Isosurfacing In Higher Dimensions

Praveen Bhaniramka[*]   Rephael Wenger[+]   Roger Crawfis[+]

ABSTRACT

Visualization algorithms have seen substantial improvements in the past several years. However, very few algorithms have been developed for directly studying data in dimensions higher than three. Most algorithms require a sampling in three-dimensions before applying any visualization algorithms. This sampling typically ignores vital features that may be present when examined in oblique cross-sections, and places an undo burden on system resources when animation through additional dimensions is desired. For time-varying data of large data sets, smooth animation is desired at interactive rates. This paper provides a fast Marching Cubes like algorithm for hypercubes of any dimension. To support this, we have developed a new algorithm to automatically generate the isosurface and triangulation tables for any dimension. This allows the efficient calculation of 4D isosurfaces, which can be interactively sliced to provide smooth animation or slicing through oblique hyperplanes. The former allows for smooth animation in a very compressed format. The latter provide better tools to study time-evolving features as they move downstream. We also provide examples in using this technique to show interval volumes or the sensitivity of a particular isovalue threshold.

1. INTRODUCTION

Given a continuous scalar field, i.e., a scalar function on R^d, an isosurface is the set of points with identical scalar values. The Marching Cubes algorithm by Lorensen and Cline is a popular, simple, and efficient algorithm for constructing a piecewise linear isosurface from scalar values in a three-dimensional regular grid [11]. The regular grid divides the volume into cubes whose vertices are the grid vertices and the isosurface is constructed piecewise within each cube. Each grid vertex is labeled positive, '+', or negative, '-', depending upon whether its value is greater than or less than the value of the isosurface. The structure of the isosurface in the cube depends only on the positive and negative labels of its eight vertices. Thus there are 2^8 ways in which the isosurface can intersect a cube. Most implementations of the marching cubes algorithm first build a table of these 2^8 cases and then use this table to determine the structure of the intersection of the surface with each cube. The actual location of the surface within the cube depends upon linear interpolations of the values of the cube vertices along the edges intersected by the isosurface.

* Silicon Graphics, Inc. , Mountain View, CA
praveenb@engr.sgi.com
+ Department of Computer & Information Science,
The Ohio State University, Columbus, OH
wenger@cis.ohio-state.edu
crawfis@cis.ohio-state.edu

By exploiting symmetry, Lorensen and Cline reduced the 2^8 cases to fourteen. They analyzed these fourteen cases by hand, constructing a triangulated surface for each case. Montani, Scateni and Scopigno added more cases to resolve certain ambiguities and inconsistencies in Lorensen and Cline's original algorithm [13]. This algorithm is commonly known as the Modified Marching Cubes algorithm.

A hypercube in four dimensions has sixteen vertices and 2^{16} possible vertex labelings. Even after exploiting symmetry, we found that we were left with 222 cases. Analyzing all these cases by hand, would have been a tedious and error prone exercise. Higher dimensions are even more problematic. Therefore, we first looked for a systematic way of generating the surface and its triangulation for each case.

Weigle and Banks generalized a variation of the marching cubes algorithm by replacing the cubes with simplices [19, 20]. Using barycentric subdivision, they broke each cube into simplices and then constructed the isosurface in each simplex. They triangulated the isosurface by recursively triangulating the various dimensional faces of the polyhedra composing the isosurface. Because of the simple structure and symmetry of a d-simplex, there are only d + 2 cases, each case corresponding to a different number of vertices with positive orientation. However, a d-cube breaks into between $d!$ and $2^{d-1}d!$ simplices, depending upon the decomposition used [19]. The time and space used by their algorithm also increases by a corresponding factor making it impractical for real problems.

Many techniques have been proposed for visualization of higher dimensions, but most of the work has dealt with the aspect of rendering higher dimensional data. Hanson et.al. in [7][8][9][10] use 4D lighting, shading, projections, rotations and plane-tracing (a generalization of ray tracing to 4D) among others, as techniques for visualizing higher dimensional data. Bajaj et.al. in [2] generalize splatting to higher dimensions. We will apply some of these techniques to our resulting higher dimensional surface and refer the reader to the references for visualization techniques in higher dimensions.

Specifically, we make the following contributions in this paper –

1. We present a new algorithm for automatically generating a lookup table of the isosurface and its triangulation for all the possible 2^{2^d} labelings of the hypercube in a d-dimensional

regular grid. To the best of our knowledge this has not been done before.

2. We outline the proof of correctness of our algorithm and also show that it generates a valid isosurface which is a a triangulated (d–1)-manifold with boundary in R^d.

3. We apply our algorithm and discuss its advantages to the following applications
   - Compact representation and smooth animation of time-varying isosurfaces.
   - Interval volumes and contour sensitivity.
   - Arbitrary slicing of higher dimensional data.

In section 2, we describe the algorithm in detail giving 2 and 3 dimensional examples and in section 3, we prove the correctness of the algorithm. In section 4, we analyze the results obtained for the 3-dimensional case. Then in section 5, we discuss the different applications and show how our approach is superior to those currently used.

2. ISOSURFACING ALGORITHM IN *R^D*

The isosurfacing algorithm can be divided into two primary steps. First, we generate the lookup table for a given dimension d. We do this for all possible 2^{2^d} cases and store the resulting table for step 2. Secondly, we generate the isosurface using the lookup table on each of the d-cubes intersected by the isosurface.

2.1 Lookup Table Generation

An edge of the hypercube intersects the isosurface if one endpoint has a positive label and one endpoint has a negative label (by the intermediate value theorem). For a given configuration of the vertices of the hypercube h, we find the midpoint of each such edge. Let $W^+(h)$ be the set of all such midpoints together with all hypercube vertices with a positive label. (Alternatively, we could use $W^-(h)$ which is the set of all such midpoints together with all hypercube vertices with negative labels.) The convex hull of $W^+(h)$ is a d-polytope lying in the hypercube and approximating the set of points in the d-cube with positive isovalues. In order to extract the desired isosurface, we remove any (d-1)-dimensional facets of this polyhedron which lie on the boundary of the hypercube. This causes the removal of any facets, which share a vertex with the hypercube. The remaining (d-1) facets comprise the isosurface in the hypercube and are written into the lookup table. To generate the complete table, we repeat the above for each of the 2^{2^d} possible cases.

Figure 1a illustrates the algorithm in 2 dimensions. The black vertices of the square (2-cube) correspond to positive labels while the white vertices correspond to the negative labels. The cross signs represent mid points of the edges with zero crossings. We start with the set $W^+(h)$, which is the union of the black vertices along with the crosses, i.e. the set of edge-isosurface intersection points, along with the positive labeled vertices. The black edges show the facets (1-simplices) of the convex hull of $W^+(h)$. Finally, removing facets on the boundary of the 2-cube gives the final triangulation of the isosurface. Figure 1b shows the steps for a 3 dimensional example.

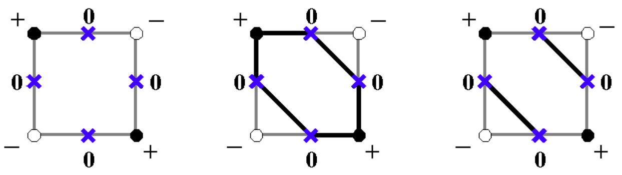


Figure 1a. Two-dimensional example of the algorithm.

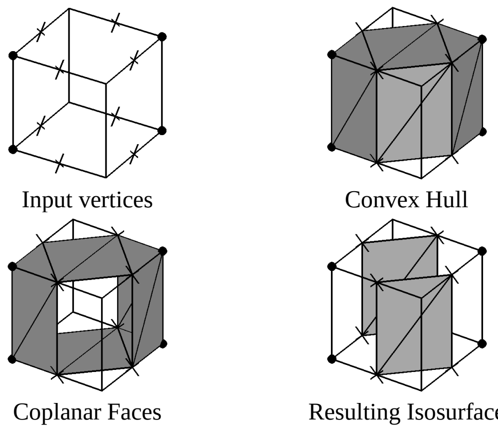


Figure 1b. Three-dimensional example of the algorithm.

The resulting d-polytope, which is the convex hull of $W^+(h)$, is not necessarily simplicial. In fact, in many instances it will contain (d-1)-facets which are not simplices, e.g. for the 3-dimensional case, there might exist 2D polygons with more than 3 vertices. These facets can be triangulated in more than one way. This is not a problem in 3D since any triangulation of the reconstructed isosurface in one cube is compatible with any triangulation of the reconstructed isosurface in another. However, in 4D the reconstructed isosurfaces in two adjacent hypercubes meet in a two dimensional polygon, for instance a planar quadrilateral. If the triangulations in these two hypercubes contain different diagonals of this quadrilateral, then the resulting triangulations will not match. This mismatch manifests itself as visible artifacts when the 4D isosurface is sliced along a given axis. Also, a mismatch makes it impossible to construct any face adjacency graphs, which might be needed for many algorithms. This is essentially the problem identified by Albertelli et. al. in [1] while constructing 3D tetrahedral grids by subdividing unstructured finite-element meshes.

To ensure consistent isosurface triangulations, we use an implementation by Clarkson [4] of an incremental convex hull algorithm. The algorithm is similar to the one described in [3] but without randomization. Clarkson's program adds one vertex at a time, updating the convex hull as necessary. It produces a complete triangulation of the convex hull whose structure depends upon the input order of the vertices. By lexicographically sorting the initial vertex set $W^+(h)$, as described in [14], we can fix this order and produce a 'canonical' triangulation. We claim that the 'canonical' triangulations of the reconstructed isosurfaces in two adjacent hypercubes properly match at their boundaries. For the 4-dimensional case, we also verified our results by computing all the 2^{24} possible triangulations of the isosurface for two adjacent 4-cubes and found that the triangulations on the boundary match. The convex hull algorithm used by Clarkson is not optimal but it seems the most practical solution to our problem of generating consistent triangulations.

Instead of using $W^+(h)$ containing the vertices with positive labels, we could have used the set $W^-(h)$ which contains the vertices with negative labels. Doing so gives a different, although equally valid, isosurface. However, using positive labels for some cases and negative labels for others can result in mismatches on the boundaries of the hypercubes. This was essentially the problem discovered by Durst in the original marching cubes algorithm [5].

2.2 Alternative Construction

An alternative method for constructing the isosurface would be to compute the convex hull of just the midpoints of the edges with zero crossings. If this convex hull is *d*-dimensional, then the boundary of this convex hull is a (*d*-1)-dimensional surface. Removing all the points lying on the boundary of the hypercube breaks this surface into two or more components. One set of components corresponds to the positive isosurface and the other set corresponds to the complementary 'negative' configuration. For instance in figure 2, removing the points on the boundary of the cube (coplanar facets), gives four components. Two of these components correspond to the desired positive isosurface and the remaining two to the isosurface for the complementary negative configuration. Finding the components corresponding to the desired positive isosurface would be more difficult compared to our approach. If the convex hull of the midpoints is (*d*-1)-dimensional, then this (*d*-1)-dimensional surface is the isosurface for both the positive and negative cases.

In our algorithm, by adding the positive vertices, we 'cover' the simplices in the negative components. This allows us to easily extract the desired 'positive' isosurface by removing the simplices lying on the boundary of the hypercube. If we had added the negative vertices to the set of midpoints, we would have obtained the isosurface corresponding to the complementary configuration. Note also that the convex hull of the midpoints and the positive vertices is always *d*-dimensional, while the convex hull of just the midpoints may not be. This slightly simplifies the convex hull construction.

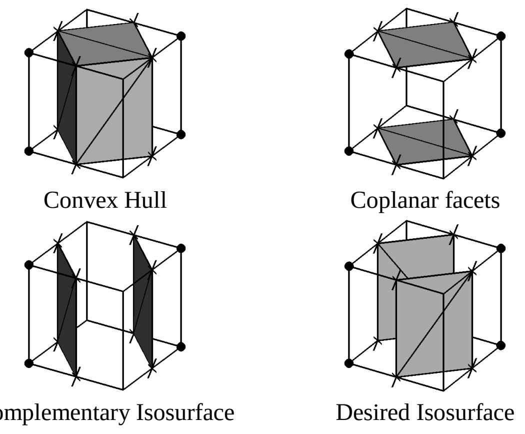


Figure 2. Alternative isosurface construction using only the mid points of edges with zero crossings

2.3 Isosurface Construction

This step is primarily an extension of the original Marching Cubes algorithm to higher dimensions, where the isosurface is constructed piecewise within each cell in the grid. For each logical *d*-cube, with 2^d voxels as its vertices, we use the lookup table generated in step 1 to construct the isosurface within the cube. The index for the *d*-cube depends on the labelings of the vertices of the *d*-cube and is given by,

$$\sum_{i=0}^{2^d-1} 2^i * color(i)$$

where, *color(i)* is one, if vertex *i* has a '+' label, and zero, if it has a '-' label. A value of either zero or $2^{2^d}-1$ for the index implies that the isosurface does not intersect this *d*-cube.

The actual surface-edge intersections are computed by linearly interpolating between the field values at the vertices of the intersected edges. Since, the algorithm computes the above for each *d*-cube in the grid, it is linear in the number of *d*-cubes. This can be improved by using techniques, which exploit spatial and temporal coherence, to locate the cells that are actually intersected by the isosurface [16][17][21][22].

3. ALGORITHM CORRECTNESS

Unless the underlying scalar function is known, it is not possible to reconstruct the exact isosurface from a discrete sampling of the function. A simple requirement is that the isosurface intersect the grid edges with one positive and one negative endpoint and only those grid edges. However, even in 3D there are many topologically distinct isosurfaces satisfying this restriction. We claim that our algorithm constructs a (*d*–1)-dimensional surface in R^d that satisfies this requirement. More specifically, it constructs a triangulated (*d*–1)-manifold with boundary in R^d. Note that a valid isosurface need not be a manifold but in many applications it is desirable that the surface be a manifold.

A set of points M in R^d is a (*d*–1)-dimensional *manifold with boundary* if the neighborhood of each point in M is homeomorphic to either R^{d-1} or a closed half-space of R^{d-1}. Intuitively, a manifold with boundary is a set of points that behave locally like a portion of (*d*–1)-dimensional Euclidean space or the boundary of a (*d*–1)-dimensional Euclidean half-space.

A set T of simplices defines a *simplicial complex* if the non-empty intersection of any two or more simplices of T is a face of each of these simplices. For instance, the non-empty intersection of any two tetrahedra is either a (triangular) face, or an edge or a vertex of the two tetrahedra and the non-empty intersection of any three tetrahedra is an edge or a vertex of all three.

Let T be the set of simplices returned by our algorithm. The set of points contained in all the simplices of T is called the underlying point set of T and denoted |T|.

We prove:
1. The set T of simplices defines a simplicial complex;
2. |T| is a (*d*–1)-dimensional manifold with boundary.

In this paper, we only have room for the main ideas behind the proof. The complete proof will appear in another article elsewhere.

Let h be a *d*-dimensional hypercube whose vertices are labeled positive or negative. As defined before, let $W^+(h)$ be the set of positive vertices of h and the midpoints of edges of h with one positive and one negative endpoint. Form $S^+(h)$ by taking the boundary of the convex hull of $W^+(h)$, removing any points on the boundary of h, and taking the closure of the remaining set. Formally, $S^+(h)$ equals $cl(\partial conv(W^+(h)) - \partial h)$, where *cl*, ∂ and *conv* are the closure, boundary and convex operators, respectively.

For instance, in Figure 1a, set $S^+(h)$ contains the two open line segments in the interior of h connecting the midpoints marked 0. It also contains the endpoints of these line segments, which lie on the boundary of h. These endpoints are in the closure of the open line segments. Similarly, in Figure 1b, set $S^+(h)$ contains the two open rectangles in the interior of h, but it also contains the rectangular boundary of these rectangles which lies on the boundary of h. Again, this rectangular boundary is the closure of these rectangles.

Let G be a regular grid whose vertices are labeled positive or negative. Our algorithm returns $S^+(h)$ for every hypercube h in G. Actually, it returns a set of $(d-1)$-dimensional simplices whose underlying point set is $S^+(h)$.

The interior of $S^+(h)$ is identical with the boundary of a *d*-dimensional convex set and so clearly forms a surface. It is much less clear that the surfaces defined by $S^+(h_1)$ and $S^+(h_2)$ for two adjacent grid hypercubes h_1 and h_2 fit together properly at their boundaries. The key point to note is that, for any k-face f of a hypercube h, the intersection of $S^+(h)$ and f is completely determined by the labels of the vertices of f.

Lemma: Let G be a regular grid whose vertices are labeled positive or negative. If h_1 and h_2 are two adjacent grid hypercubes in R^d and $f = h_1 \cap h_2$, then $S^+(h_1) \cap f = S^+(h_2) \cap f$.

Let $(p_1,\ldots,p_n)$ be a sequence of points in convex position in R^d. We call a triangulation T' of $conv(p_1,\ldots,p_n)$ canonical if $T'-p_n$, the simplicial complex T' with all the simplices incident on p_n removed, is a canonical triangulation of $conv(p_1,\ldots,p_{n-1})$. A single simplex is a canonical triangulation. Intuitively, a canonical triangulation is built by adding points and their incident simplices in sequential order. This is exactly what Clarkson's convex hull program [4] does. Thus the simplices reported by that program form a cannonical triangulation of the facets of the convex hull. The points are sorted in lexicographic order. We claim and prove that the cannonical triangulations of $S^+(h_1)$ and $S^+(h_2)$ are identical on their intersection. Thus the simplices in T fit together to define a simplicial complex.

Since T defines a simplicial complex, we need only check neighborhoods of the vertices of T to prove that |T| is a manifold with boundary. We construct an explicit mapping from the neighborhood of each vertex to R^{d-1} or a closed half-space in R^{d-1}. Thus |T| is a $(d-1)$-dimensional manifold with boundary. Our construction of $W^+(h)$ and $S^+(h)$ ensures that this manifold intersects the grid edges with one positive and one negative endpoint and only those grid edges.

Our proof only holds when the isosurface vertices are located at the midpoints of edges. In Section 2.3 we propose computing the locations of isosurface vertices by linearly interpolating between the field values at the endpoints of intersected edges. It may be possible that replacing the midpoints by interpolants causes the isosurface to intersect itself. This cannot happen in three dimensions, but we do not know if it can happen in higher dimensions. The simplices will still fit together properly to form an abstract simplicial complex and a manifold with boundary, but the embedding of that manifold given by the values of the interpolated points may intersect itself within a given hypercube.

Instead of using a lookup table, we could construct $conv(W^+(h))$ and $S^+(h)$ for each hypercube separately. We could then replace the midpoints in each $W^+(h)$ by the linear interpolants and form the convex hull of these interpolants. The proof of correctness outlined above holds for this modified algorithm. The only difference is that the set $W^+(h)$ has changed.

A more practical approach might be a hybrid scheme in which we sometimes use table lookup and sometimes construct $S^+(h)$ from interpolated values. However, it is not clear how such a scheme would guarantee that surfaces and triangulations from adjacent hypercubes would fit together properly.

4. COMPARISON TO THE MODIFIED MARCHING CUBES

Our algorithm generates topologically the same lookup table as proposed in the Modified Marching Cubes algorithm [13], except for one case where we get a different, but an equally valid, isosurface (figure 3). When two negative vertices are diagonally opposite to each other, our algorithm generates a tunnel-like surface, while the Marching Cubes algorithm gives two disconnected surfaces. One drawback of this approach is that it increases the number of triangles from two to six for this particular case. It is curious to note that this is not one of the ambiguous cases pointed out in the Nielson and Hamann's Asymptotic Decider [15]. Our method has a tendency to keep the '+' vertices together, preserving pathways when possible. This property can also be seen in the first two cases in figure 4, which shows the ambiguous cases given in [13] and the respective complementary cases generated by our algorithm. It can also be seen that all the cases generated by our algorithm are topologically the same as those suggested in Modified Marching Cubes.

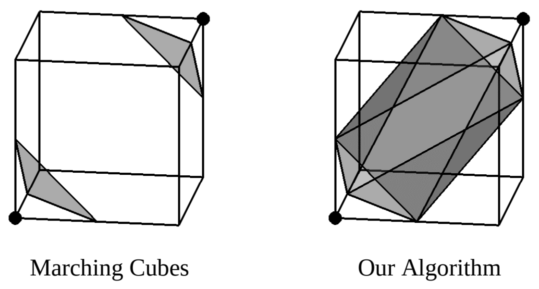


Figure 3. Different triangulations generated by our algorithm as compared to Marching Cubes.

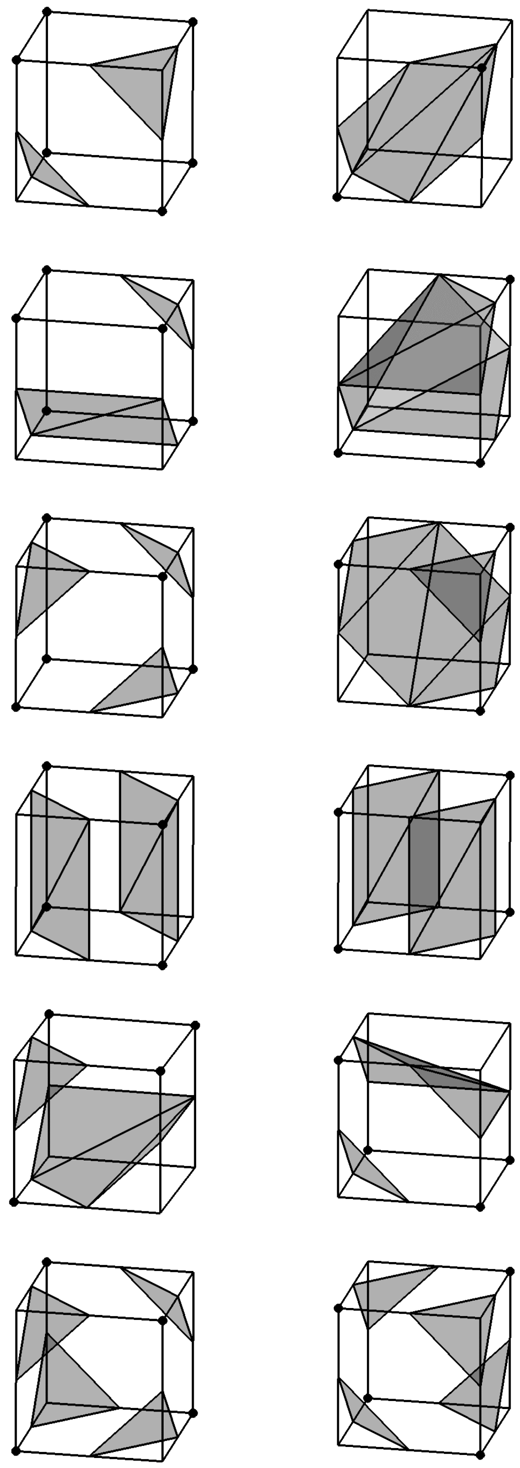


Figure 4 The left row shows the ambiguous cases shown in Modified Marching Cubes while, the right row shows the corresponding complementary cases generated by our algorithm.

5. APPLICATIONS

5.1 Time Varying Isosurfaces

A number of different techniques have been introduced for fast isosurface extraction and compressed representation of time-varying fields [16][17][21][22]. Our algorithm provides another approach to compact representation of time varying isosurfaces, similar to that of Weigle and Banks [20]. The drawback of their method is that they decompose each 4-cell into 192 4-simplices and then recursively contour each of the simplices. This approach results in a very large number of tetrahedra compared to our method.

We output our tetrahedral grid sorted in time, hence, we need only compute the intersection for a given interval of the time constraint. This approach makes the slicing independent of the total number of time steps and speeds up the slicing considerably. On an SGI Octane, computing a time slice is interactive for an isosurface already generated from a 40x40x40x36 size data set. This is much faster compared to 2 minutes in [20] and can be attributed to the large number of tetrahedra generated due to simplicial decomposition of each 4-cell. Constructing the isosurface in R^4 allows slicing at non-integral time steps, effectively merging the steps of interpolation and isosurface extraction into one, allowing us to generate smooth animations of the time-varying isosurface very efficiently.

For 10 time steps of the Jet Shockwave data set, an isovalue of 37 generated an isosurface with 8,021,739 tetrahedra and 1,394,104 vertices in 1109 seconds. The isosurface intersected 1,317,975 hypercubes, giving an average of around 6 tetrahedra per hypercube. The total number of triangles generated for the same 10 time steps by Marching Cubes was 1,796,350.

Time-varying isosurfaces can also be used as a compressed representation in volume morphing applications. Figure 7 shows a sequence of frames generated using a time varying function. This function is radial at time 0, migrates to a toroidal function at time 1, and then to a union of two radial functions at time 2. The movie file, morph.mpv, (on the CD ROM proceedings) shows an animation of this function. Note how easily this technique can handle topology changes.

5.2 Interval Volumes and Contour Sensitivity

For a trivariate function $f(x,y,z)$ sampled on a 3D rectilinear grid, the interval volume [14] is defined by $I_f(\alpha,\beta) = \{(x,y,z): \alpha \leq f(x,y,z) \leq \beta \}$. Fujishuro [6] discuss applications of interval volumes and propose a solid fitting algorithm for tetrahedralizing the interval volume by extending Marching Cubes. Max et.al. [12] and Nielson et.al. [14] compute the tetrahedralization by decomposing each cube in the grid to five tetrahedra. Nielson then uses an efficient lookup table to compute the interval volume within each simplex and decompose it into tetrahedra. Since they do this for simplices rather than cubes, the number of tetrahedra generated is very large. We project the problem of interval volume tetrahedralization as a 4-dimensional problem as follows.

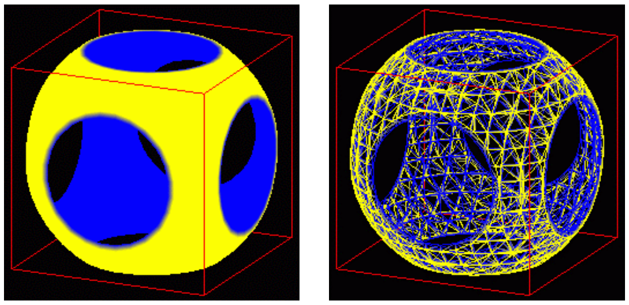


Figure 5. Interval volume for the sphere function

From $f(x,y,z)$, construct a 4D function $F(x,y,z,t)$ given by,

$$F(x,y,z,t) = \begin{cases} f(x,y,z) - \alpha & \text{for } t = 0 \\ f(x,y,z) - \beta & \text{for } t = 1 \end{cases}$$

The interval volume I_f is then given by the isosurface $F(x,y,z,t) = 0$. We use our algorithm to compute the isosurface for $F = 0$ and then use parallel projection to project the resulting isosurface to 3D to give the tetrahedralization of I_f. In order to compare our technique to that of Nielson's, we show the results obtained for the following function sampled on a 14x14x14 grid for $\alpha = 0.35$, $\beta = 0.37$ (Figure 5).

$$f(x,y,z) = (x - 0.5)^2 + (y - 0.5)^2 + (z - 0.5)^2$$

The resulting tetrahedralization consists of 4204 tetrahedra and 1496 vertices as compared to 8500 tetrahedra and 3224 vertices in [14]. Furthermore, the 3D isosurface can be animated to show the contour sensitivity by slicing the 4D isosurface. Slicing allows the user to quickly move back and forth between different isovalues to see the contour sensitivity. Sections of the isosurface, which are more sensitive to the isovalue can be seen to change more rapidly compared to other sections.

In order to visualize a larger range of isovalues ($\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n$) this idea can be extended to tetrahedralize the volume using n steps along the t axis. Thus, the function $F(x,y,z,t)$, is given by

$$F(x,y,z,t_i) = f(x,y,z) - \alpha_i \qquad \text{for } 1 \le i \le n$$

Using 5 steps, we generated the interval volume for timestep 60 of Jet Shockwave for isovalues of 17 through 57. The resulting interval volume consisted of 2,563,331 tetrahedra and 501,110 vertices compared to over 7 million triangles generated by Marching Cubes for the same range of values. The movie file, interval.mpv, shows the slices for the interval at intermediate isovalues.

5.3 Arbitrary Slicing of higher dimensions

The higher dimensional isosurfaces can be sliced along arbitrary axes or hyperplanes in R^4. For example, slicing along the z-axis allows one to see the various trends in the isosurface at a given z value in the grid. Animating the slice allows one to see the isosurface changing along the z-axis. This can be useful to follow a time-evolving structure, by taking a slice in the structures direction of movement.

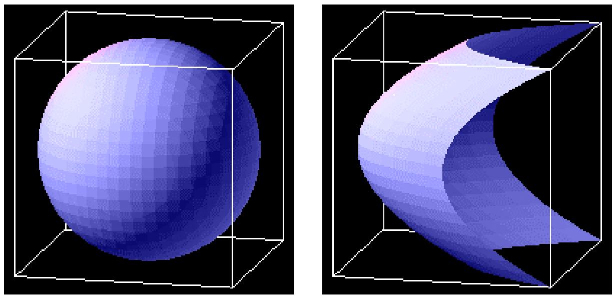


Figure 6. Slices of F along different coordinate axes

Figure 6 shows slices of the following function along different axes. The function gives a sphere along the w-axis, while along the x-axis a paraboloid is generated.

$$F(x,y,z,w) = (x - 0.5)^2 + (y - 0.5)^2 + (z - 0.5)^2 - \mathrm{c}w - \mathrm{d}$$

where, c and d are constants.

Figure 8 shows slices of the Jet Shockwave isosurface along different axes. Interesting features can be seen from the slices, e.g. the y-slice shows features developing with time in the cross section of the isosurface along the x-z plane. The movie file, zSlice.mpv, shows an animation of the z-slice for increasing z values. Patterns can be seen as the time evolving 'tube' is sampled at different z-values.

6. RESULTS

In this paper, we have shown an efficient algorithm for constructing isosurfaces in d-dimensional grids and also discussed a few applications of the algorithm. The lookup table for 4-dimensional contours has 2^{16} entries with a maximum of 26 tetrahedra in a given case. The average was approximately 13 tetrahedra for the complete table. In practice, the average is much less then 13. For the Jet Shock Wave data set, the average number of tetrahedra per hypercube was approximately 6.

We also generated a five-dimensional isosurface by combining the interval volume and time varying techniques together. Rather than precompute the lookup table, we used a lazy evaluation method with our algorithm to generate the entries as needed. The interval hypervolume for the Jet Shockwave having isovalues 27 and 37 for timesteps 56 and 57, consisted of 3.3 million hypertetrahedra and 2.2 million vertices for the 256x256x256x2x2 size data set. The average number of hypertetrahedra per intersected 5-cube was approximately 24.

7. FUTURE WORK

For four-dimensional isosurfaces, tetrahedral mesh compression techniques can be employed to reduce the memory and processing overhead for the rendering system, also allowing level-of-detail rendering of the time evolving isosurface. We plan to extend current tetrahedral mesh simplification techniques [18] to allow compression in four dimensions.

A future direction of research would be to integrate time-varying isosurfaces with a volume morphing application. This would allow the user to interactively slice back and forth to animate the volume as well as interactively manipulate the model in three dimensions.

Further, fast isosurface extraction algorithms [16][17][21][22] can be used to replace the linear search for the cells intersected by the isosurface to speed up the isosurface construction. A lookup table for a five dimensional isosurface, will have 2^{32} entries. We are currently working on ways to generate entries on the fly and then caching the more frequently used ones.

8. ACKNOWLEDGEMENTS

This work was supported by NSF career grant #ACI- 9876022. We would like to thank Zhi Yao, Donglin Liang and Hoseok Kang for contributing to the introductory ideas for this work. We would also like to thank Zbigniew Fiedorowicz for helpful consultations on topology. The Jet Shockwave data is part of the Advanced Visualization Technology Center's data repository and appears courtesy of the University of Chicago.

9. REFERENCES

[1] ALBERTELLI, G. AND CRAWFIS, R.A. Efficient subdivision of finite-element datasets into consistent tetrahedra. In *Proceedings of Visualization '97 (1997)*. pp. 213-219

[2] BAJAJ, C.L., PASCUCCI, V. AND RABBIOLO, G. Hypervolume Visualization: A challenge in simplicity. In *Proceedings of the 1998 Symposium on Volume Visualization (1998)*, pp. 95-102.

[3] CLARKSON, K.L., MEHLHORN, K. AND SEIDEL, R. Four results on randomized incremental constructions. *Comp. Geom.: Theory and Applications* (1993).

[4] CLARKSON, K.L. http://cm.bell-labs.com/who/clarkson and http://cm.bell-labs.com/netlib/voronoi/hull.htm.

[5] DURST, M. Additional reference to Marching Cubes. *Computer Graphics 22*, 4 (1988), pp. 72-73.

[6] FUJISHIRO, I., MAEDA, Y., SATO, H. AND TAKESHIMA, Y. Volumetric data exploration using interval volume. In *IEEE Transactions On Visualization and Computer Graphics 2, 2 (1996)*, pp. 144 –155.

[7] HANSON, A.J. Rotations for n-dimensional Graphics. *Graphics Gems V*. pp. 55-64. Academic Press, Cambridge, MA, 1995.

[8] HANSON, A.J. AND HENG, P.A. Visualizing the Fourth Dimension using Geometry and Light. In *Proceedings of Visualization '91 (1991)*, pp. 321-328.

[9] HANSON, A.J. AND CROSS, R.A. Interactive visualization methods for four dimensions. In *Proceedings of Visualization '93 (1993)*, pp. 196-203.

[10] HANSON, A.J. AND HENG, P.A. Four-dimensional views of 3D scalar fields. In *Proceedings of Visualization '92 (1992)*, pp. 84-91.

[11] LORENSEN, W. AND CLINE, H. Marching Cubes: A high resolution 3d surface construction algorithm. *Computer Graphics 21*, 4 (1987), pp. 163-170.

[12] MAX, N, HANRAHAN, P. AND CRAWFIS, R. Area and volume coherence for efficient visualization of 3D scalar functions. *Computer Graphics 24*, 5 (1990), pp. 27-33.

[13] MONTANI, C., SCATENI, R. AND SCOPIGNO, R. A modified look-up table for implicit disambiguation of Marching Cubes. *Visual Computer 10* (1994), pp. 353-355.

[14] NIELSON, G.M. AND SUNG, J. Interval Volume Tetrahedralization. In *Proceedings of Visualization '97 (1997)*.

[15] NIELSON, G.M. AND HAMANN, B. The Asymptotic Decider: Resolving the ambiguity in Marching Cubes. In *Proceedings of Visualization '91 (1991)*. pp. 83-91.

[16] SHEN, H.W. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In *Proceedings of Visualization '98 (1998)*, pp. 159 – 164.

[17] SUTTON, P. AND HANSEN, C.D. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (T-BON). In *Proceedings of Visualization '99 (1999)*, pp. 147 – 154.

[18] TROTTS, I.J., HAMANN, B., JOY, K.I. AND WILEY, D.F. Simplification of tetrahedral meshes. In *Proceedings of Visualization '98 (1998)*. pp. 287-295.

[19] WEIGLE, C. AND BANKS, D. Complex-valued contour meshing. In *Proceedings of Visualization'96* (1996) pp. 173-180.

[20] WEIGLE, C. AND BANKS, D. Extracting isovalued features in 4-dimensional scalar fields. In *Proceedings of the 1998 Symposium on Volume Visualization* (1998), pp. 103-110.

[21] WILHEMS, J. AND GELDER, A.V. Octrees for faster isosurface generation. In *ACM Transactions on Graphics 11, 3(1992)*, pp 201-227.

[22] WILHEMS, J. AND GELDER, A.V. Multi-dimensional trees for controlled volume rendering and compression. In *Proceedings of the 1994 Symposium on Volume Visualization* (1994), pp. 27- 34.

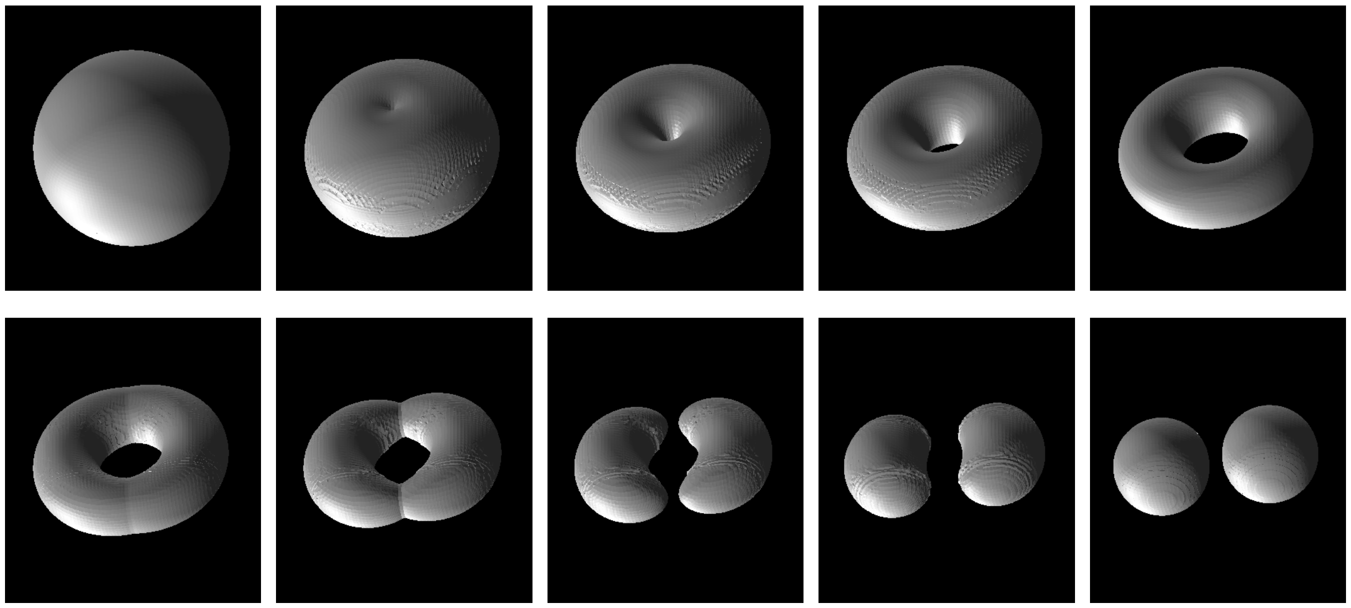
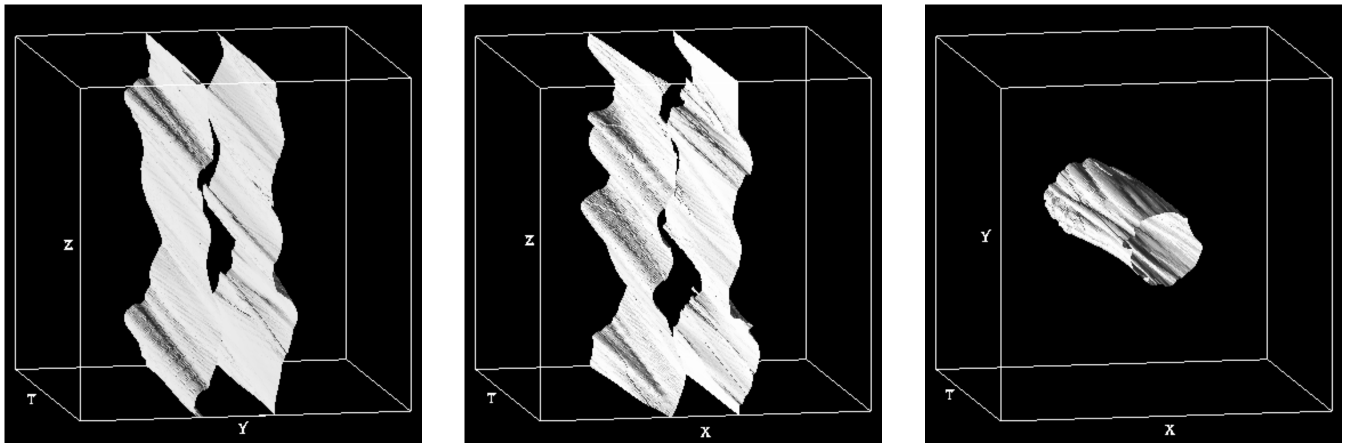


Figure 7. Sequence of 10 frames showing slices of a time varying function.



Slice along X-axis Slice along Y-axis Slice along Z-axis

Figure 8. Slices of a time-varying isosurface for the Jet Shockwave data set along different axes. Isovalue = 37, Timesteps = 56-65

Isosurfacing in Higher Dimensions

Praveen Bhaniramka, Rephael Wenger, Roger Crawfis