Fast Multiresolution Extraction of Multiple Transparent Isosurfaces

Thomas Gerstner

Department for Applied Mathematics, University of Bonn
Wegelerstr. 6, 53115 Bonn, Germany
gerstner@iam.uni-bonn.de

Abstract. In this paper, we present a multiresolution algorithm which is capable to render multiple transparent isosurfaces under real–time constraints. To this end, the underlying 3D data set is covered with a hierarchical tetrahedral grid. The multiresolution extraction algorithm is then based on an adaptive traversal of the tetrahedral grid with the help of error indicators. The display of transparent isosurfaces using alpha blending requires a back–to–front rendering of the isosurface triangles. This is achieved by a hierarchical sorting procedure of the tetrahedra and the hierarchical computation of data gradients. We will also comment on the automated selection of suitable isovalues for visualization applications.

1 Introduction

Interactive rendering of large volumetric data sets is a hard task. Besides direct volume rendering methods, such as ray casting, splatting, or 3D texture mapping, indirect volume rendering techniques, such as isosurface extraction, are frequently applied. In both settings, hardware acceleration and multiresolution techniques are often required in order to achieve real–time visualization performance.

Isosurface extraction algorithms rely on the ability of current graphics processors to render large amounts of triangles very quickly. Still, the total rendering time is often dominated by the extraction time of the isosurface, that is the computation of the isosurface triangle vertices. Here, multiresolution methods allow significant reductions of the number of isosurface triangles through suitable approximations of the volume, thereby speeding up both extraction and rendering time.

The display of multiple isosurfaces can be used as a surrogate for direct volume rendering techniques, especially when spiky transfer functions are used. Thereby the selection of isovalues can be done statically, automatically adapted to the data set, or defined by the user. Since the number of displayed isosurfaces will not be very large in interactive applications, the huge number of degrees of freedom in transfer function design is also drastically reduced. Isosurfaces with different isovalues are completely nested and therefore the inner isosurfaces are completely obscured by the outer isosurface except at the boundary of the data set. Thus, multiple isosurfaces have to be rendered transparently which is usually also supported by the graphics hardware. However, then the triangles have to be processed in a strict back–to–front fashion. This requires a view–dependent sorting of all the isosurface triangles. Once the user changes the viewpoint, the sorting time will dominate the total rendering time.

The goal of this paper is to show that in volumetric multiresolution methods this sorting step to be done hierarchically in constant time. Thereby we will focus on a specific well–known multiresolution method based on recursive bisection of tetrahedra. The hierarchical sorting is done in three phases: sorting of the initial tetrahedra, recursive sorting of child tetrahedra during the tree traversal, and sorting of the isosurface components inside each tetrahedron. View–dependent sorting also requires the computation of data gradients inside each tetrahedron. Although these gradients can be precomputed, they require large amounts of memory. We will therefore show how gradients can quickly be computed hierarchically on–the–fly. Finally, we will comment on the automated selection of suitable isovalues.

This paper is organized as follows. Section 2 reviews related work. Section 3 shortly discusses the construction of multiresolution isosurfaces based on tetrahedral bisection. Sorting is done in Section 4. Section 5 explains hierarchical gradient computation. Visualization examples are shown in Section 6. Section 7 describes a technique for automated isovalue selection. The final remarks of Section 8 conclude the paper.

2 Related Work

Multiresolution techniques have been successfully applied to the four most popular direct volume rendering algorithms such as 3D texture mapping [11, 27], ray casting [4, 18, 28], splatting [9, 12, 13], and the shear–warp transformation [3, 31]. For a detailed (non–multiresolution) comparison of these methods see [17].

Isosurface extraction can be very slow when marching algorithms [15, 21] which scan the complete data set are used. Therefore, a variety of methods have been designed which try to avoid to search through regions where no intersection with the isosurface occurs. To this end, hierarchical partitions of either the geometric [30] or the span space [14] are constructed. For a survey and comparison of available methods see [1, 26].

Multiresolution isosurface extraction methods are characterized by a hierachical decomposition of the underlying geometric space. Through suitable approximations of the data, they are also able to extract approximate isosurfaces with varying complexity. The various methods mainly differ in the type of hierarchy and interpolation, such as octrees [24], red tetrahedral refinement [8, 10, 29], tetrahedral bisection [5–7, 20, 32], hierarchical Delaunay triangulations [2], or wavelet techniques [25]. With the help of bounds for the minimum and maximum data value inside each subdomain, the scanning of empty regions is also avoided.

If the data domain is refined adaptively, it can happen that the extracted isosurface contains cracks at transition zones where the mesh resolution changes. For this problem, different solutions have been devised such as remeshing [8], point insertion [24], projection [19], blending [10], and saturation [5–7, 32].

3 Multiresolution Isosurface Extraction

In this section, we will explain the construction of multiresolution isosurfaces based on tetrahedral bisection and error indicators. The algorithms have already been described in detail in previous works but for clarity we shortly repeat the basic steps here.

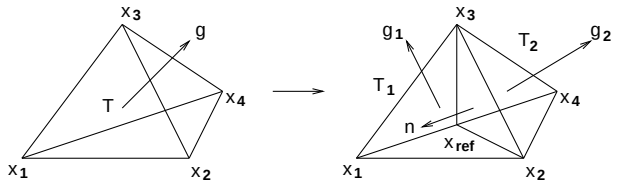

Fig. 1. Bisection of a tetrahedron T into two child tetrahedra T_1 and T_2.

Let us consider a nested hierarchy of tetrahedral grids where the tetrahedra are refined by recursive bisection [16, 23]. For a tetrahedron T the midpoint of a predetermined (in our case the longest) edge $e_{\text{ref}}(T)$ is chosen as a new vertex $x_{\text{ref}}(T)$. Then, the tetrahedron is split at the face spanned by $x_{\text{ref}}(T)$ and the two vertices of T opposite to $e_{\text{ref}}(T)$ into two child tetrahedra $T_1(T)$ and $T_2(T)$ (Figure 1). Through recursive application of the refinement rule a binary tree hierarchy is inferred on the tetrahedra.

The adaptive multiresolution isosurface algorithm is based on a depth first traversal of the binary tree. On every tetrahedron for a stopping criterion is checked. If it is true, the algorithms stops and renders the local isosurfaces using the look–up table of the marching tetrahedra algorithm [21]. Otherwise, the two children are visited recursively.

If the algorithms stops on a specific tetrahedron T and refines another tetrahedron which shares the refinement edge, an inconsistency occurs at the hanging node $x_{\text{ref}}(T)$. This leads to cracks in the isosurface. Therefore, we ensure that whenever a tetrahedron is refined, all tetrahedra sharing its refinement edge are refined as well. This can be achieved by definition of error indicators η on the refinement vertices, i.e. $\eta(T) = \eta(x_{\text{ref}}(T))$, and choosing $\eta(T) < \varepsilon$ as a stopping criterion for some user specified threshold value ε. If the error indicator values are saturated [5–7, 32], no hanging nodes can occur for all possible values of ε. This way, the extraction algorithm is completely local and information from neighboring tetrahedra is never required.

Furthermore, the traversal of the binary tree is also stopped if the tetrahedron is not a candidate for an intersection with one of the isosurfaces. In our case, it is checked whether any of the isovalues is contained in the interval consisting of all the data values inside the tetrahedron. This information can either be explicitly computed in a bottom–up traversal of the tree [30] or be obtained from already available error indicator values [7]. This way, the complexity of the extraction algorithm is of the order of the output (the number of drawn triangles), independent of the size of the input in practice.

4 Transparency Sorting

The most efficient way to display transparent surfaces is through alpha blending, which is supported by basically all manufacturers of graphics cards. Alpha blending requires a back–to–front sorting of the rendered primitives, though. In principle, the isosurface meshes could first be extracted and then the triangles be sorted, but it turns out that the sorting time then dominates the total rendering time. This is especially bad since rotation of the isosurface is the predominant user action in visualization and thus sorting has to be done for almost every frame.

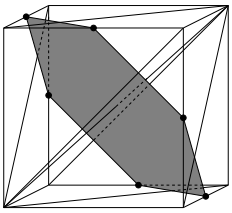

Fig. 2. Sorting points and normal plane of the initial tetrahedral mesh.

On the other hand, multiresolution methods allow a back–to–front sorting during the extraction phase which eliminates above problem. We will now show how this sorting can be done at virtually no extra cost for recursive bisection tetrahedral meshes. Thereby three different sorting problems arise: sorting of the initial tetrahedra, recursive sorting of the child tetrahedra during the adaptive tree traversal, and sorting of the isosurface components inside a tetrahedron during extraction. Let us emphasize here that only those tetrahedra that are visited during the adaptive tree traversal are sorted.

4.1 Initial Tetrahedra

We assume that the input volume data is arranged in a uniform grid with n^3 nodes, $n = 2^k + 1$. The initial tetrahedral mesh consists of the six tetrahedra whose vertices are adjacent corners of the cube and which all share the same diagonal of that cube (Figure 2). Applying then the refinement scheme of the previous section, all refinement vertices $x_{\text{ref}}(T)$ will fall onto grid points of the original data set.

These tetrahedra have to be sorted starting with the most distant tetrahedron and ending with the closest one. Let v be the viewing vector (from the eye to the object) and n be the normal of a separating plane of a pair of non–intersecting tetrahedra T_i and T_j. Let us assume that T_i lies in direction of the normal n and T_j is in opposite direction. Then, T_i is behind T_j if $v \cdot n > 0$ and before T_j if $v \cdot n < 0$. If $v \cdot n = 0$ those tetrahedra could be processed in parallel.

For the six initial tetrahedra, 15 normals of separating planes between any two tetrahedra are possible. These normals can be computed as the vector between two distinguished points each located on the boundary of one tetrahedron. These six points are the midpoints of those edges whose endpoints are not endpoints of the diagonal of the cube thereby spanning a normal plane (as shown in Figure 2). With this information, any sorting algorithm such as quicksort can immediately be applied to sort the tetrahedra.

4.2 Child Tetrahedra

During the tree traversal, a tetrahedron is split into two child tetrahedra. The separating plane between those tetrahedra is spanned by the points x_2, x_3 and x_{ref} (Figure 1). Let the normal n of the separating plane point towards T_1. Then, similarly to the previous section, T_1 has to be processed before T_2 if $v \cdot n > 0$ and vice versa otherwise.

The normal n could be computed by $\overline{x_2 x_{\text{ref}}} \times \overline{x_3 x_{\text{ref}}}$ on–the–fly, but it is more efficient to precompute and store this information for each type of reference tetrahedron. In our

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
N_1	36	43	42	47	48	26	44	35	33	46	45	37	25	27	29	31	33	35	34	32	39	41	42	43
N_2	26	28	30	32	34	36	37	38	40	27	25	44	45	46	31	29	40	38	48	47	41	39	30	28

N	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
N_1	49	51	53	55	57	59	61	63	65	67	69	71	52	60	62	56	58	66	70	72	68	64	50	54
N_2	50	52	54	56	58	60	62	64	66	68	70	72	71	55	57	59	61	69	65	51	63	67	53	49

N	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72
N_1	11	16	18	7	2	19	2	9	16	21	23	18	4	9	4	14	21	11	7	23	6	14	6	19
N_2	17	1	12	1	20	10	8	24	22	3	17	3	10	15	13	8	12	5	22	5	15	20	24	13

Table 1. Mapping table for the reference tetrahedra numbers.

case, there are 72 reference tetrahedra. Of the three basic types of tetrahedra which cycle all three refinement levels (see [5]) there are 24 instances from the respective rotation and mirror symmetry classes. For example, the six tetrahedra of the first type all share the same diagonal of the cube and there are four different diagonals possible.

Not surprisingly, the numbers of the reference tetrahedra can be determined hierarchically. So given the reference number N of a tetrahedron, the reference numbers of the two child tetrahedra N_1 and N_2 need to be determined. It turns out that this mapping appears to be quite erratic and no simple formula can be given for $N_1(N)$ and $N_2(N)$ (or, at least, we found none). Since the complete mapping is fairly difficult to obtain due to its cyclic structure, we state it in table 1. The six initial tetrahedra are numbered from 1 to 6.

4.3 Isosurface Components inside a Tetrahedron

Now that we've finally sorted all the tetrahedra in the adaptive tetrahedral mesh the only thing left is to sort the isosurface components within each tetrahedron. This case rarely arises for fine resolution datasets and large differences in between isovalues. But our multiresolution algorithm tries to extract and render isosurfaces on coarse tetrahedra and thereby often several isosurfaces will intersect a given tetrahedron making this sorting necessary.

Let $\{i_1, \ldots, i_m\}$ the ordered set of isovalues starting with i_1 being the smallest and i_m being the largest. Let us assume that for the ordered subset $\{i_j, \ldots, i_k\}$ of all isovalues the corresponding isosurfaces intersect the current tetrahedron. Then, the isosurfaces have to be rendered either starting with the lowest isovalue i_j and ending with the highest isovalue i_k, or starting with i_k and ending with i_j. Since linear interpolation is used inside each tetrahedron, all isosurface components are parallel to each other. The order of the isosurfaces is therefore determined by the inner product of the normal of the isosurfaces with the viewing vector. The normal of the isosurface triangles is just the gradient g of the linear function spanned by the data values at the vertices of the tetrahedron. We have therefore the sorting test: if $v \cdot g > 0$, then the highest isovalue has to be processed first, if $v \cdot g < 0$, the lowest one. Note that for $v \cdot g = 0$ the isosurface triangles are parallel to the viewing vector and therefore nearly invisible. How these gradients can be computed efficiently will be shown in the next section.

5 Hierarchical Gradient Computation

As we have seen in the previous section, transparency sorting requires the data gradients. These gradients can in principle be precomputed but they will then require a lot of memory. Since the number of tetrahedra in the mesh is about six times the number of vertices, the required memory would be roughly $6 \cdot 3n^3$ floating point numbers in addition to the n^3 data values. It is therefore advisable to compute these gradients on–the–fly, but in straightforward implementation, gradient computation would significantly decrease the performance of the multiresolution algorithm. We will now show how gradients can be computed very efficiently hierarchically.

Let us recall that the gradient g on the tetrahedron T for a linear function $f(x, y, z) = ax + by + cz + d$ is given by $g = \nabla f = (a\ b\ c)^T$. The coefficients a, b, c can be computed for data values $f_1, \ldots, f_4$ at the respective vertices $(x_1, y_1, z_1), \ldots, (x_4, y_4, z_4)$ by the solution of the linear system

$$\begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}.$$

Given the solution of this system, we now want to compute the gradients g_1, g_2 of the two child tetrahedra. Let us first look at the first child T_1 (see Figure 1). The coefficients a_1, b_1, c_1 of g_1 are given by the following system

$$\begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_{\text{ref}} & y_{\text{ref}} & z_{\text{ref}} & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_{\text{ref}} \end{pmatrix},$$

where f_{ref} is the data value at the refinement vertex. Let us now rewrite the first system through replacement of the fourth row with the sum of the first and fourth rows divided by two,

$$\begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ \frac{x_1+x_4}{2} & \frac{y_1+y_4}{2} & \frac{z_1+z_4}{2} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \frac{f_1+f_4}{2} \end{pmatrix}.$$

Since $x_{\text{ref}} = (x_1 + x_4)/2$ (and similarly for y and z), we see that the system for the child differs from the system for the parent only by the fourth component of the right hand side. Let the inverse of above matrix be given by (w_{ij}). Then we have for a

$$a = w_{11}f_1 + w_{12}f_2 + w_{13}f_3 + w_{14}(f_1 + f_4)/2$$

and for a_1

$$a_1 = w_{11}f_1 + w_{12}f_2 + w_{13}f_3 + w_{14}f_{\text{ref}}.$$

Differencing yields

$$a_1 = a + w_{14}(f_{\text{ref}} - (f_1 + f_4)/2).$$

Repeating this step for b and c we end up with

$$\begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} + (f_{\text{ref}} - (f_1 + f_4)/2) \begin{pmatrix} w_{14} \\ w_{24} \\ w_{34} \end{pmatrix}.$$

Completely analogously, the gradient g_2 of T_2 can be computed from g by replacement of w_{i4} with w_{i1}. Now, only the corresponding w_{ij} for all 72 reference tetrahedra have to be computed and stored in advance. The numbers of the reference tetrahedra can be determined during the traversal by the numbering scheme of the previous section. In comparison to direct computation (matrix inversion) or the plain usage of reference elements (matrix–vector multiplication), the hierarchical method requires just scalar multiplication and vector addition. Note that the factor $f_{\text{ref}} - (f_1 + f_4)/2$ is exactly the wavelet coefficient in the piecewise linear lazy wavelet representation. Of course the gradients can be reused for flat illumination shading of the isosurface triangles, although we use Goraud shading in our examples.

6 Visualization Examples

As a first example serves the well–known buckyball data set (courtesy of AVS). Figure 5 shows three isosurfaces for isovalues of 0.05, 0.15 and 0.25. The colors of the isosurfaces are blue, red and green (from outer to inner) with opacities of 0.3, 0.5, and 0.7. The error thresholds ε of the six images are 0.0, 0.01, 0.02, 0.04, 0.08, and 0.16. The number of triangles are 1775762, 979730, 438042, 277698, 171465, and 92309.

The second example in Figure 6 shows the electron density around the cap of a nanotube (courtesy of A. Caglar, Univ. of Bonn). In addition to the isosurfaces, the different atoms of the molecule are shown (hydrogen in blue, boron in red, and nitrite in green). The isovalues are 0.05 (yellow), 0.2 (green), and 0.35 (blue) with opacities of 0.3. The ε–values are identical to the buckyball example resulting in 230452, 175054, 143126, 96327, 55933, and 42630 triangles. The atoms are rendered as small textured balls and are inserted during the tree traversal at the appropriate position. In this example, it was not necessary to split the textures across tetrahedra since they were small enough and placed in the centers of the tetrahedra. In other cases, it may be necessary to split such textures, though.

The combined rendering and extraction time of the algorithm is about 270000 triangles/sec on an SGI Onyx^2 (R10000, 195 MHz). The same algorithm for opaque isosurfaces achieves about 300000 triangles/sec, so the time required for sorting is moderate (about 10%) and does not degrade the total performance significantly.

7 Isovalue Selection

In many applications there is a close corresponence between data values and useful isovalues. For example, in medical imaging, bone, tissue and blood vessels have certain known reflectivities returned from medical scanners. In many other applications this correspondence may be unknown or changing in between data sets. Certainly, it is often

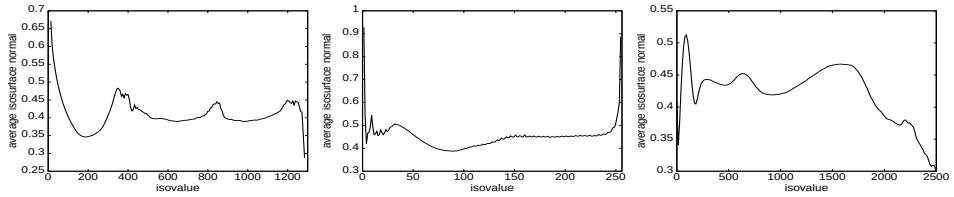

Fig. 3. Average isosurface normal sizes $\bar{g}$ for the tooth, sheep and knee data sets.

the best way to determine suitable isovalues by trial and error. However, in some cases, it may also be helpful to offer the user first guesses of suitable isosurfaces and let him or her decide whose are useful and whose not.

Probably the most straightforward way to determine suitable isovalues is by looking at the gradient field of the data set. Let us define the (discrete) average size of the normals of an isosurface as

$$\bar{g}(i) = \frac{\sum_{T_j : s(i) \cap T_j \neq \emptyset} area(s(i) \cap T_j) \cdot |g(T_j)|}{\sum_{T_j : s(i) \cap T_j \neq \emptyset} area(s(i) \cap T_j)}$$

where i is the isovalue, $s(i)$ the triangulated isosurface, and $g(T_j)$ the data gradient on T_j (which is the normal of the isosurface). The local minima and maxima of $\bar{g}(i)$ then characterize possible isovalues. The maxima will separate homogeneous areas and the minima are the centers of these areas.

Let us take a look at some concrete graphs $\bar{g}(i)$ for the three different data sets (courtesy of B. Lorensen, General Electric) which have been used for the transfer function bake–off at Visualization 2000 [22]. Those data sets, CT respectively MRI scans of a tooth, a sheep's heart, and a knee serve as benchmarks in transfer function design. The computed $\bar{g}(i)$ for the three data sets are shown in Figure 3.

In all cases, we found the minima more useful. For the tooth data set there are three clearly visible minima. The leftmost minimum corresponds to the cylindrical outer shell of the medium in which the tooth was set prior to scanning. The other two minima give the surface and the enamel of the tooth as shown in Figure 4, left. The sheep data set shows only one distinguished minimum indicating the heart's surface and inner blood vessels (Figure 4 middle). Manual scanning of other isovalues revealed no further useful isosurfaces. The knee data sets shows three clear minima despite the high amount of noise in the data corresponding to skin, muscular tissue, and bone (Figure 4 right). Although we did not do so, such noisy data should be smoothed before isosurfacing.

8 Concluding Remarks

In this paper, we have shown how multiple transparent isosurfaces can be extracted interactively using the tetrahedral bisection hierarchy. This was achieved by an adaptive tree traversal, a hierarchical sorting procedure of the tetrahedra and isosurface triangles,

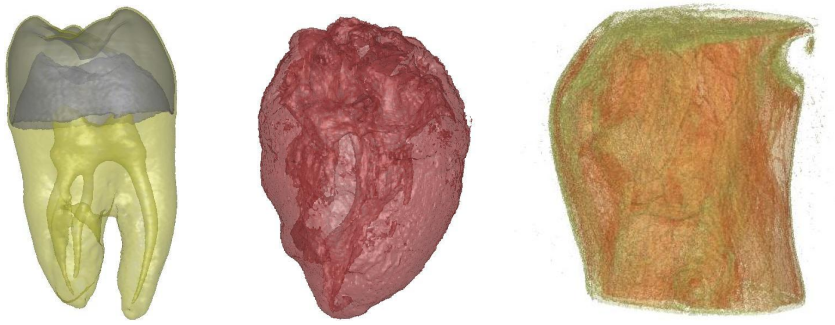

Fig. 4. The tooth, sheep and knee data sets rendered with multiple transparent isosurfaces.

and the hierarchical computation of data gradients. Furthermore, we have shown how isovalues can be selected (semi-)automatically based on the local minima of the average isosurface normal graphs.

Let us remark that in comparison to direct volume rendering methods isosurface extraction requires no special purpose hardware and gives images with sharp boundaries. Also, besides the isovalues, colors and opacities, which can be obtained quickly, no further design parameters are necessary. On the downside, details in between isosurfaces are lost and cannot be displayed with this methodology. Of course, the sorting and numbering algorithms for the tetrahedra can be used in direct volume rendering algorithms based on tetrahedral splats or ray casting (with inverted sorting order).

References

1. C. Bajaj, V. Pascucci, and D. Schikore. Accelerated Isocontouring of Scalar Fields. In C. Bajaj, editor, *Data Visualization Techniques*. John Wiley and Sons, 1998.
2. P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution Representation and Visualization of Volume Data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369, 1997.
3. F. Dong, M. Krokos, and G. Clapworthy. Fast Volume Rendering and Data Classification using Multiresolution Min–Max Octrees. *Computer Graphics Forum*, 19(3):359–367, 2000.
4. T. Ertl, R. Westermann, and R. Grosso. Multiresolution and Hierarchical Methods for the Visualization of Volume Data. *Future Generation Computer Systems*, 15(1):31–42, 1999.
5. T. Gerstner and R. Pajarola. Topology Preserving and Controlled Topology Simplifying Multiresolution Isosurface Extraction. In *Proc. IEEE Visualization 2000*, pages 259–266. IEEE Computer Society Press, 2000.
6. T. Gerstner and M. Rumpf. Multiresolutional Parallel Isosurface Extraction based on Tetrahedral Bisection. In M. Chen, A. Kaufman, and R. Yagel, editors, *Volume Graphics*, pages 267–278. Springer, 2000.
7. T. Gerstner, M. Rumpf, and U. Weikard. Error Indicators for Multilevel Visualization and Computing on Nested Grids. *Computers & Graphics*, 24(3):363–373, 2000.
8. R. Grosso, C. Lürig, and T. Ertl. The Multilevel Finite Element Method for Adaptive Mesh Optimization and Visualization of Volume Data. In *Proc. IEEE Visualization '97*, pages 387–394. IEEE Computer Society Press, 1997.

9. B. Guo. A Multiscale Model for Structure–based Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):291–301, 1995.
10. D. Holliday and G. Nielson. Progressive Volume Model for Rectilinear Data using Tetrahedral Coons Volumes. In W. de Leeuw and R. van Liere, editors, *Data Visualization 2000*, pages 83–92. Springer, 2000.
11. E. LaMar, B. Hamann, and K. Joy. Multiresolution Techniques for Interactive Texture–based Volume Visualization. In *Proc. IEEE Visualization '99*, pages 355–362. IEEE Press, 1999.
12. D. Laur and P. Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. *Computer Graphics (SIGGRAPH '91 Proc.)*, pages 285–288, 1991.
13. L. Lippert and M. Gross. Fast Wavelet based Volume Rendering by Accumulation of Transparent Texture Maps. *Computer Graphics Forum*, 14(3):431–444, 1995.
14. Y. Livnat, H. Shen, and C. Johnson. A Near Optimal Isosurface Extraction Algorithm using the Span Space. *IEEE Trans. on Visualization and Computer Graphics*, 2(1):73–83, 1996.
15. W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4):163–169, 1987.
16. J. Maubach. Local Bisection Refinement for n-simplicial Grids generated by Reflection. *SIAM J. Sci. Comp.*, 16:210–227, 1995.
17. M. Meissner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A Practical Evaluation of Popular Volume Rendering Algorithms. In *Proc. Volume Visualization 2000*, pages 81–91. ACM Press, 2000.
18. S. Muraki. Approximation and Rendering of Volume Data using Wavelet Transforms. *Computer Graphics and Applications*, 13(4):50–56, 1993.
19. M. Ohlberger and M. Rumpf. Adaptive Projection Methods in Multiresolutional Scientific Visualization. *IEEE Trans. on Visualization and Computer Graphics*, 4(4):74–94, 1998.
20. V. Pascucci and C. Bajaj. Time Critical Isosurface Refinement and Smoothing. In *Proc. Volume Visualization 2000*, pages 33–42. ACM Press, 2000.
21. B. Payne and A. Toga. Surface Mapping Brain Function on 3D Models. *IEEE Computer Graphics and Applications*, 10(5):33–41, 1990.
22. H. Pfister (org.), B. Lorensen, C. Bajaj, G. Kindlmann, and W. Schroeder. The Transfer Function Bake–Off. Panel session at IEEE Visualization '00, 2000.
23. M. Rivara and C. Levin. A 3D Refinement Algorithm suitable for Adaptive and Multi-Grid Techniques. *Comm. Appl. Num. Meth.*, 8:281–290, 1992.
24. R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree–based Decimation of Marching Cubes Surfaces. In *Proc. IEEE Visualization '96*, pages 335–344. IEEE Press, 1996.
25. O. Staadt, M. Gross, and R. Weber. Multiresolution Compression and Reconstruction. In *Proc. IEEE Visualization '97*, pages 337–364. IEEE Computer Society Press, 1997.
26. P. Sutton, C. Hansen, H.-W. Shen, and D. Schikore. A Case Study of Isosurface Extraction Algorithm Performance. In W. de Leeuw and R. van Liere, editors, *Data Visualization 2000*, pages 259–268. Springer, 2000.
27. M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, and T. Ertl. Level–of–Detail Volume Rendering via 3D Textures. In *Proc. Volume Vis. 2000*, pages 7–13. ACM Press, 2000.
28. R. Westermann. A Multiresolution Framework for Volume Rendering. In *Proc. Volume Visualization 94*, pages 51–57. ACM Press, 1994.
29. R. Westermann, L. Kobbelt, and T. Ertl. Real–Time Exploration of Regular Volume Data by Adaptive Reconstruction of Isosurfaces. *The Visual Computer*, 15:100–111, 1999.
30. J. Wilhelms and A. Van Gelder. Octrees for Faster Isosurface Generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.
31. Y. Yang, F. Lin, and H. Seah. Fast Multi–Resolution Volume Rendering. In M. Chen, A. Kaufman, and R. Yagel, editors, *Volume Graphics*, pages 185–197. Springer, 2000.
32. Y. Zhou, B. Chen, and A. Kaufman. Multiresolution Tetrahedral Framework for Visualizing Volume Data. In *Proc. IEEE Visualization '97*, pages 135–142. IEEE Press, 1997.

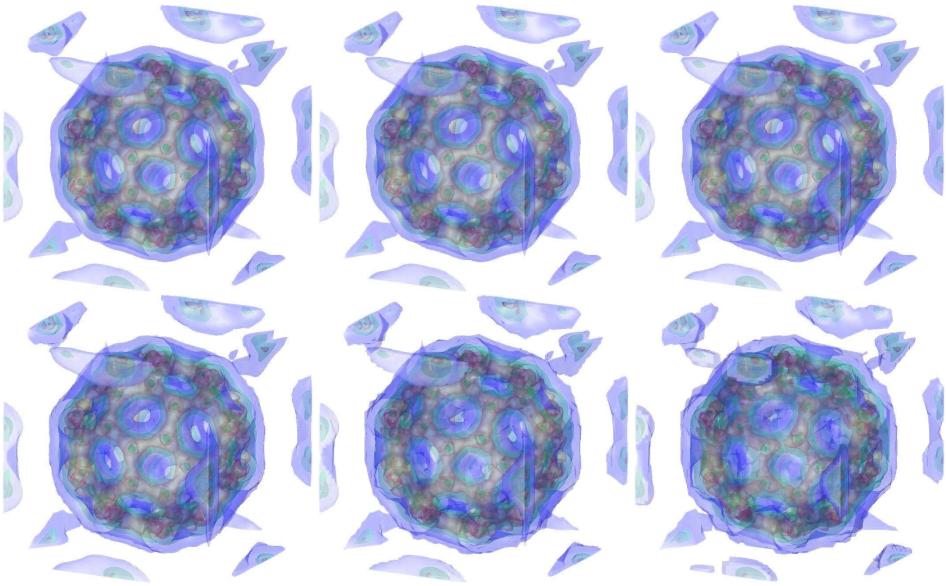

Fig. 5. Multiple transparent isosurfaces of the buckyball data set for varying error thresholds.

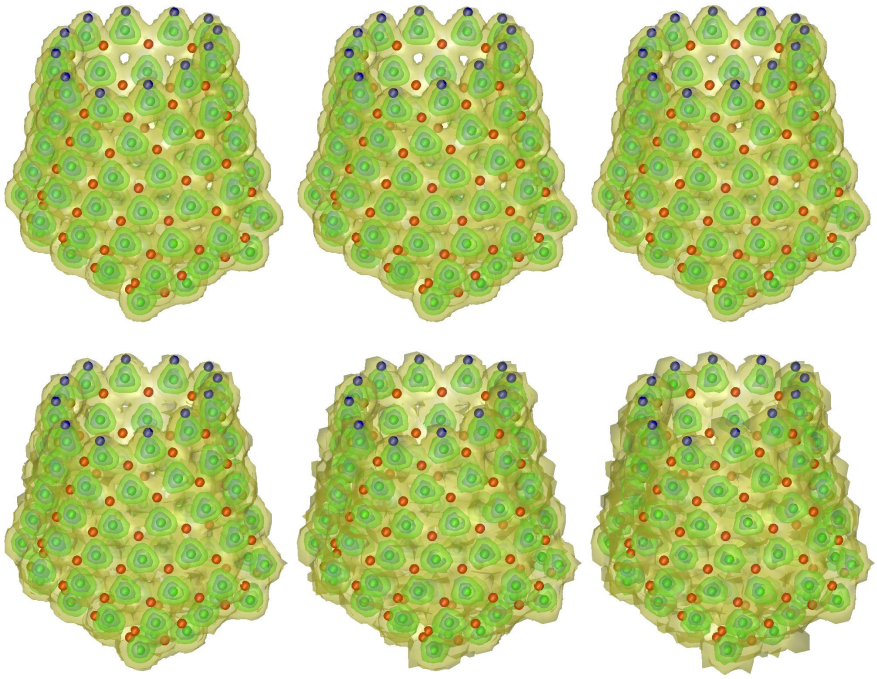

Fig. 6. Multiple transparent isosurfaces of a boron–nitrite nanotube cap with corresponding atomic positions for varying error thresholds.