Seamless Multiresolution Isosurfaces Using Wavelets

Tushar Udeshi Randy Hudson*
Michael E. Papka†

Futures Laboratory
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
{udeshi,hudson,papka}@mcs.anl.gov

Abstract

Data sets that are being produced by today's simulations, such as the ones generated by DOE's ASCI program, are too large for real-time exploration and visualization. Therefore, new methods of visualizing these data sets need to be investigated. We present a method that combines isosurface representations of different resolutions into a seamless solution, virtually free of cracks and overlaps. This technique combines existing isosurface generation algorithms and wavelet theory to produce a real-time solution to multiple-resolution isosurfaces.

Keywords: Multiresolution isosurfaces, wavelets, crack removal

1 Introduction

The extraction of polygonal isosurfaces is a widely used visualization method for scalar fields in volumetric data. It is especially useful for visualizing data sets containing objects having well-defined boundaries, where lighting and shading of the polygonal surfaces enhance the 3D structures. A common algorithm used for isosurface extraction is marching cubes [8]. This algorithm traverses all cells of the volume and generates the polygonal isosurface by trilinear interpolation along the edges of the cells. The algorithm is trivially parallelizable, and we have been able to perform isosurface extraction, using a parallel implementation that is part of the *Visualization Toolkit* (VTK) [18], of larger data sets (1024^3 regular grids) in real time [7]. Even though the production of the polygonal surfaces is done in a reasonable time, the number of polygons generated by these algorithms quickly exceeds the capabilities of the rendering hardware.

Isosurfaces are also a commonly used visualization technique within virtual reality environments because they make full use of the three-dimensional space and the creation of the surface is fast enough for real-time exploration of moderately sized data sets [15, 17]. As data sets grow larger, however, it is becoming difficult to use virtual environments for investigation. Again this difficulty is due not to the generation time of the surfaces, but to the number of polygons generated. One can decimate the resulting surface, but this approach adds to the computation time of the final surface and is not acceptable in a real-time environment. Therefore, one must explore methods to calculate isosurfaces with polygon counts that are reasonably sized for real-time rendering.

This paper describes a technique that enables the exploratory scientific visualization of large data sets using isosurfaces on a wide variety of machines, from low-end graphics workstations to high-end virtual reality systems. This technique allows the user to modify the isosurface threshold interactively, as well as reduce the number of triangles generated so that they can be displayed in real time. The user can also choose a region of interest that is displayed in high resolution while the rest of the data set is displayed at a lower resolution. The ability to show both high- and low-resolution isosurfaces at the same time allows the user to maintain real-time frame rates from the rendering engine, look at a particular region of the data in full detail, and have a sense of the global context of the data set as well. The technique uses the well-studied wavelet transform to construct multiple resolutions of rectilinear data while introducing minimum error. In the following sections of the paper we show how wavelet coefficients can be manipulated at the boundary between two resolutions to ensure continuity of the function, and we present an efficient technique that ensures continuity of the isosurface generated. The isosurface has no overlap, but in the worst case hairline cracks may appear at multiresolution boundaries. However, these cracks are insignificant and, we believe, do not hinder the process of understanding and visualization of the data set.

2 Background

Several attempts have been made to reduce the number of triangles generated by the marching cubes algorithm. The most notable is the decimation algorithm by Schroeder et al. [19]. A full high-resolution isosurface generated by marching cubes is substituted with a simpler mesh generated with only a subset of the original vertices. This approach is too computationally expensive, however, because a large number of triangles are first generated, only to be eliminated at a later stage.

Several adaptive marching cube algorithms have been proposed [21, 10, 20, 11]. These techniques traverse all voxels and at least partially extract the isosurface for each voxel. Neighboring voxels are then merged if the isosurface is found to be similar. However, these techniques are inappropriate for exploratory scientific visualization because merging of cells is directly linked to the isosurface threshold; that is, at least a partial full resolution isosurface is extracted and the merging process repeated every time the user modifies the isovalue. Moreover, additional memory is required to store the multiple resolutions.

The benefits of octrees for faster reconstruction of isosurfaces for regular volume data was first recognized by Wilhelms and Gelder [26]. The minimum and maximum density value of the subtree rooted at every inner node are stored. Large branches of the volume that do not intersect the isosurface can thus be skipped. Westermann et al. [25] used average pyramid octrees for exploratory visualization in which the level of refinement of a particular inner node is determined by a user-defined focal point and radius of interest. Their technique requires neighboring cells to vary by not more than one level of resolution. Extra processing is required to maintain continuity of the data set at multiresolution boundaries. The technique generates additional polygons to fill in cracks at multiresolution boundaries. Moreover, additional memory is required to store the full octree.

Wavelet-based techniques are commonly used for extracting multiresolution representations of a function. Mallat [9] proposed a framework for multiresolution decomposition of a measurable, square-integrable, one-dimensional function $f(x)$. The model is extensible to higher dimensions. An approximation operator $\mathbf{A}_{2^j}$ is defined that projects $f(x)$ at the resolution 2^j. Among all approximated functions at resolution 2^j, $\mathbf{A}_{2^j} f(x)$ is the function most similar to $f(x)$. More formally, assuming $\mathbf{V}_{2^j}$ to be the set of all possible approximations at the resolution 2^j of all measurable,

*Also member of ASCI Flash Center, University of Chicago

†Also member of Computation Institute and Department of Computer Science, University of Chicago

square-integrable one-dimensional functions, this can be expressed as follows:

$$\forall g(x) \in \mathbf{V}_{2^j}, \|g(x) - f(x)\| \geq \|\mathbf{A}_{2^j} f(x) - f(x)\| \quad (1)$$

The multiresolution transform is implemented efficiently by the "cascade" algorithm, which successively decomposes a signal $\mathbf{A}_{2^{j+1}} f(x)$ to a coarser signal $\mathbf{A}_{2^j} f(x)$ and a "detail" signal $\mathbf{D}_{2^j} f(x)$ (also called wavelet coefficients). The wavelet coefficients store the difference between the two resolutions and are computed by projecting the signal onto orthogonal wavelet basis functions. This transform is fully reversible and requires no additional storage. The cascade algorithm runs in linear time with respect to the number of samples of the original discrete signal.

In volume visualization only finite signals are considered, while Mallat's multiresolution transform is designed for infinite signals. Using symmetric or antisymmetric wavelet basis functions allows for (anti)symmetric extensions of the function at the boundary. Therefore, it is desirable for the wavelet basis functions to be (anti)symmetric when the signal is defined only over a finite interval. It has been proven that compactly supported, (anti)symmetric, orthogonal wavelet basis functions of degree greater than zero cannot be constructed [1]. On the other hand, relaxing the orthogonality condition allows the construction of smooth, symmetric, and compactly supported biorthogonal wavelets. The error introduced by performing a multiresolution transform using biorthogonal wavelet basis functions is within a small constant of the minimum possible error (as described in Equation 1) [1]. Moreover, extremely efficient, in-place computation of the biorthogonal wavelet transform is possible using the lifting scheme [23].

Several researchers have used wavelets to obtain multiple resolutions of three-dimensional rectilinear data [12, 6, 24, 16]. However, none of these techniques allow the user to specify a region of interest or permit multiple resolutions in different regions of the data at the same time. Gross et al. [4] use wavelet decomposition to adaptively generate polygonal data representing a terrain from a height field. They also modify wavelet coefficients to define a region of interest. However, extra processing is involved, requiring lookup into a table with 625 entries to maintain continuity of the polygonal mesh at multiresolution boundaries.

3 Theory

In this section we first justify our choice of subsampling technique (Section 3.1). We then show how we modify wavelet coefficients to ensure continuity of the function at multiresolution boundaries (Section 3.2). We conclude this section by describing an efficient technique of eliminating cracks in the isosurface at multiresolution boundaries (Section 3.3).

3.1 Subsampling Technique - Wavelet Decomposition Using Linear Biorthogonal Basis

The marching cubes algorithm assumes that the scalar field is piecewise linear along the cell edges; the intersection of the isosurface along an edge of the data set is determined by linearly interpolating between the two boundary data points of the edge. Linear wavelet basis functions have one vanishing moment. Hence, if one iteration of the cascade algorithm is performed using linear wavelet bases on a linear function, the function is represented exactly. When higher-order wavelet basis functions are used, even higher-order functions can be represented exactly. However, higher-order basis functions have a wider support, thus increasing the computational cost of the multiresolution transform. Therefore, linear basis functions are best suited to represent piecewise linear functions. Hereafter we refer to decomposition of a signal at resolution 2^j into a signal at resolution 2^{j-1} and wavelet coefficients using the lifting scheme with linear biorthogonal wavelet basis functions as one iteration of the *forward wavelet transform*. A reconstruction of a signal from a coarser resolution signal and wavelet coefficients using the same algorithm is referred to as the *reverse wavelet transform*.

Table 1: Error introduced by the three subsampling techniques. Note that the range of data values in this data set is 0 to 255.

Subsampling Technique	L1 Error	RMS Error
Average pyramid with box filter	15.5254	34.9054
Accepting every eighth sample	13.0492	33.2336
Three iterations of the forward wavelet transform	10.6413	27.0047

Figure 1 shows the effect of subsampling using three techniques. An isosurface is extracted after three levels of subsampling of the iron protein data set[1]. Note that after this subsampling process, only 0.195% of the data is retained. Figure 1(a) is obtained by averaging every $8 \times 8 \times 8$ subgrid, which can also be considered as three iterations of the cascade algorithm using Haar wavelets [5]. Clearly, there is more severe loss of detail here than with the other two methods. This is because the Haar wavelet basis functions have zero vanishing moments and are thus suited to represent only piecewise constant functions, whereas the marching cubes algorithm assumes the scalar field to be piecewise linear. The naive strategy of skipping over samples may cause considerable aliasing, as seen in Figure 1(b). Table 1 gives the L1 and the root mean square (RMS) error introduced by the three methods of subsampling. All subsampled data sets were first supersampled back to the original resolution by trilinear interpolation and then compared against the original iron protein data set. Again, this is justified because the isosurface extraction algorithm assumes the scalar field to be piecewise linear along its edges.

3.2 Ensuring Function Continuity

In order to ensure continuity of the isosurface generated, it is imperative to ensure continuity of the function at the boundary between two resolutions. We demonstrate this in two dimensions with the help of Figure 2. We need to ensure $u1 = u2$, $d1 = d2$, and $m = \frac{u1+u2}{2}$, again assuming piecewise linearity of the function.

If the subsampling strategy used is to skip over samples, as was done in Figure 1(b), the function will always match at the data samples common to the two resolutions ($u1 = u2$ and $d1 = d2$ in Figure 2), whereas other samples at the higher resolution (m in Figure 2) will need to be explicitly recalculated by linearly interpolating the lower-resolution values. This was done in [20]. If average pyramids are used as the subsampling strategy, as was done in Figure 1(a), the function is not guaranteed to be continuous even at the data points shared between the two resolutions. In [25], data values at the coarser level are modified to be equal to the corresponding data values at the finer level ($u1$ is set to $u2$ and $d1$ is set to $d2$ in Figure 2). Other samples at the finer level are recalculated by linear interpolation (m is set to $\frac{u1+u2}{2}$ in Figure 2).

We use a linear biorthogonal wavelet transform implemented by the lifting scheme [23] to obtain multiple resolutions of our data set. The lifting scheme does an in-place computation of this transform and is composed of two stages: *predict* and *update*. During the *predict* stage of the forward wavelet transform, in each iteration, all odd-indexed samples are replaced by wavelet coefficients.

[1] Available with the standard VTK data distribution courtesy Kitware Inc. 68^3 regular grid

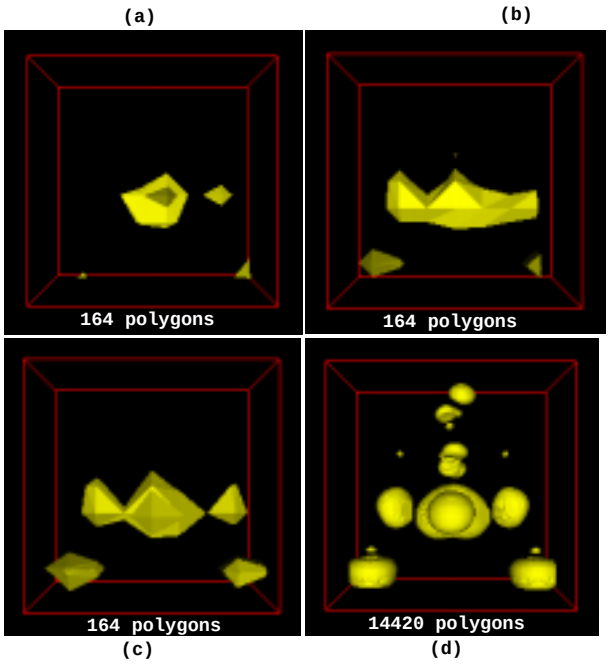

Figure 1: Isosurface of iron protein data set with a threshold of 127; (a), (b), and (c) are subsampled by a factor of eight in all three dimensions. (a) Data set subsampled using average pyramid and box filter; (b) data subsampled by skipping over every eight samples; (c) data subsampled by applying linear, biorthogonal wavelet transform three times; (d) the original data set.

The opposite occurs during the *predict* stage of the reverse wavelet transform. During the *update* stage, all even-indexed samples are transformed to represent the function at a lower or higher resolution in the forward or reverse wavelet transform, respectively. For example, in Figure 2, the low-resolution region is generated by two iterations, and the high-resolution region by one iteration, of the forward wavelet transform.

When using linear basis functions while performing the wavelet transform, a wavelet coefficient at a particular point x gives an indication of how far the function is from being linear at x. A wavelet coefficient of zero at x indicates that the function is perfectly linear at x; so if a reverse wavelet transform is done to transform the function one level of resolution higher, the data value at x will be a linear interpolation of its neighboring data values at the higher resolution. This is illustrated for the one-dimensional case in Figure 3.

When using linear basis functions, the value by which an even-indexed sample is altered during the *update* stage is a function of only its surrounding wavelet coefficients. Therefore, in one dimension, if the two neighboring wavelet coefficients of a data value at a point x are zero, the data value remains unchanged when transformed one resolution higher by performing one iteration of the reverse wavelet transform. This is illustrated in Figure 4. Similarly, in two dimensions, 8 wavelet coefficients *update* a data point, while in three dimensions the count is 26.

The observations of the preceding two paragraphs give us a framework for ensuring continuity of the function at the boundary between two resolutions. For the following discussion, refer to Figure 5. Assume for clarity of explanation that the data set is n dimensional ($n = 1, 2, 3$) and of resolution s in all n dimensions. Let $\mathbf{l}$ and $\mathbf{h}$ be user-defined n-dimensional points defining an n-dimensional rectangular region that forms a boundary to the high resolution region. Assume that the user sets the low-resolution region to one-level subsampling (one iteration of the forward wavelet

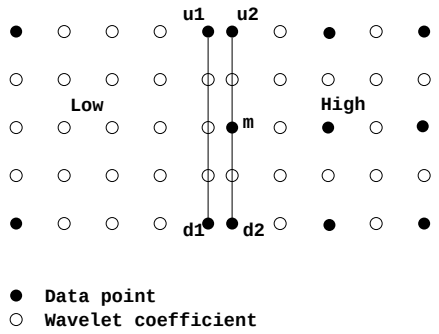

Figure 2: Multiresolution boundary

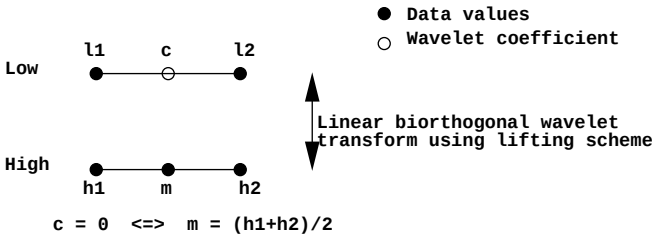

Figure 3: A zero wavelet coefficient at a point x implies that the function is linear at x.

transform achieves this) and the high-resolution region at the resolution of the original data set (the highest resolution). Note that $0 <= l_i < h_i < s, i = 0, 1, ..n-1$. Also note that l_i and $(s - h_i)$ need to be even so that they may bound the one-level subsampled low-resolution regions. Here is the sequence of steps required to generate a seamless multiresolution function in this particular example:

1. Perform one iteration of the forward wavelet transform to the original data set to bring the entire data set one level of resolution down.

2. Modify wavelet coefficients to ensure continuity. Assume that w_i is the position of a wavelet coefficient W in the i^{th} dimension. Set $W = 0$ if $w_i <= l_i + c_w$ or $w_i >= h_i - c_w$, where c_w is called the *cushion* width. This basically nullifies all wavelet coefficients inside the low-resolution region and inside a cushion at the boundary of the high-resolution region. This is illustrated in two dimensions in Figure 5. The cushion width c_w is half the width of a low-resolution voxel, which in the above example is 2, and so c_w is 1.

3. Do one iteration of a reverse wavelet transform.

Setting wavelet coefficients to zero as described above ensures that the data values in the low-resolution region do not change while doing the reverse wavelet transform (step 3 above). Also all other data values in this region are forced to be linearly interpolated from their neighboring data values and that includes the data values at the boundary between the two resolutions. This ensures continuity of the function.

In the above example, the two resolutions varied by only one level. However, this is not a requirement. The two resolutions can vary by arbitrary amounts as long as the cushion is set to half the width of the low-resolution voxel. This guarantees that at every iteration of the reverse wavelet transform, wavelet coefficients on and outside the boundary of the high-resolution region as well as those that line the boundary inside the high-resolution region are

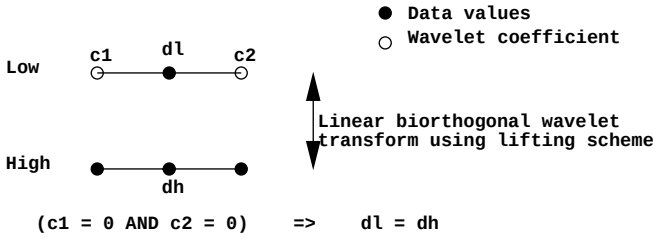

Figure 4: A data value remains unchanged between two resolutions if its surrounding wavelet coefficients are zero.

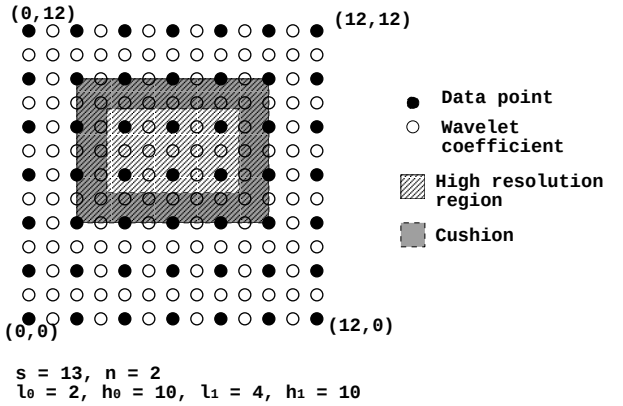

Figure 5: Smooth transition between two resolutions with the help of a cushion

zero. This, in turn, ensures that after the reverse wavelet transform, the data values at the coarser resolution on and outside the high-resolution boundary are unchanged. Also, the new data values generated on and outside the high-resolution boundary during the *predict* stage of the lifting scheme are guaranteed to be linearly interpolated from the neighboring data values at the user-defined low-resolution level.

What this technique effectively does is create a cushion of voxels at the boundary that forms a smooth transition between the two resolutions; one side of the cushion is at a high resolution, while the other side is at a low resolution.

A More Efficient Algorithm

The algorithm described above is a little wasteful. A reverse wavelet transform is applied to the low-resolution region even though the data values there are unaffected by this transform. However, a simple improvement to the algorithm can eliminate this waste. Here is the modified algorithm:

1. Transform the entire data set to the user-specified low resolution.

2. Set all wavelet coefficients in the cushion region (See Figure 5) to zero.

3. Apply the reverse wavelet transform only to the high-resolution region (which includes the cushion) to the level specified by the user.

This not only reduces the computational cost of transforming between the two resolutions but also reduces the amount of storage required to store the wavelet coefficients that are set to zero. It is important to have the ability to recover the data set because the user should be able to move the high-resolution region as well as change the two resolution levels. We now need to store only the wavelet coefficients in the cushion region in order to recover the data set.

3.3 Ensuring Isosurface Continuity

In Section 3.2, we described a method to ensure function continuity in a linear sense at a multiresolution boundary. When we apply the marching cubes algorithm to such a function in two dimensions (called marching squares), the resulting contour lines generated are continuous. Figure 6 illustrates this. However, this is not generally true in the three-dimensional case. In the marching cubes algorithm, the function is assumed to be linear only along the edges. The intersection of the isosurface is calculated only along the edges of the cubical voxels. Adjacent intersection points are connected to form contours, which are then triangulated to give isosurface geometry. Note that if the function were trilinearly interpolated at every point in the volume, as is done in volume rendering [2], or even bilinearly interpolated along the faces of the data set, the resulting isosurface would be continuous along the multiresolution boundary, given that the scalar field is continuous.

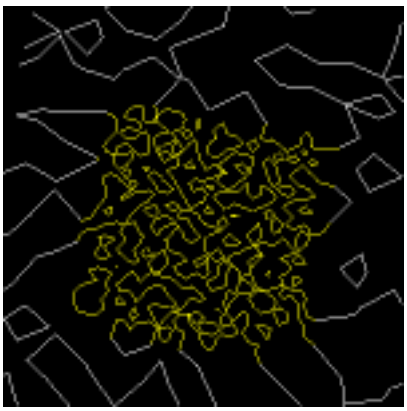

Figure 6: In two dimensions, continuity of the function ensures continuity of the isolines. The white contour lines are generated from a region that is two levels of resolution lower than the high-resolution region from which the yellow contour lines are generated. The data set was generated as a two-dimensional array of random numbers.

Consider a face of a low-resolution cube at a multiresolution boundary. Assuming the scalar value at a vertex point to be its third dimension, the four points at the vertices of this face may be nonplanar. In this case, the approximation of the scalar field used by marching cubes (as described in the previous paragraph) is faceted. Assume that all wavelet coefficients surrounding the vertices are set to zero, as described in Section 3.2. Then, the points introduced along the edges of the face during the *predict* stage of the reverse wavelet transform do not change the points of intersection of the isosurface with that face. This is because the new data points introduced along the edges are linearly interpolated from the edge points. However, the scalar value at the center of the face, which is an average of the scalar values at the four boundary vertices, may not lie on the faceted scalar field approximation. The different resolution isolines thus generated will be coincident along the edges of the face but not inside the face, as was observed in [25]. This results in cracks at the multiresolution boundary, as illustrated in Figure 7.

Fortunately we found an efficient and easy-to-implement way around this problem. Refer to Figure 8(a). Consider a nonplanar multiresolution boundary face F with vertices v_1, v_2, v_3, v_4. Assume the isosurface intersects F along adjacent edges, $\overline{v_1v_2}$ and $\overline{v_2v_3}$. This is equivalent to saying that the approximated scalar field

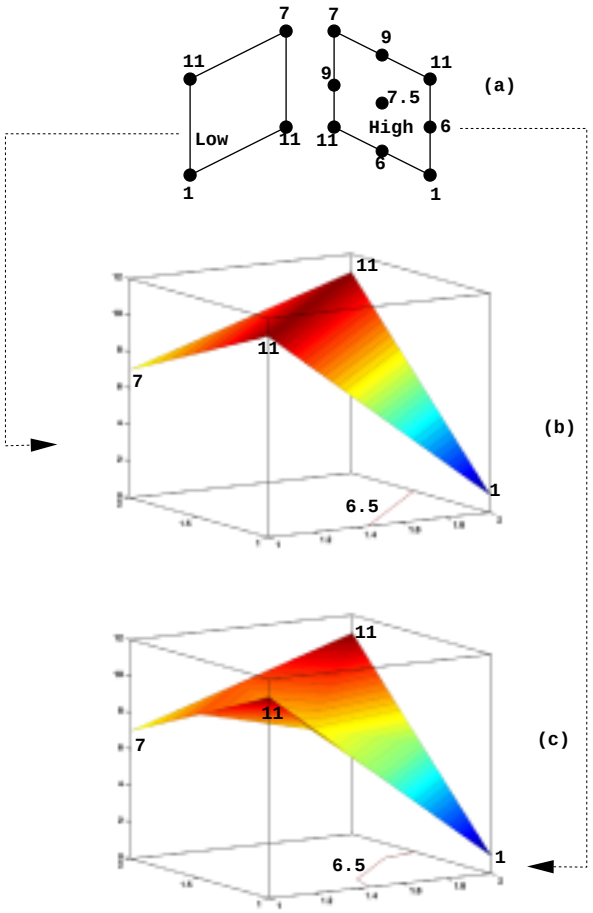

Figure 7: A crack is formed in spite of the function being continuous at the boundary if the four vertices at a face are non-coplanar: (a) is a face at a boundary where the resolutions differ by one level. (b) and (c) give the surface plots of the boundary face at low and high resolutions, respectively, as well as a contour line shown in red with an isovalue of 6.5. Note the fold in the low-resolution surface plot. The two surface plots differ at the center at the cell. The contour lines coincide along the edges of the face but not inside it.

has a fold along the diagonal $\overline{v_1 v_3}$. After nullifying the wavelet coefficients in the cushion and doing one level of a reverse wavelet transform, we go up by one resolution. We then modify the scalar value at center of F, c, to lie along the fold, that is, we set its value to $\frac{v_1+v_3}{2}$.

Refer now to Figure 8(b) and (c). Assume that the isosurface passes though opposite edges of F, $\overline{v_1 v_2}$, and $\overline{v_4 v_3}$. After going up one resolution, the scalar values at the center of F may have to be adjusted so that the isolines generated coincide with the low-resolution isoline on F. This is done by pulling the scalar value at the center of F up or down, until the new, high-resolution isoline bumps against the low-resolution isoline. In more detail, this is accomplished as follows:

1. Calculate the intersection point p of the low-resolution isoline with the edge $\overline{m_1 m_2}$ that connects the midpoints of the other two edges ($\overline{v_4 v_1}$ and $\overline{v_2 v_3}$).

2. Set the scalar value at the center of F, c, so that the intersection of the high-resolution isosurface with F remains unchanged along $\overline{m_1 m_2}$ (i.e., the high-resolution isoline passes through p).

Assuming that the isovalue threshold is iso, $\mathbf{v}(x)$ is the scalar value at location x, and referring to Figure 8(b) and (c), the scalar value at c is changed according to the following pseudocode:

```
/* calculate e = scalar value at p */
let e = 1/2 × { (iso − v(v4)) / (v(v3) − v(v4)) + (iso − v(v1)) / (v(v2) − v(v1)) }
if (e > 0.5) /* Figure 8(b) */
    let e = 2 × e − 1
    let v(c) = (iso − e × v(m2)) / (1 − e)
else /* Figure 8(c) */
    let e = 2 × e
    let v(c) = (iso − v(m1)) / e + v(m1)
endif
```

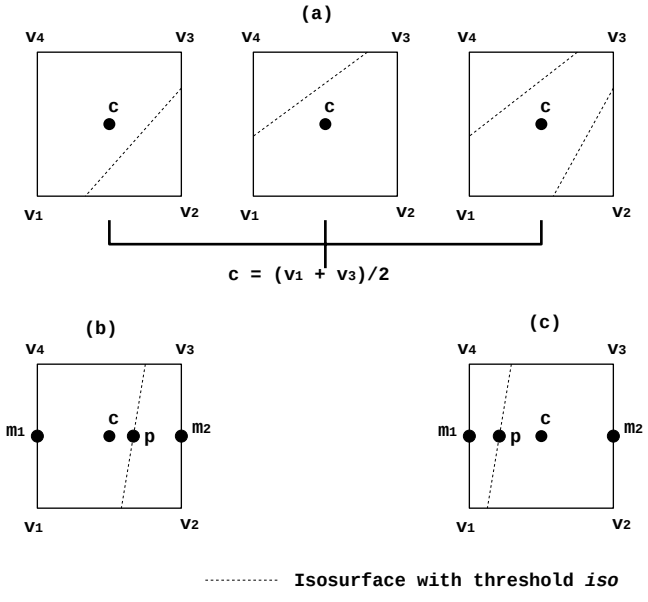

Figure 8: Modifying the center value of a boundary face to ensure continuity of the isosurface. Note that rotationally symmetric cases have been omitted: (a) isosurface intersects adjacent edges of boundary face; (b) and (c) isosurface intersects opposite edges.

Refer to Figure 9. In this case the isoline passing through opposite edges of F intersects with two high-resolution edges of the

data set on F. There can be instances in which it is impossible to adjust the scalar value at c so that the high-resolution isoline passes through both p and q. In our implementation, we adjust the scalar value at c so that the isoline passes through p. This guarantees that the two isolines at different resolutions coincide along at least half the face F (along $\overline{p}r$ in Figure 9). Hairline cracks may appear on the other half. Figure 10 shows such a crack. We have experimentally found that only approximately 4.19% of multiresolution boundary faces, which intersect the isosurface, develop these cracks; the maximum area of the triangle forming the crack is 2.84% of the area of the low-resolution boundary face, while the average crack area is 1.22%. We believe that these cracks do not hinder the visualization of the data set.

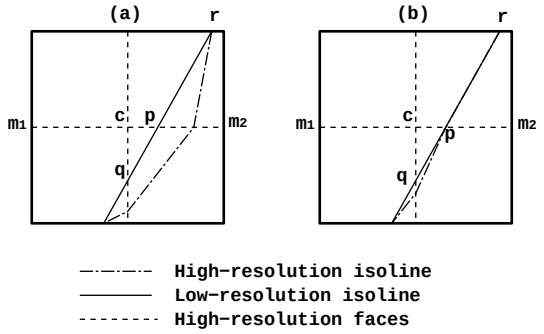

Figure 9: (a) A low-resolution isoline may intersect two high-resolution edges. (b) Our correction forces the high-resolution isoline to coincide with one intersection point(p) but not both. Thin cracks may result.

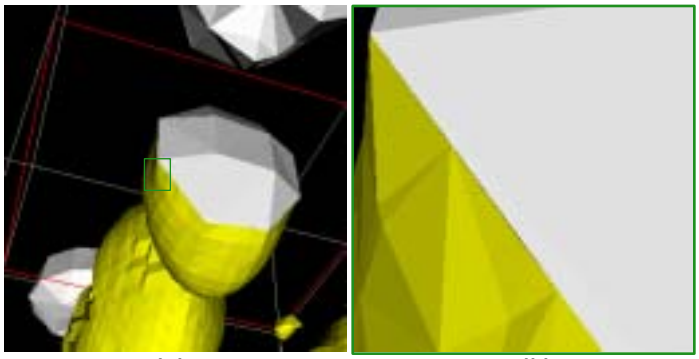

Figure 10: An example of a minute crack generated in the iron protein data set. (a) The global view showing the boundary of the region where the crack exits as a green rectangle; (b) the crack magnified.

Note that the modification of the data set introduced above is easily reversible. Before performing any iteration of the forward wavelet transform involving only the high-resolution region, the scalar values at the centers of all faces along multiresolution boundaries need to be restored. This is done by simply setting the scalar value at the center of a face to the average of the scalar values at its four vertices.

Our technique fills cracks by modifying sparse data values along multiresolution boundaries so that the high-resolution edges are forced to coincide with the corresponding low-resolution edges. The technique is efficient because the change at a particular vertex is a function of only its immediate neighbors and the current isovalue. The change is also trivially reversible. Figure 11 shows isosurface extraction with and without this correction applied to the boundaries.

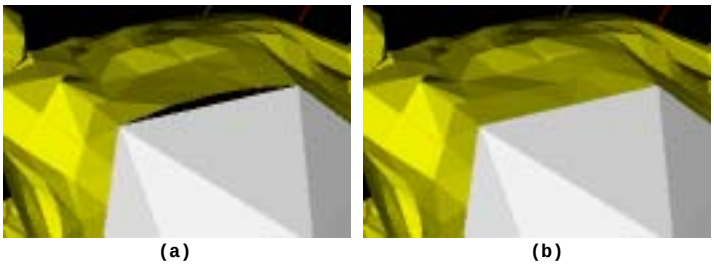

Figure 11: Fixing the cracks by modifying data points at boundaries. The yellow region is the isosurface generated at the highest resolution; the white region is the isosurface generated from down sampling by two levels: (a) without the correction, cracks visible (b) With the correction, no cracks.

Marching Cubes Ambiguities

Up to now, we have discussed techniques to ensure continuity of the isosurface. Note that the isosurfaces of the different resolution regions are extracted with separate calls to the marching cubes algorithm. An ambiguous face [13] may exist at a multiresolution boundary. In order to ensure topological consistency across the boundary, the same disambiguation choice needs to be made at neighboring multiresolution cells. We use the so-called single-entry cubical table marching cubes technique [14]. In this technique, an arbitrary choice is used at the ambiguous face. However, neighboring cells produce the same contours at the common face, thus maintaining topological consistency.

4 Implementation

We built our system using the Visualization Toolkit (VTK) [18], which is free software.[2] The code is written using C++. Isosurface extraction is done using a VTK function called synchronized templates, which is an improved version of standard single-entry cubical marching cubes. Therefore, marching cubes ambiguities at multiresolution boundaries, as discussed in Section 3.2, are taken care of.

We built our biorthogonal wavelet transform functions starting with free software[3] called LIFTPACK [3]. LIFTPACK supports one- and two-dimensional biorthogonal wavelet transforms of arbitrary order using the lifting scheme. We extended the transform to three dimensions and reduced its computational requirement, as explained below.

LIFTPACK uses the so-called non standard technique [22] to construct two-dimensional wavelet bases from one-dimensional basis functions. Assuming `1D_Transform` is a function performing one iteration of a forward wavelet transform in one dimension, one can express a forward wavelet transform in two dimensions, using the non standard technique, algorithmically as follows:

```
for each row r
    1D_Transform(r)
endfor
for each column c
    1D_Transform(c)
endfor
```

[2] Available at http://www.kitware.com/vtk.html

[3] Available at http://www.cs.sc.edu/~fernande/liftpack/

Note that in the algorithm, the wavelet coefficients generated after applying the transform to the rows are again subjected to a forward wavelet transform when iterating over the columns. We modified the second loop of the algorithm so that `1D_Transform` is not applied to columns containing only wavelet coefficients (i.e., in two dimensions, we skipped over every alternate column). If the data set size is $s \times s$, the number of data elements that are processed by `1D_Transform` reduces from $2s^2$ to $1.5s^2$: a factor of 25%. This is illustrated in Figure 12. Note that this modification generates the same multiresolution data sets as the original algorithm. Also, the effect of null wavelet coefficients as discussed in Section 3.2 remains the same. The following pseudocode describes the modified algorithm in two dimensions:

```
for each row r
    1D_Transform(r)
endfor
for each alternate column c
    1D_Transform(c)
endfor
```

Note that the computation savings are even greater when going up to three dimensions. `1D_Transform` is applied to only one-fourth of the rows in the third dimension. Consequently, the number of data elements processed by `1D_Transform`, reduces from $3s^3$ to $1.75s^3$, a factor of 41.7%.

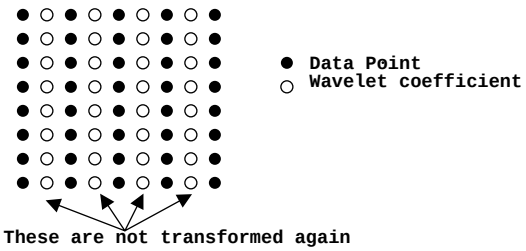

Figure 12: The data set after applying `1D_Transform` to all the rows. The columns composed of only wavelet coefficients need not be further processed by `1D_Transform`.

Also note that the multidimensional wavelet transform as described in the above pseudocode is easily parallelizable, since each iteration of the loop can be independently executed.

Region of Interest

The user can specify a rectangular region defining a region of interest as well as the resolution level of the two regions. Assume l and h are the user-specified low- and high-resolution levels, respectively, that is, the number of levels these regions are subsampled (note that $\lfloor \log_2 s \rfloor \geq l \geq h$). The initialization steps are as follows:

1. Perform l iterations of the forward wavelet transform on the full data set to bring it down to the user-specified low-resolution level.
2. Store wavelet coefficients in the cushion region, and then nullify them.
3. Perform $(l - h)$ iterations of the reverse wavelet transform only on the high-resolution region. Modify data points at the center of boundary faces after each iteration, as described in Section 3.2.
4. Perform isosurface extraction. The different resolution regions are processed by separate calls to the synchronized templates function.

The resolution levels can be changed by the user too. Assuming l' and h' are the new low and high resolutions, respectively, the following steps are performed:

1. Perform $(l - h)$ iterations of the forward wavelet transform on the old high-resolution region, restoring data values at the center of boundary faces before each iteration, as described in Section 3.2.
2. Restore the wavelet coefficients in the old cushion.
3. **if** $(l' > l)$
   Perform $(l' - l)$ iterations of the forward wavelet transform on the full data set
   else if $(l' < l)$
   Perform $(l - l')$ iterations of the reverse wavelet transform on the full data set
   This step brings the entire data set to the new low-resolution l'.
4. Store the wavelet coefficients in the new cushion and then nullify them.
5. Perform $(l' - h')$ iterations of the reverse wavelet transform only on the high-resolution region. Modify data points at the center of boundary faces after each iteration, as described in Section 3.2.
6. Perform isosurface extraction. The different-resolution regions are processed by separate calls to the synchronized templates algorithm.

The region of interest can be interactively moved and resized. The steps involved for this are same as above, skipping step 3 and replacing l' by l and h' by h in step 5. We recognize that for step 5 in this situation, the isosurface extraction needs to be done only in a limited number of voxels. If H and H' are the old and new high-resolution regions and C and C' are the old and new cushions, isosurface extraction need be done only in the region $(H - H') \cup (H' - H) \cup C \cup C'$. We haven't implemented this feature, but we foresee a reduction in the isosurface extraction times in Table 3.

5 Results

In this section we give timings and output images of our implementation on different machines. Timings are taken on a two-processor SGI octane^{TM} with an MXI graphics board and 256 MB RAM(SG), a two-processor Intel PIII^{TM} with a 32 MB Matrox Millennium 400 video card and 512 MB RAM, running Windows NT^{TM} (NT) and a two-processor Intel PIII^{TM} with a 16 MB Matrox Millennium 400 video card and 256 MB RAM, running Red Hat Linux^{TM} (LX). We haven't parallelized our implementation of the wavelet transform at this stage, and the code is not optimized.

We show results from the Rayleigh-Taylor data set,[4] which is a $128 \times 512 \times 128$ regular grid. Images at different resolutions are shown in Figure 13 and timings listed in Table 2. We observe that the wavelet transform time increases only logarithmically with number of levels. This is as expected because the number of samples to be processed reduces by one-eighth after each iteration. Note that the wavelet transform processes the full high-resolution data and therefore takes several seconds at startup, as shown in Table 2. However, this can be considered a preprocessing step. A change in the regions-of-interest box requires a multiresolution transform only of the high-resolution region. These timings are shown in Table 3.

[4]The Rayleigh-Taylor data used in this work was in part generated by the DOE-supported ASCI/Alliance Center for Astrophysical Thermonuclear Flashes at the University of Chicago.

Table 2: Rayleigh-Taylor data set at different resolutions. The wavelet transform times indicated are those needed to transform the full-resolution data set to the low resolution specified, and is required only at startup.

Res.	Polygon Count	Wavelet Transform Time (sec)			Isosurface Extraction time (sec)			Frame Rendering Time (sec)		
		SG	NT	LX	SG	NT	LX	SG	NT	LX
full	307482	N/A	N/A	N/A	15.1	7.53	4.00	0.52	1.80	1.89
half	76714	6.28	11.6	2.28	1.97	0.95	0.47	0.13	0.49	0.58
quarter	18234	7.13	13.0	2.61	0.30	0.16	0.07	.037	0.14	0.19
eighth	3612	7.21	13.2	2.66	0.06	0.03	0.01	.015	.063	.071

We now show images with a region of interest specified. In Figure 14, the region in the red box is the user-specified high-resolution region. This box can be interactively moved and resized by the user. The levels of the two resolutions can also be interactively changed.

Refer to Table 3. Every time the region of interest or the resolution level(s) are modified, wavelet transforms (which includes modification of the wavelet coefficients) as well as isosurface extraction are performed. These steps take only a few seconds even with a single-threaded implementation. On the other hand, we get big wins on frame rates. Note that the isosurfaces generated are seamless between the two resolutions. Figure 15 shows a closeup of multiresolution boundaries.

Table 3: Rayleigh-Taylor data set with regions of interest. The above wavelet transform and isosurface extraction times are those that occur when the region of interest box is moved or resized. Images of the region of interest box are shown in Figure 14.

High : Low	Polygon Count	Wavelet Transform Time (sec)			Isosurface Extraction Time (sec)			Frame Rendering Time (sec)		
		SG	NT	LX	SG	NT	LX	SG	NT	LX
0:1 14(a)	98896	0.23	0.77	0.13	2.26	1.20	0.58	0.18	0.63	0.98
0:2 14(b)	49986	0.29	1.06	0.17	0.66	0.52	0.19	0.09	0.32	0.50
0:3 14(c)	43766	0.56	1.53	0.22	0.46	0.49	0.15	0.08	0.27	0.44

6 Conclusions and Future Work

We have demonstrated an efficient method for displaying seamless multiresolution isosurfaces interactively. The method makes use of the power of wavelet theory to generate multiple resolutions of the data while introducing nearly minimal error, thus allowing exploratory scientific visualization on a wide variety of machines. The multiresolution transform we use guarantees that the multiresolution scalar field generated is C^0 continuous.

Future work includes an inspection of wavelet coefficients at different regions of the data set, using them as a measure of the resolution needed to faithfully represent the data and thus providing a framework for generation of adaptive isosurfaces. Regions of interest may also be defined based on the user gaze within a environment

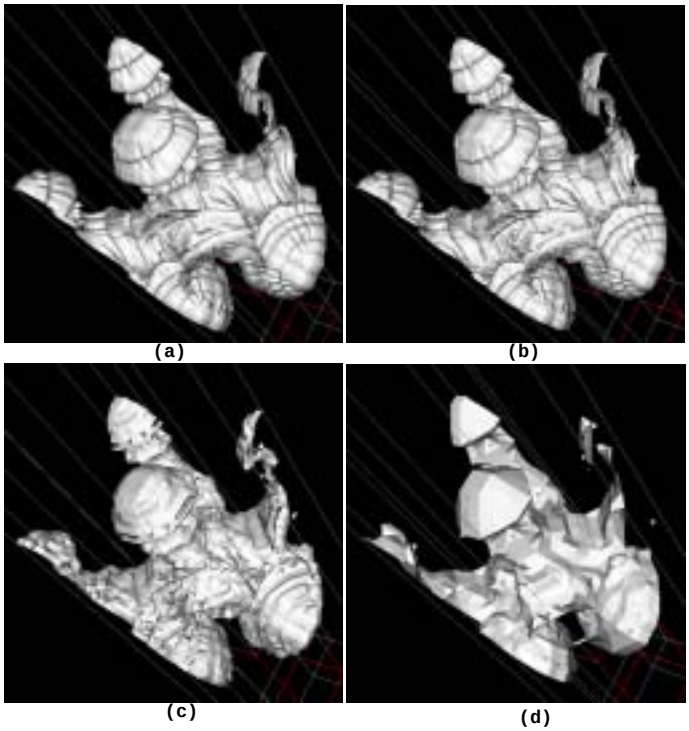

Figure 13: Isosurface of Rayleigh-Taylor data at multiple resolutions: (a) full resolution; (b) one-level subsampling (87.5% reduction); (c) Two level subsampling (98.44% reduction); (d) three level subsampling (99.80% reduction).

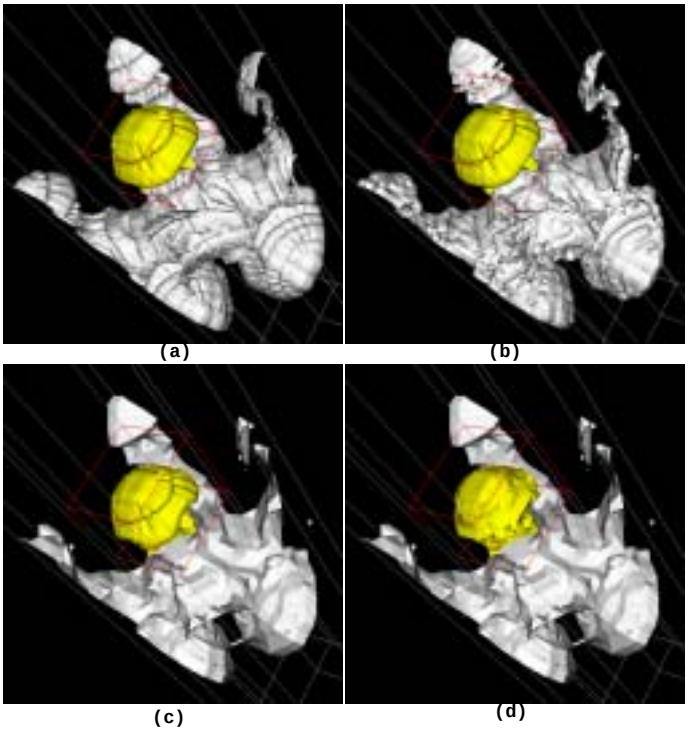

Figure 14: Isosurface of Rayleigh-Taylor data with region of interest specified as a red box: (a) high:full resolution, low:half resolution; (b) high:full resolution, low:quarter resolution; (c) high:full resolution, low:one-eighth resolution (d) high:half resolution, low:quarter resolution.

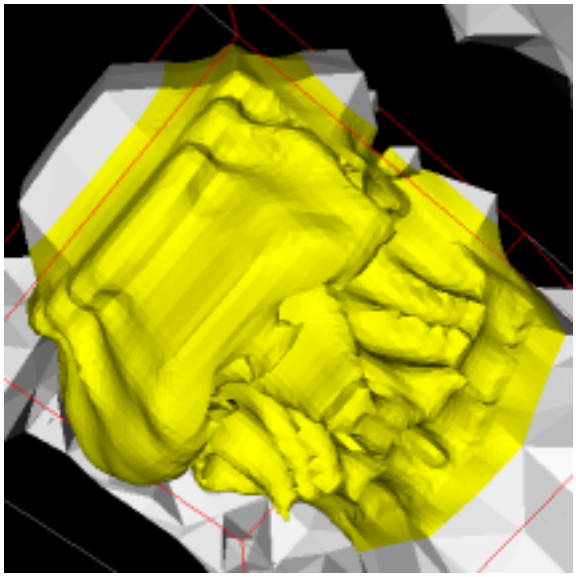

Figure 15: Seamless multi resolution boundaries.

where such information is already being collected, such as in a virtual environment. In the area of remote visualization, we plan to investigate progressive transmission of multiresolution isosurfaces using wavelet-based multiresolution decomposition.

Acknowledgments

We thank Wim Sweldens of Bell Laboratories and Charles Law and Will Schroeder of Kitware for answering some of our queries. We thank Rick Stevens and Gail Pieper for reading and commenting on the paper. We thank the Futures Laboratory staff at Argonne National Laboratory for their help and support. This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

References

[1] DAUBECHIES, I. *Ten Lectures on Wavelets*. SIAM, Philadelphia, 1992.

[2] DREBIN, R. A., CARPENTER, L., AND HANRAHAN, P. Volume rendering. In *Computer Graphics (SIGGRAPH '88 Proceedings)* (Aug. 1988), J. Dill, Ed., vol. 22, pp. 65–74.

[3] FERNÁNDEZ, G., PERIASWAMY, S., AND SWELDENS, W. LIFTPACK: A software package for wavelet transforms using lifting. In *Wavelet Applications in Signal and Image Processing IV* (1996), M. Unser, A. Aldroubi, and A. F. Laine, Eds., Proc. SPIE 2825, pp. 396–408.

[4] GROSS, M. H., STAADT, O. G., AND GATTI, R. Efficient triangular surface approximations using wavelets and quadtree data structures. *IEEE Transactions on Visualization and Computer Graphics 2*, 2 (June 1996), 130–143. ISBN 1077-2626.

[5] HAAR, A. *Zur theorie der orthogonalen funktionensysteme*. Mathematische Annalen, LXIX, 1910.

[6] IHM, I., AND PARK, S. Wavelet-based 3d compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum 18*, 1 (March 1999), 3–15. ISSN 1067-7055.

[7] LAW, C. C., MARTIN, K. M., SCHROEDER, W. J., AND TEMKIN, J. A multi-threaded streaming pipeline architecture of large structured data sets. In *Proceedings of Visualization '99* (October 1999), D. Ebert, M. Gross, and B. Hamann, Eds., IEEE Computer Society and ACM, ACM Press, pp. 225–232.

[8] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (July 1987), M. C. Stone, Ed., vol. 21, pp. 163–169.

[9] MALLAT, S. A theory of multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11* (1989), 674–693.

[10] MONTANI, C., SCATENI, R., AND SCOPIGNO, R. Discretized marching cubes. *IEEE Visualization '94* (October 1994), 281–287. ISBN 0-8186-6627-7.

[11] MULLER, H., AND STARK, M. Adaptive generation of surfaces in volume data. *The Visual Computer 9*, 4 (Jan. 1993), 182–199.

[12] MURAKI, S. Volume data and wavelet transform. *IEEE Computer Graphics and Applications 13*, 4 (July 1993), 50–56.

[13] NIELSON, G. M., AND HAMANN, B. The asymptotic decider: Removing the ambiguity in marching cubes. In *Visualization '91* (1991), pp. 83–91.

[14] NING, P., AND BLOOMENTHAL, J. An evaluation of implicit surface tilers. *IEEE Computer Graphics and Applications 13*, 6 (1993), 33–41.

[15] PAPKA, M. E. Extending genetic programming for discrete volume visualization. Master's thesis, University of Illinois at Chicago, 1994.

[16] RODLER, F. Wavelet-based 3d compression with fast random access for very large volume data. *Pacific Graphics '99* (October 1999). Held in Seoul, Korea.

[17] ROY, T. M., CRUZ-NIERA, C., AND DEFANTI, T. A. Cosmic worm in the cave: Steering a high-performance computing application from a virtual environment. *Presence: Teleoperators and Virtual Environments 4*, 2 (Spring 1995), 121–131.

[18] SCHROEDER, W., MARTIN, K., AND LORENSEN, B. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Prentice Hall, 1995.

[19] SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. Decimation of triangle meshes. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 65–70.

[20] SHEKHAR, R., FAYYAD, E., YAGEL, R., AND CORNHILL, J. F. Octree-based decimation of marching cubes surfaces. In *IEEE Visualization '96* (Oct. 1996), IEEE. ISBN 0-89791-864-9.

[21] SHU, R., ZHOU, C., AND KANKANHALLI, M. S. Adaptive marching cubes. *The Visual Computer 11*, 4 (1995), 202–217. ISSN 0178-2789.

[22] STOLLNITZ, E., DEROSE, T., AND SALESIN, D. *Wavelets for Computer Graphics*. Morgan Kaufmann, San Francisco, CA, 1996.

[23] SWELDENS, W. The lifting scheme: A new philosophy in biorthogonal wavelet constructions. In *Wavelet Applications in Signal and Image Processing III* (1995), A. F. Laine and M. Unser, Eds., Proc. SPIE 2569, pp. 68–79.

[24] WESTERMANN, R. A multiresolution framework for volume rendering. In *1994 Symposium on Volume Visualization* (Oct. 1994), A. Kaufman and W. Krueger, Eds., ACM SIGGRAPH, pp. 51–58. ISBN 0-89791-741-3.

[25] WESTERMANN, R., KOBBELT, L., AND ERTL, T. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer 15*, 2 (1999), 100–111. ISSN 0178-2789.

[26] WILHELMS, J., AND GELDER, A. V. Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 3 (July 1992), 201–227.