using a quadtree representation. One of the authors has previously looked at this with regard to terrain representation [9]; in this technique one stores a very coarse version of the data set (one quadrilateral or zone), see Figure 1. As the hierarchy is descended, each cell is subdivided into four sub-zones. When one descends the pyramid fine-detail representations of the data set are presented. At the base of the pyramid lies the original data.

We discuss two schemes for rendering images. The first scheme discusses the addition of a minimum-size constraint for the selection of zones from the hierarchy. When considering very large data sets the data being rendered may contain graphical primitives smaller than a pixel. One can reduce the work in the visualization algorithms if one does not apply the visualization algorithm multiple times to the multiple zones that are rendered to the same pixel. In our scheme we descend the hierarchy until we reach zones that are of approximate pixel size. Instead of continuing to subdivide we render this zone. This paradigm can also be explored in the realm of overlaying a plot with the corresponding mesh. If we have a large data set we will not be able to see the underlying plot. Our technique can specify a minimum rendering size.

The second scheme we utilize is based on a specified frame rate constraint. When constructing a given plot we will start with the coarsest representation; we consider the error between this coarsest representation and the actual data. We subdivide the quadrangle with the largest error. We continue the subdivision of the quadrangle with the largest error until we have selected as many zones as possible while satisfying the frame rate constraints. The zones that will be rendered are from a variety of levels of detail. We call the set of zones that will be rendered a *surrogate mesh*.

Our drawing paradigm attempts to optimize the quality of the image with respect to a rendering of the actual data set. We perform this optimization at run time. Alternatives, such as [2, 7], look at techniques for specifying a priori how to optimally subdivide the mesh. These techniques provide the same output regardless of the view. Our technique differs in that we attempt to provide an optimal resolution based on the current view. Lindstrom et. al. [8] also take this approach. Lindstrom's technique provides continuous levels of detail for regular grid representations of height fields. Our technique is applicable to more general rectilinear meshes and "nice" curvilinear meshes.

Hierarchical data representations have been developed for unstructured tetrahedral meshes in the context of visualization. An octree-based approach is described in [3]; this technique has been applied to marching cube algorithms for iso-surface generation. The idea is to minimize the number of triangles needed to construct the surface. Wilhelms and Van Gelder [15] also use octree-based methods for isosurface generation.

Wavelet-based multiresolution techniques are concerned with the construction and application of certain basis functions that allow the approximation of a given function at increasing levels of detail. These techniques are described by a variety of authors in [1, 6, 13, 14]. These techniques describe basis and wavelet functions for obtaining different resolution views of a dataset. This is different from our technique in that we store the different resolution views so that we can render different areas of the same data set at different resolutions.

2. The Multiresolution Hierarchy and Rendering Techniques

Our method attempts to derive a surrogate mesh from zones in the multiresolution structure. We do not limit ourselves to selecting zones at the same level in the hierarchy. This allows us to utilize more zones in areas where the data set is fluctuating, and utilize fewer zones in smooth sections of the data set.

We utilize the quadtree representation of the multiresolution hierarchy to store information that is too expensive to calculate while traversing the quadtree during rendering. Rendering consists of a two-pass traversal of the multiresolution hierarchy. During the first pass, we select the zones to be rendered by the algorithm. During the second pass, we "patch" holes and remove overlaps due to utilizing mesh representations at two different levels of representation.

We consider two techniques of zone selection. The first technique is used to perform anti-aliased rendering. This technique involves a recursive traversal of the quadtree. When the size of the quadrilateral representing the parent zone is below a certain minimum size we select this zone for rendering.

The second technique optimizes the quality of rendering while maintaining a pre-specified frame rate. This technique involves priority queue traversal of the quadtree. Hierarchical zones are given a priority with respect to the error between the hierarchical zone and the underlying data. The zones with the most error are subdivided, by descending the quadtree.

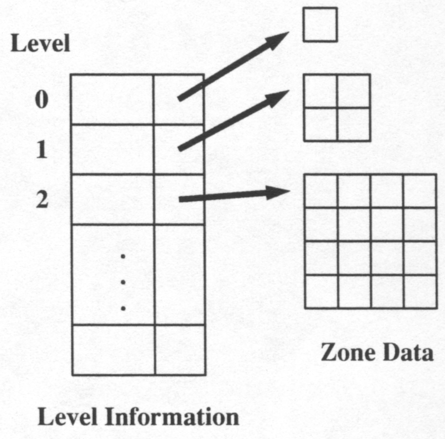

Figure 2. Quadtree Array-of-Arrays storage scheme. The quadtree representation occurs from the fact that at a level i, each zone is subdivided into four children at the $i+1$ level. Therefore each level has four times as much information.

2.1. Quadtree representation

Although many pointer-based schemes have been used to represent quadtrees, we use a different approach in order to save memory. Using this approach, an array of levels stores information common to each level in the hierarchy. Included in this structure is a pointer to the zone data for each level. The multiresolution hierarchy is stored conceptually as shown in Figure 2; the zone data is a two-dimensional array of zones for a given level. We can specify an address function: "Address(l, r, c)," which returns the address of the hierarchical zone at the given level l, row r and column c. Since we are dealing with an array of levels and a two-dimensional array of data at each level, we can determine the location of a hierarchical zone with three pointer dereferences. In our implementation, we store the two-dimensional array of hierarchical data at a given level in a one-dimensional array. This requires only two array dereferences while adding a shift, a multiply and an addition operation.

Utilizing the addressing scheme above, provides convenient access to neighboring information. We can calculate the address for relative zones (relative to a current zone at level l, row r and column c) as follows:

- **Parent zone.** The parent zone is the zone located exactly one level coarser in the hierarchy. The current zone is a higher-detailed fraction of the parent. The address for the parent zone is Address$(l-1, r/2, c/2)$.

- **Child zones.** The children zones are given by the zones located one level below the current zone. Children zones are the four children (subdividing both rows and columns). These addresses are given by Address$(l+1, 2r+\alpha, 2c+\beta)$, $\alpha, \beta \in \{0, 1\}$.

- **Sibling zones.** The siblings of a zone are those zones at the same level that share an edge or vertex with a given zone. Note that sibling zones do not necessarily share the same parent. The addresses of sibling zones is given by Address$(l, r+\alpha, c+\beta)$, $\alpha, \beta \in \{-1, 1\}$.

- **Final descendents.** One can determine the actual data that underlies a given hierarchical zone by looking at all zones in the *final* mesh which are found by recursively subdividing all children of the current zone. The final descendents will lie at level f, where f is the index of the final level, and have a row index between $2^{f-l}r$ and $2^{f-l}(r+1)-1$, and a column index between $2^{f-l}c$ and $2^{f-l}(c+1)-1$.

There are several advantages to this approach. It allows for compact storage of the data hierarchy. Since there is no need to store pointers to children zones, the amount of memory required to represent each zone is reduced by four pointers. This scheme is advantageous in the ease with which we can access sibling zones and use them in the calculations for the current zone. An additional advantage is that we can determine the zones in the underlying mesh in constant time; in contrast if we were using a pointer-based scheme accessing the actual data could require time proportional to the number of levels in the hierarchy. This flexibility allows us to reuse information at the bottom of the hierarchy rather that duplicating it at the hierarchical zone representation.

A disadvantage of this approach is that one must take special care for meshes which are logically non-square or logically non-power-of-two. Although these requirements do not significantly increase zone access time, the coding

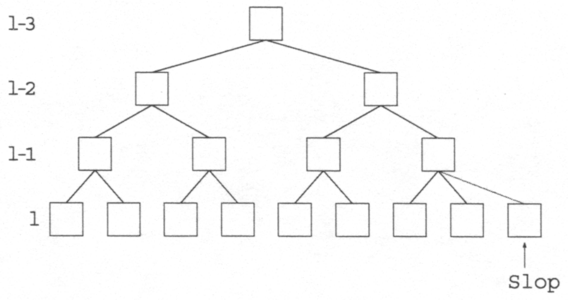

Figure 3. Handling non-power-of-two data in 1D.

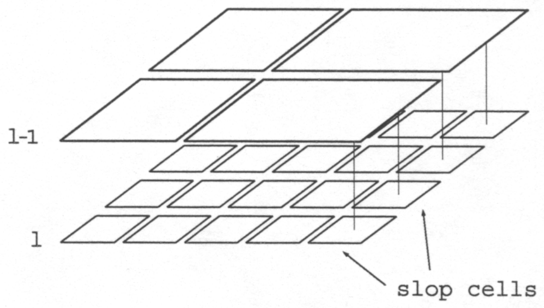

Figure 4. Handling non-power-of-two data in 2D.

complexity of the algorithms used for manipulating the hierarchy is slightly increased. Since zone memory is located in contiguous segments, pruning of the multiresolution tree cannot be performed where the underlying data is all the same.

2.2. Handling Non-square and non-power-of-two data sets

Square power-of-two data sets are easily represented by the quadtree representation. Unfortunately, most real-life data sets do not exist in this manner. We will first extend the quadtree representation discussed above for rectangular power-of-two data sets. We will then generalize this structure for non-power-of-two data sets.

For each level in the hierarchy we maintain two flags, which indicate whether zones in the level should be subdivided into row children, column children or both. For example a data set may contain more columns than rows. We have the flexibility to choose at which levels we will split rows since there are more column splits than row splits. Our current implementation would attempt to postpone splitting the rows as long as possible. This technique allows us to reduce the amount of space necessary for storage of the multiresolution hierarchy.

Handling logically non-power-of-two data is handled by combining *slop*, or leftover, zones. A slop zone is shown in Figure 3 occurring at level l. In principle, such a slop zone can occur at any level in the hierarchy. In the one-dimensional case, a slop zone results when the number of zones is odd. Slop zones can also occur in the two-dimensional case, see Figure 4. If a level has an odd number of rows or columns, the level will have slop zones. When constructing the hierarchy for a level l_i where the next level l_{i+1} will have an odd number of rows, we split the children into three, *i.e.* children addresses will be: Address$(l_{i+1}, 2r + \alpha, 2c + \beta)$ where $\alpha \in \{0, 1, 2\}, \beta \in \{0, 1\}$. In the above it is assumed that the columns are split. This scheme specifies how to address children involving slop.

The addressing scheme with respect to determining parents must be slightly modified. When dividing the a zones row and column indices by two, one needs to check whether this is greater than the greatest row or column index at the higher level. This is because slop zones will result in an incorrect address for parent zones. Addressing for siblings and final data will not change.

Considering the one-dimensional example, this means that the last zone in level $l - 1$ has three children instead of two. We record which levels have slop children by storing this information along with the levels. During traversal of the hierarchy, before subdividing a zone, these slop flags are checked in addition to the zone's logical positioning. If the slop flags indicate that a level has slop children, and the zone being subdivided is in the last logical position in the level, then the zone may be divided into three, six or nine children instead of four.

Combining the techniques for handling non-square and non-power-of-two data sets, we are able to build a hierarchical quadtree representation for any $m \times n$ mesh. The algorithms presented in the rest of the section will assume square, power-of-two data sets for the sake of clarity.

2.3. Rendering for minimal zone-size

The minimal-feature approach uses a recursive descent algorithm which systematically selects zones from the hierarchy based on the size of individual zones. The algorithm starts by considering the zone at the top of the hierarchy for rendering. If the zone is inside the field of view and it is larger than the minimal size threshold, the zone is subdivided, and the zone's children are considered. If the zone is in the field of view, but it is smaller than the minimal size threshold, the zone is selected for rendering. We indicate zones that are to be rendered with a flag in the data structure. We use *granularity* as a measure of size in the terms of pixels of the smallest zone that should be subdivided. The size of a pixel in screen coordinates is transformed to object space and then scaled by the granularity factor. The result-

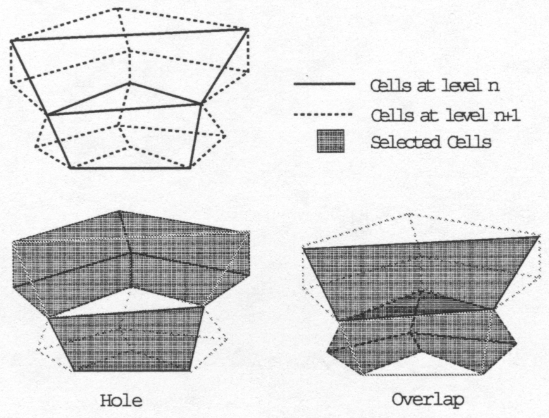

Figure 5. Discontinuity Problems.

ing width in object space is used to determine the smallest zones that should be selected from the hierarchy. Note that if one is creating visualizations where the output zone size will depend on the projection (i.e. three-dimensional view), one should perform the comparison of the object in screen space.

2.4. Rendering for frame-rate or error goals

Here, it is our objective to minimize the error between rendered polygons and the actual data set, while maintaining a desired frame rate constraint. We utilize a greedy algorithm to perform the zone selection. Initially, we subdivide the coarsest representation of the data set, a single quadrilateral, and place this in the priority queue, where priority is given by the error. We iteratively remove the hierarchical zone from the queue with the largest error and subdivide this zone placing its children back onto the priority queue. If a zone is outside the field of view, it is not placed on the priority queue. This process is continued until the number of zones scheduled for rendering, plus those in the queue, is greater than our polygon budget. Our polygon budget is calculated by the number of polygons that can be rendered while maintaining a given frame rate. We utilize a dynamic feedback algorithm so that the frame rate can adjust the polygon budget to the network and machine load.

2.5. Discontinuity problems and constraints

Since the zone selection process chooses zones from different levels of detail, there is the potential for discontinuity problems to occur. This problem manifests itself as holes or overlaps. This is illustrated in Figure 5. The approach we use for dealing with this problem is similar to the crack patching strategy presented in [12]. Vertices of higher resolution zones are constrained to edges of lower-resolution zones by moving the vertices to their projections along the

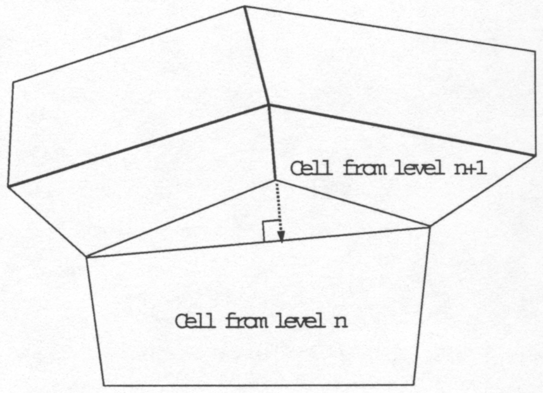

Figure 6. Constraining a vertex by projection onto a lower resolution edge.

edge of the lower resolution zones. This is illustrated in Figure 6. A similar problem occurs with function value data.

Considering our particular implementation, we constrain vertices and function values by making two passes through the hierarchy. The first pass, zone selection, selects nodes using either the minimal size approach, or the frame rate/quality approach. This phase labels zones as:

- **leaf cell**, if the zone was selected to be rendered;
- **branch cell**, if the zone is an ancestor of some selected zone; and
- **out cell**, if the zone is outside the field of view and it and all descendents should not be considered.

In the second pass, a recursive algorithm starts at the root of the quadtree (the coarsest zone) and considers all descendents. If a zone has been labeled a branch zone, the sibling zones are examined to see if they are labeled as leaf zones. if at least one sibling zone is labeled a leaf zone then the vertices of the descendents of the branch zone must be constrained.

The above technique for "patching" holes and removing overlaps provides correct results for "nice" curvilinear grids. The characteristics of a nice grid is one where for any coarse zone the cells in the actual grid lie within the grid. For example: if we have a zone with indices in the final grid $(\vec{X}_{left,top}, \vec{X}_{right,bottom})$, then all grid points $\vec{X}_{i,j}$ such that $left < i < right, top < j < bottom$ must lie within the hierarchical quadrilateral. Examples of some nice and not-nice grids are shown in Figure 7.

3. Results

We use the scheme described in this paper in two different visualization techniques. We support pseudo-color rendering (*i.e.* Gouraud-shading of zones based on function

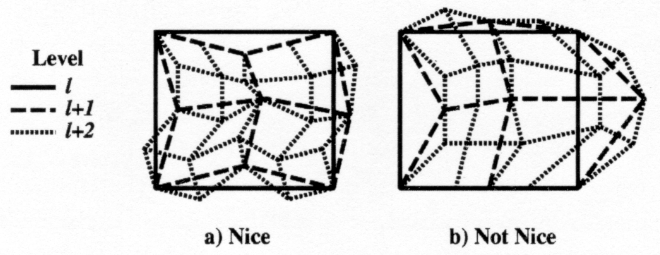

Figure 7. Example "nice" and "not-nice" grids.

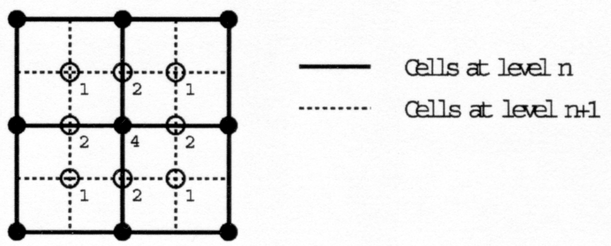

Figure 8. Weighted average of function values.

value) of a data set and mesh plots. The pseudo-color algorithm is time-constrained by the number of polygons rendered. The mesh plot is not constrained by time (rendering lines is quick), but produces aliasing effects if the underlying mesh is too dense.

During the build phase of the algorithm we precompute information that will require too much time to compute during the rendering stage. This precomputation is currently performed on-line since the build stage is relatively inexpensive (on the magnitude of fifteen seconds to a minute). The attributes that are precomputed are:

- **Extensions.** This value allows us to build a bounding box for the zone. The extension value used with the average position of a zone provides enough information to perform field of view clipping during traversal of the quadtree hierarchy.
- **Function value.** Each level of the hierarchy stores an approximation of the function value. We approximate the function value associated with a zone by applying a simple weighting scheme to the underlying mesh. Figure 8 displays two levels of the hierarchy. The function values of the finer mesh are averaged as indicated by the attached weights.
- **Error.** We measure the error between a zone approximating many zones of actual data by using the L^∞ norm (maximum deviation). Thus, we reduce the maximum error between our approximation and the actual data.
- **Size metric.** The size of a hierarchical zone is calculated for use with minimal size rendering. We utilize the minimum diagonal of the zone in approximating its size. This metric is a reasonable estimate of zone size even if a zone is abnormally elongated.

We have tested our multiresolution techniques for a number of very large rectilinear and curvilinear data sets and have observed that real-time visualization is possible for meshes with reasonable errors. Our implementation allows real-time scaling and translation of data sets with the greatest benefits occurring when the imaging resolution is much lower than the resolution of the data set itself or when viewing a zoomed in portion of the data.

no. frames per sec.	no. polygons	% error
10	934	25.8
5	2290	19.3
1	13,654	10.45
0.2	57,884	4.71

Table 1. Frame rates, number of rendered polygons, and error for the San Francisco Bay data set.

The statistics and images we provide in this section are based on our implementation executed on a 250 Mhz SGI Indigo^2 IMPACT workstation with 128 megabytes of RAM. The images were generated at a resolution of 640×640 pixels.

Table 1 provides the results of adaptive rendering performed on a digital elevation model of the San Francisco Bay. This data set is comprised of 1.44 million zones (1200×1200). The table lists the number of zones that were extracted from the hierarchy and rendered (using Gouraud shading) at various frame rates. In the pseudo-color rendered images a scale from blue to green to red is used to indicate function values. The scale for the San Francisco Bay images is shown in Figure 9. The relative error measure that we use in this table is the maximum deviation between the rendered polygons representing a hierarchical zone and the actual data. Figures 11 through 14 show the images produced at the given frame rates and the corresponding surrogate meshes. Figure 15 shows an image of a magnified portion of the data rendered at one frame per second. The number of zones chosen from the hierarchy is 7,956 and the associated error is 2.05%.

We have tested our techniques for plotting grid lines as

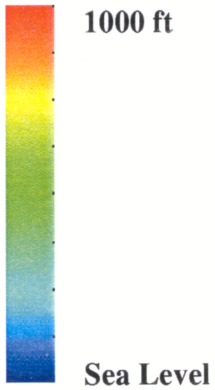

Figure 9. Color bar used for pseudo-color plots.

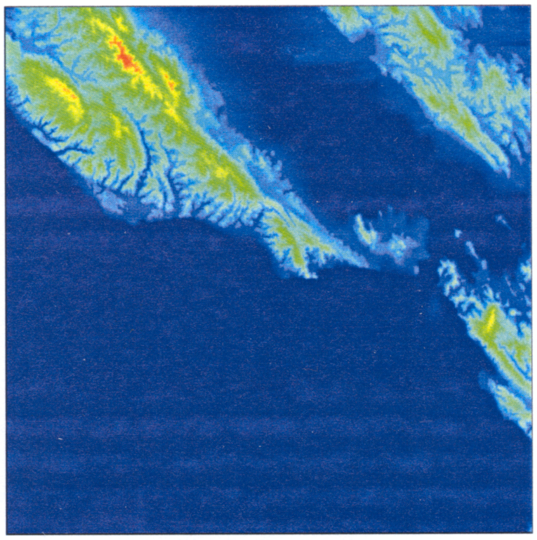

Figure 10. San Francisco Bay Full Resolution (Non-Adaptive).

well. For many applications it is just as important to visualize and understand the underlying mesh structure as it is to analyze function values. Figure 16 demonstrates the importance of minimal-feature rendering when overlaying plots. In the context of mesh line rendering, the adaptive approach that we use for minimal-feature rendering allows us to generate images that do not contain zones occupying fewer than a certain number of pixels. This is crucial to avoid rendering images that do not reflect the underlying mesh structure at all due to the fact that zone size is less than a single pixel.

4. Conclusions

We have presented a simple and robust approach for the representation of two-dimensional curvilinear data. This approach allows one to render large data sets in real time by utilizing a precomputed hierarchy. Based on user specifications, our rendering algorithm selects the level in this hierarchy that is appropriate for either rendering a certain number of frames per second or rendering graphical primitives that are not smaller than a certain number of pixels.

We represent the information needed to describe and traverse the hierarchy using a minimal amount of memory. For moderately sized data sets one can generate the hierarchy in a matter of seconds. Thus, a stand-alone pre-processing step to construct the hierarchy is only necessary for extremely large data sets. The selection process of a set of zones is efficient – it hardly impacts the rendering process itself, *i.e.*, the selection process does not cause any bottlenecks for image generation. Low-error images can be produced at relatively high frame rates.

Our current work suggests several avenues for future work. Two such avenues include extending these techniques to an octree representation of three-dimensional data and generalizing this approach for "non-nice" curvilinear grids, and unstructured data sets.

5. Acknowledgments

This work was funded by Lawrence Livermore National Laboratory (LLNL) under contract W-7405-ENG-48 to the University of California, Davis. We would like to thank the members of the Visualization Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis, and the members of the MeshTV development team at LLNL for their help.

References

[1] G. P. Bonneau, S. Hahmann, and G. M. Nielson. BLaC-wavelets: A multiresolution analysis with non-nested spaces. In R. Yagel and G. Nielson, editors, *Visualization '96*, pages 43–48. IEEE Computer Society Press, 1996.

[2] A. Certain, J. Popović, T. Derose, T. Duchamp, D. Salesin, and W. Stuetzle. Interactive multiresolution surface viewing. In *SIGGRAPH 96 Conference Proceedings*, pages 91–97. ACM SIGGRAPH, 1996.

[3] P. Cignoni, L. D. Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution modeling and visualization of volume data based on simplicial complexes. In A. E. Kaufman and W. Krüger, editors, *1994 Symposium on Volume Visualization*, pages 19–26, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[5] L. D. Floriani. A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics & Applications*, 9(2):67–78, 1989.

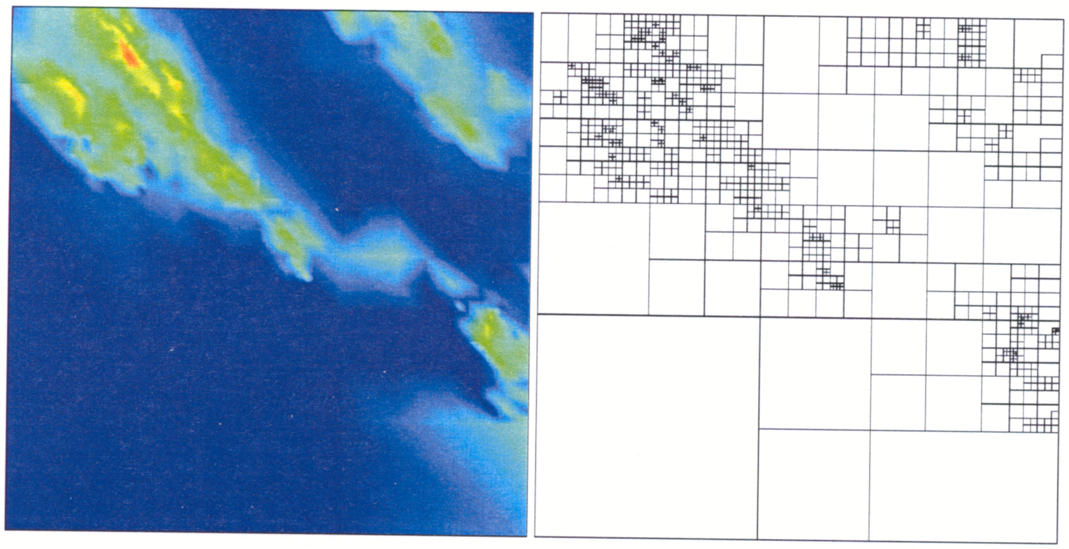

Figure 11. San Francisco Bay at 10 fps.

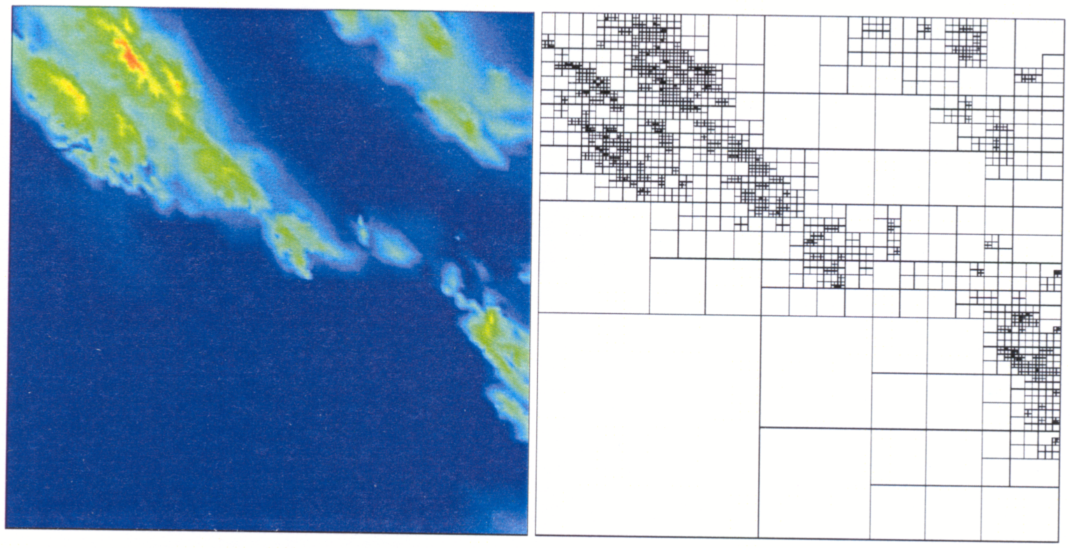

Figure 12. San Francisco Bay at 5 fps.

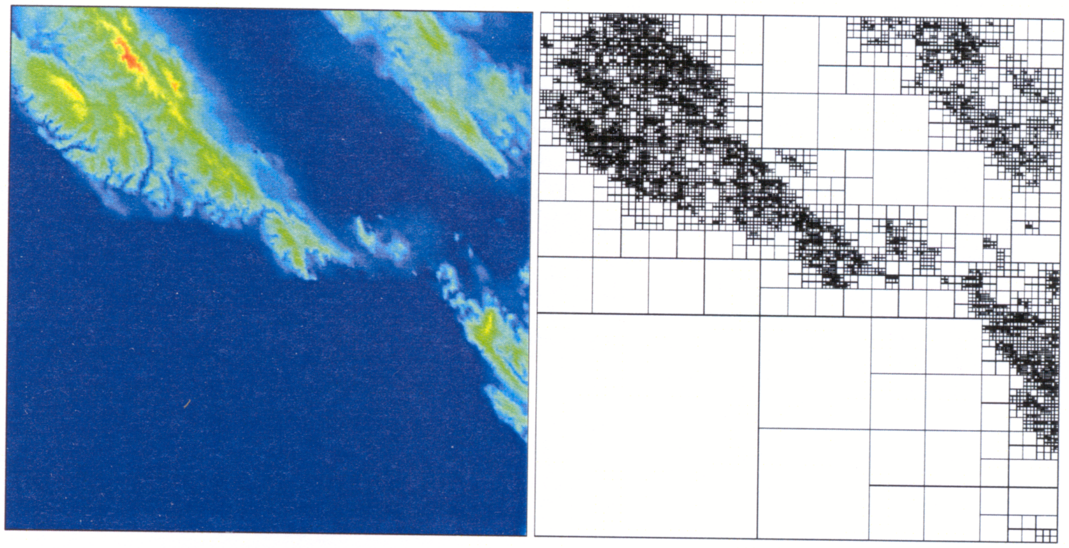

Figure 13. San Francisco Bay Render at 1 fps.

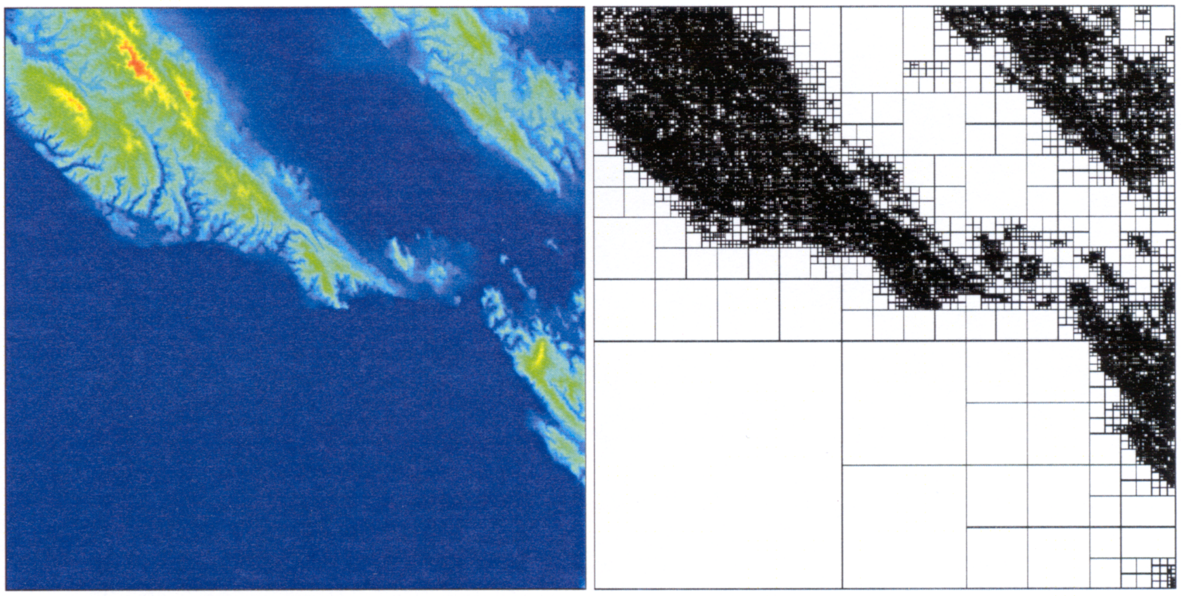
Figure 14. San Francisco Bay at 0.2 fps.

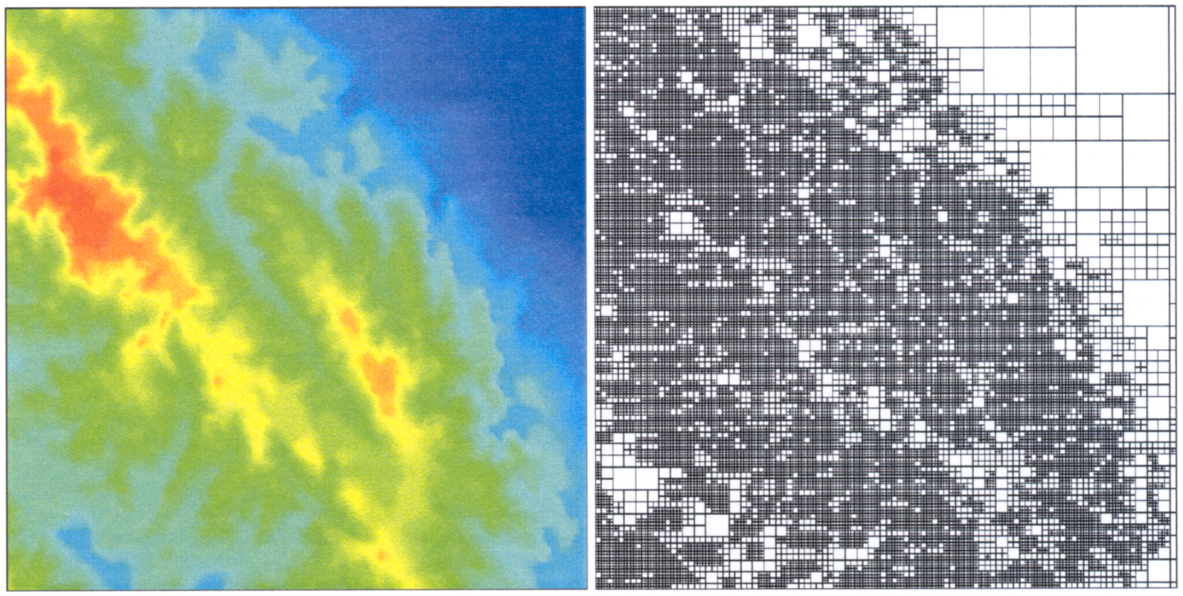
Figure 15. San Francisco Bay at 1 fps (Zoom 6x).

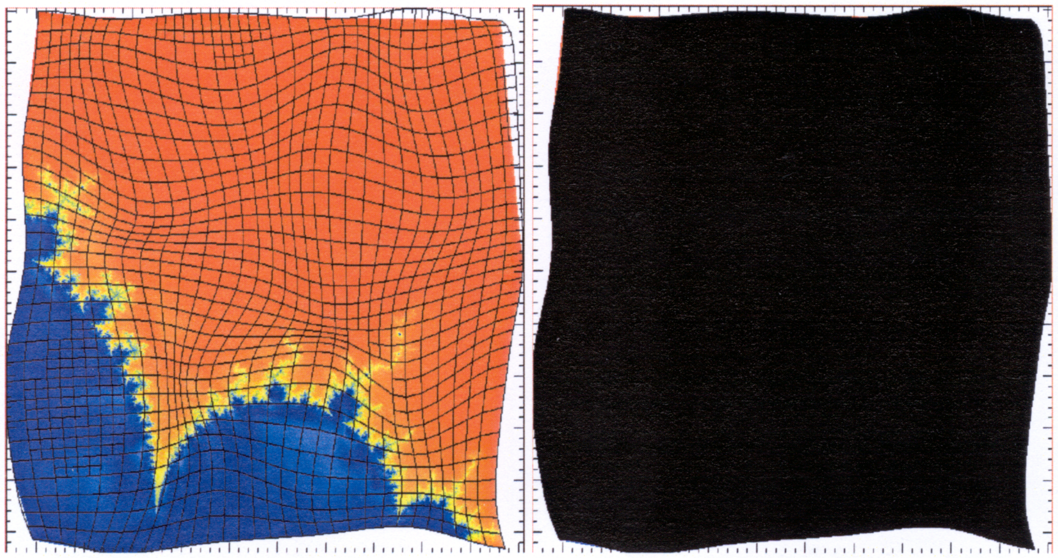

Figure 16. Perturbed Mandelbrot set with mesh grid overlay. We show using a 32 pixel minimal feature constraint (left), and a mesh plot without the minimal feature constraint (right).

[6] M. H. Gross, R. Gatti, and O. Staadt. Fast multiresolution surface meshing. In G. M. Nielson and D. Silver, editors, *Visualization '95*, pages 135–142. IEEE Computer Society Press, 1995.

[7] H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 99–108. ACM SIGGRAPH, 1996.

[8] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fields. In *SIGGRAPH 96 Conference Proceedings*, pages 109–118. ACM SIGGRAPH, 1996.

[9] M. C. Miller. *Multiscale compression of digital terrain data to meet real time rendering constraints*. PhD thesis, Department of Electrical and Computer Engineering, University of California, Davis, 1995.

[10] H. Samet. The quadtree and related hierarchical data structures. *Computing Surveys*, 16(2):187–260, 1984.

[11] H. Samet. *The design and analysis of spatial data structures*. Addison Wesley, Reading, MA., 1990.

[12] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill. Octree-based decimation of marching cubes surfaces. In R. Yagel and G. M. Nielson, editors, *Visualization '96*, pages 335–342. IEEE Computer Society Press, 1996.

[13] H. Tao and R. J. Moorhead. Progressive transmission of scientific data using biorthogonal wavelet transforms. In D. Bergeron and A. E. Kaufman, editors, *Visualization '94*, pages 93–99. IEEE Computer Socieity Press, 1994.

[14] R. Westermann. Compression domain rendering of time-resolved volume data. In G. M. Nielson and D. Silver, editors, *Visualization '95*, pages 135–142. IEEE Computer Society Press, 1995.

[15] J. Wilhelms and A. V. Gelder. Octrees for faster isosurface generation. *ACM Transactions on Computer Graphics*, 11(3):201–227, 1992.

[16] J. C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal meshes. In R. Yagel and G. M. Nielson, editors, *Visualization '96*, pages 327–334. IEEE Computer Society Press, 1996.