Extending Interval Volumes into Four Dimensions

Caixia Zhang*, Daqing Xue*, Roger Crawfis*, Rephael Wenger*
The Ohio State University

ABSTRACT

In this paper, we study the interval segmentation and direct rendering of time-varying volumetric data to provide a more effective and interactive volume rendering of time-varying structured and unstructured grids. Our segmentation is based upon intervals within the scalar field between time steps, producing a set of geometrically defined time-varying interval volumes. To construct the time-varying interval volumes, we cast the problem one dimension higher, using a five-dimensional iso-contour construction for interactive computation or segmentation. The key point of this paper is how to render the time-varying interval volumes directly. We directly render the 4D interval volumes by projecting the 4D simplices onto 3D, decomposing the projected 4-simplices to 3-simplices and then rendering them using a modified hardware-implemented projected tetrahedron method. In this way, we can see how interval volumes change with the time in one view. The algorithm is independent of the topology of the polyhedral cells comprising the grid, and thus offers an excellent enhancement to the volume rendering of time-varying unstructured grids. Another advantage of this algorithm is that various volumetric and surface boundaries can be embedded into the time-varying interval volumes.

CR Categories: I.3.3 [Picture/Image Generation]: Display algorithms

Keywords: Volume Rendering, Unstructured Grids, Interval Volumes, Time-Varying data, Projected Tetrahedra

1 INTRODUCTION

With the widespread use of high performance computing systems, some application simulations are capable of producing large datasets. These simulations tend to be time varying, adding another dimension to the problem. This paper mainly deals with the interval segmentation and rendering for time-varying structured and unstructured datasets.

A traditional method to render time-varying data is to take a snapshot of the data for each particular time step and generate an animation from the time series data. Some people have worked on the hypervolume visualization [2] and high dimensional direct rendering of time-varying volumetric data [27], but their algorithms do not apply for unstructured grids.

Interval volumes have been used to segment and render structured and unstructured volumetric data [7]. It is a region-of-interest extraction algorithm, and fast volume visualization techniques are employed to render the interval volumes. Under the framework of the high-dimensional iso-contouring algorithm, interval volumes can easily be computed using two schemes for time varying data sets. The first scheme computes the interval volumes separately for each time step of the dataset using the algorithm discussed in [7]. The user can then cycle through all the time steps to visualize the data. It is obvious that this scheme does not show the relationship of the interval volumes between different time steps well.

An alternative approach is to use the high-dimensional isosurfacing algorithm to compute a 4-dimensional volume representing a time-varying interval volume. This can be accomplished by applying the isosurfacing algorithm directly on a 5-dimensional grid to generate a surface comprised of 4-simplices. Then, we can render the 4-tetrahedra by slicing the interval volumes between the time steps. An alternative method and the focus of this paper is to render the 4-simplex from the interval volumes by integrating in the time directly, rather than projecting and slicing them.

In the following sections we look at some of the previous work done in this domain. We then give an overview of our time-varying interval volume computation algorithm, followed with rendering methods and visualization techniques for the time-varying interval volumes. We end with some results of our work and present future directions for research in this domain.

2 PREVIOUS WORK

Previous work that relates to our research primarily focuses on high-dimensional scientific visualization, unstructured volume rendering and interval volumes.

High-Dimensional Visualization – Hanson et al. [11, 12, 13] introduced a general technique, as well as an interactive system, for visualizing surfaces and volumes embedded in four dimensions. In their method, 3D scalar fields were treated as elevation maps in four dimensions in the same way 2D scalar fields could be viewed as 3D terrains. Bajaj et al. [2] developed an interface that provides “global views” of scalar fields independent of the dimension of their embedded space and generalized the object space projection technique into a hyper-volume projection method. Texture mapping hardware was utilized to directly render n-dimensional views of the global scalar field. Woodring et al. [26, 27] treated the time-varying data as four-dimensional data, and applied high dimensional slicing and projection techniques to generate an image hyperplane. The results of their technique generated a volume that is the projection of hyperplanes along a 4D projection vector, which can be rendered using traditional volume rendering techniques.

Unstructured Volume Rendering – Shirley and Tuchman [19] presented an algorithm for hardware accelerated rendering of unstructured tetrahedral grids by approximating the projection to screen space using a set of triangles. Grids consisting of different cells are first decomposed into a tetrahedral representation using simplicial decomposition techniques [1][17]. Williams extended Shirley-Tuchman’s approach to implement direct projection of other polyhedral cells in their HIAC rendering system [25] and used high accuracy light integration functions to model the light transport through the medium [24]. Recently, with the advent of

*{zhangc, xue, crawfis, wenger}@cse.ohio-state.edu

programmable graphics hardware, a tremendous amount of work has been done in implementing the Shirley-Tuchman algorithm on graphics hardware using the programmable vertex and fragment shader pipelines on the GPUs [21][28][15]. In all of the above cases, the rendering performance of the projected tetrahedra algorithm is typically proportional to the number of cells to be rendered. The rendering process involves visibility sorting (usually $O(n\log n)$) and projection ($O(n)$) of the polyhedral cells. As an alternative to projection, polyhedral cells can also be rendered using ray casting [22].

Interval Volumes – An *interval volume* is the set of points in a scalar field enclosed between two isosurfaces defined by two different isovalues. Fujishiro [8] introduced interval volumes as a solid fitting algorithm. A few applications of interval volumes were presented in [10][9]. Fujishiro computed a tetrahedralization of the interval volume by computing the intersection of two convex polyhedra enclosed by the isosurfaces given by the Marching cubes algorithm [16], within each cell. Nielson [18] computed the tetrahedralization by first decomposing each cube in the grid to five tetrahedra. Nielson then used an efficient lookup table to compute the interval volume within each simplex and decompose it into tetrahedra. The tetrahedralization was constructed manually by analyzing all the possible intersections of a tetrahedron with an interval enclosed by two isosurfaces. Banks [3,4] counted the cases for a family of visualization techniques, including iso-contours and interval volumes.

The above work is on the interval volumes of a single scalar field. For interval volumes with respect to a time-varying dataset, Ji et al. [14] tracked the interval volumes using higher dimensional isosurfacing, and rendered an iso-contour surface of the interval volumes. They did not directly render the 4D interval volumes.

In this paper, we work on the computation and direct rendering of the time-varying interval volumes. We use interval volumes to create disjoint volume segments, or intervals. Interval volumes provide a segmentation of the data into easily discernable regions. The direct rendering of the time-varying interval volumes makes it possible to get the distribution and relationship of the interval volumes across time steps, and help us to understand the time-varying structured and unstructured volumetric fields.

3 Time-varying Interval Volume Computation

In [5], we presented a new algorithm for computing isosurfaces in arbitrary dimensional data sets. The algorithm proceeds by generating isosurface patches within each d-dimensional polyhedral cell comprising the d-dimensional grid. The output of the algorithm is a set of (d-1)-dimensional simplices forming a piecewise linear approximation to the isosurface. The algorithm constructs the isosurface piecewise within each cell in the grid using the convex hull of an appropriate set of points. In [6] we present a proof of correctness for the d-dimensional isosurface construction and show that it correctly produces a triangulation of a (d-1)-manifold with boundary. Here, we give a short review of the algorithm. See [7] for more details.

For a function $f(x,y,z)$ sampled on a three dimensional grid, the interval volume [8] is defined by $I_f(\alpha,\beta) = \{(x,y,z): \alpha \le f(x,y,z) \le \beta\}$. More generally, for a function $f: R^d \rightarrow R$ in any dimension, the interval volume is defined by $I_f(\alpha,\beta) = \{(x_1,\ldots,x_d): \alpha \le f(x_1,\ldots,x_d) \le \beta\}$. Intuitively, the interval volume is the set of points enclosed between the two isosurfaces corresponding to the isovalues, α and β. For a d-dimensional grid, the interval volume is a d-dimensional subset of the grid and can be represented by a collection of d-simplices.

The interval volume algorithm proceeds as follows:

1. Let $f(x_1,\ldots,x_d)$ define a d-dimensional function.
2. Let scalar values, α, β ($\alpha < \beta$), be the desired isovalues bounding the interval.
3. Let $F(x_1,\ldots,x_d, w)$ be the (d+1)-dimensional function, given by, $F(x_1,\ldots,x_d,w) = f(x_1,\ldots,x_d) - (\alpha(1-w) + \beta w)$, such that
$$F(x_1,\cdots,x_d,w) = \begin{cases} f(x_1,\cdots,x_d) - \alpha, & for\ w = 0 \\ f(x_1,\cdots,x_d) - \beta, & for\ w = 1 \end{cases}$$
4. Compute the zero-valued isosurface, S, given by $F(x_1,\ldots,x_d, w) = 0$ for $0 \le w \le 1$.
5. Let π be the projection function mapping R^{d+1} to R^d given by $\pi(x_1,\ldots,x_d,x_{d+1}) = (x_1,\ldots,x_d)$. The desired interval volume, $I_f(\alpha,\beta)$, is then given by $\pi(S)$.

For a time-varying scalar grid with hexahedral cells, the construction of the time-varying interval volumes is a five-dimensional isosurfacing problem. Given a four dimensional scalar field $f(x,y,z,t)$, the interval volume consists of all the points which satisfy $\alpha \le f(x,y,z,t) \le \beta$. Following the above interval volume algorithm, in order to compute the time-varying interval volume, we first create a five dimensional scalar field $F(x,y,z,t,w)$, such that $F(x,y,z,t,0) = f(x,y,z,t) - \alpha$ and $F(x,y,z,t,1) = f(x,y,z,t) - \beta$. Then, the interval volume $\alpha \le f(x,y,z,t) \le \beta$ can be extracted by first computing the zero isosurface of the five dimensional function $F(x,y,z,t,w)$, and then projecting the resulting isosurface along the w axis to four dimensional space.

Here, we should note that the entries of the isosurface lookup table for 5D hypercube are too large to be stored in the main memory. Since a 5D hypercube contains 32 vertices, the size of the table will contain 2^{32} = 4G entries. As pointed out in [14], not all the four billion cases are possible. Only $3^{16} \approx$ 43M entries are possible for interval volumes. However, this size may be still too large to be processed in core. One solution is to compute the entries of the lookup table at runtime and cache them into a hash table which is small enough to fit into the main memory. In this paper, we use this caching method to store the 5D isosurface lookup table. See [20] for the lookup table generation code.

Since the isosurface triangulation is consistent, the interval volume triangulation will also be consistent. The problem of face mismatches across cell boundaries is often referred to as the *cracking problem* [18]. Our algorithm handles the cracking problem in the table generation stage by using a lexicographical ordering of the isosurface vertices and then building the convex hull incrementally, adding one vertex at a time in the specified order. This is similar to the scheme used by [18] and [17], which ensures canonical triangulations across cell boundaries and generates consistent meshes. However, we still have to worry about the decomposition from 4-simplices to 3-simplices for the purpose of rendering. And we will address this problem in the next section.

4 Time-varying Interval Volume Rendering

After we extract the 4-simplices comprising the 4D interval volume, one rendering possibility is to slice the 4-simplices parallel to the time axis to generate 3-simplices (i.e. 3D tetrahedra) for a corresponding time step. The resulting 3D tetrahedra can then be rendered. This scheme is analogous to rendering time varying isosurfaces [5], but allows slicing at non-

integral time-steps to compute interpolated interval volumes between consecutive time steps. Figure 1 shows one example of the time slicing. The left image and the right image are the interval volumes with respect to the two time steps *t1* and *t2*. The middle image is the corresponding interval volume with the time value $t = (t1 + t2)/2$.

In this paper, we are more interested in the direct rendering of the 4-simplices extracted from the 5D isosurface lookup table, in order to understand the distribution and relationship of the time-varying interval volumes across time steps. Now the challenge is how to render the 4-simplices to the 2D image space. The following subsections will explain the projection of 4-simplices to 3-simplices and the projection of 3-simplices to image space.

4.1 Projection of 4-simplices to 3-simplices

4.1.1 Classification of projected 4-simplices

Each 4-simplex extracted from the 5D isosurface lookup table has five vertices with coordinates (x, y, z, t). Every two vertices out of the five are connected by an edge. The 4-simplex is projected to 3D along a given projection direction in 4D: $\pi(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3)$, where π is the projection function. Here, we use a projection along the t axis as an example.

The five projected vertices compose some volume in three dimensions, except in some degenerated cases where the five projected vertices form a triangle, or a line, or a point. There are six common cases for the spatial relationship of the projected 4-simplex, as shown in Figure 2. They are either labeled as general cases, or the degenerated cases which still compose a volume in 3D (for example, four vertices coplanar, three vertices colinear, and two vertices coincident). The more severe degenerate cases, where the projected vertices are all co-planar, are not considered in this paper, because they do not produce volumetric entities.

The projected 4-simplices are classified as different types based on the spatial relationship of the five vertices of the projected 4-simplex along the t axis in three-dimensional space. Figure 2 illustrates the six common cases of the 4-simplex projection. Class 1 and class 2 are general cases. In class 1, no vertex is inside a tetrahedron composed of the other four vertices. In class 2, one vertex is inside the tetrahedron of the other four vertices (in Figure 2, P5 is inside the tetrahedron P1P2P3P4). Class 3, class 4 and class 5 are degenerated cases. In class 3, four vertices (P1, P2, P3, P4) are coplanar. P5 is inside the triangle of P1P2P3 in class 3(b). In class 4, three vertices (P1, P4, P3) are colinear, and in class 5, two vertices (P4, P5) are coincident.

A projected 4-simplex with 5 vertices is classified step by step using the flow chart in Figure 3.

In this paper, we classify the projected 4-simplices into two general cases and four degenerated cases. The degenerated cases generate fewer decomposed tetrehedra in section 4.1.2 and improve the rendering performance. We could just consider only

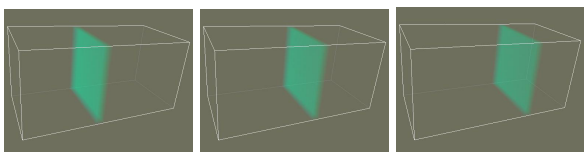


Figure 1. Results of time slicing

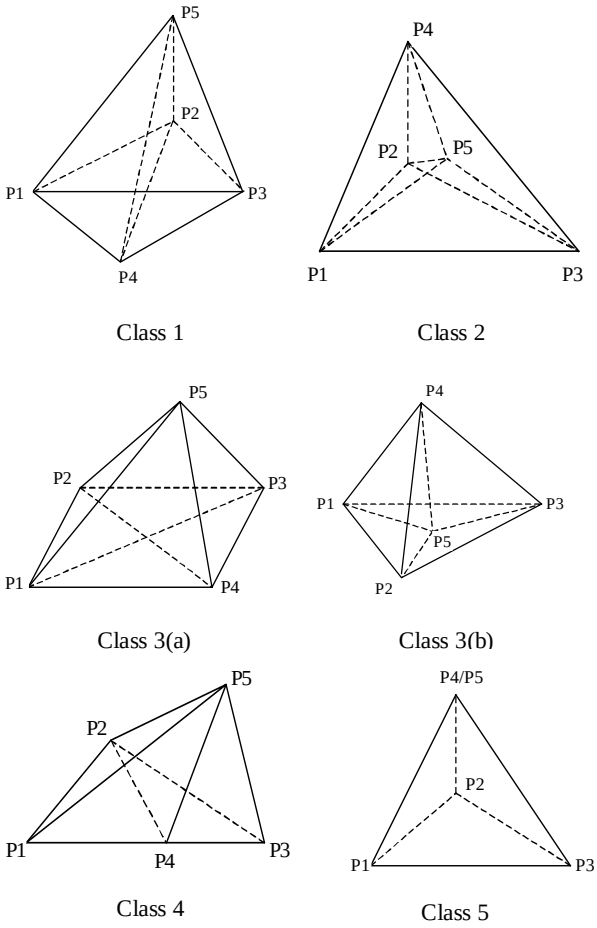


Figure 2. Classification of projected 4-simplex

general cases and combine the degenerated cases into the general cases. For example, class 3(b) and class 5 can be combined into class 2, with the vertex P5 moving from the face P1P2P3 or from the vertex P4 to inside the tetrahedron P1P2P3P4. Similarly, class 3(a) and class 4 can be combined into class 1, with the vertex P4 moved from the position coplanar with P1P2P3 or colinear with P1P3 to the position on the opposite side of P5 with respect to the face P1P2P3. This generalization of the cases will generate more tetrahedra (many of them with nearly zero volume) and/or will need some checking to distinguish them in the tetrahedralization stage.

This classification will guide us in decomposing the projected 4-simplices into tetrahedra for the purpose of rendering.

4.1.2 Tetrahedralization of projected 4-simplices

Our first attempt at this problem was to project 4-simplices to 3-simplices along the time axis by simply ignoring the time value (t) and keeping only the position information (x, y, z) for each vertex. Then, the projected 4-simplices are decomposed into tetrahedra in 3D space based on the above classification in Figure 2. Table 1 was constructed by hand and shows the possible decomposition of each class.

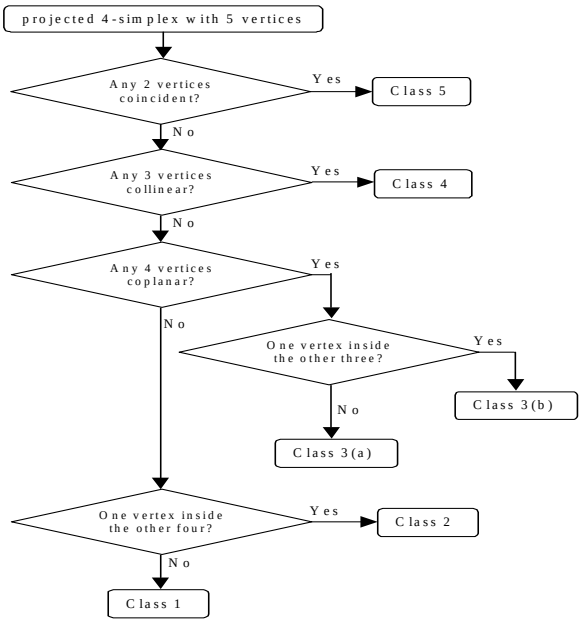


Figure 3. Flow chart of the classification of the projected 4-simplex

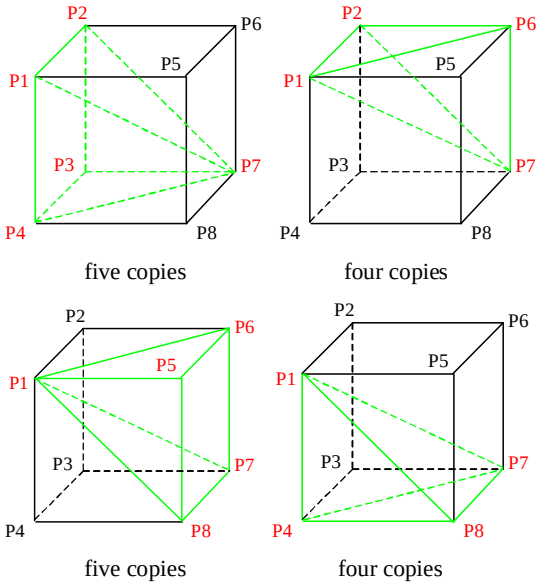


Figure 5. Overlapping copies inside a cube cell

After rendering the resulting 3D tetrahedra using the Projected Tetrahedron method, we found that the result was not correct. An image of a constant plate is shown in Figure 4. There are some obvious patterns on the plate. As we know, many 4-simplices are extracted from the lookup table for each hypercube cell, and then are projected to 3D and decomposed to tetrahedra. By keeping track of the tetrahedral components inside each cell, we find that each tetrahedron is in right place and the tetrahedra as a whole fill the cell. However, we also find that the projections of a set of 4-simplices overlap. For example, for a cube cell from our constant plate example as shown in Figure 4, some space is shared five times by the projection of 4-simplices, while other space is shared only four times (as shown in Figure 5). This uneven overlapping distribution of the projected 4-simplices causes a non-constant opacity throughout the cell.

Table 1. Original decomposition of the projected 4-simplices

Class	Possible Tetrahedra
Class 1	P1P2P3P4 and P1P2P3P5 or: P1P2P4P6, P1P3P4P6, P2P3P4P6, P1P2P5P6, P1P3P5P6 and P2P3P5P6
Class 2	P1P2P3P4, or: P1P2P3P5, P1P2P4P5, P1P3P4P5 and P2P3P4P5
Class 3(a)	P1P2P3P5 and P1P3P4P5 or P1P2P4P5 and P2P3P4P5 or: P1P2P5P6, P2P3P5P6, P3P4P5P6, P1P4P5P6
Class 3(b)	P1P2P3P4, or: P1P2P4P5, P1P3P4P5 and P2P3P4P5
Class 4	P1P2P3P5, or: P1P2P4P5 and P2P3P4P5
Class 5	P1P2P3P4

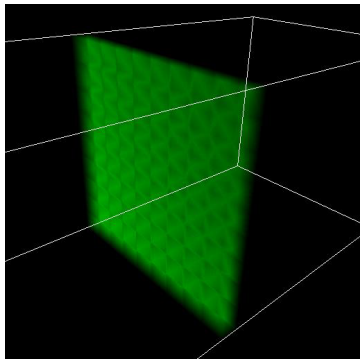


Figure 4. Incorrect rendering result of a constant plate in four dimensions

The key observation in the incorrect opacity is that the length of the projection through time cannot be ignored during the projection of the 4-simplices along the time axis. During the projection, each vertex obtains a Δt value. The value Δt is calculated in a similar way as the calculation of Δz in the projected tetrahedron algorithm, but along the time dimension. The basic idea is that a ray is cast along the time projection and Δt is calculated as the length of the ray that passes through the 4-simplices. Here the projection is from 4D to 3D. So, a vertex in 3D has a non-zero Δt value if it has two different t values along the ray in the t dimension and the Δt is calculated as the difference of the two t values. That means, one vertex has a non-zero Δt if it is overlapped with another vertex in 3D (here, the vertex can be an original projected vertex or a point which is interpolated by other projected vertices after the projection to 3D). Vertices with a non-zero Δt value are illustrated by the red points in Figure 6 for each case of the projected 4-simplices.

In class 5, P4 and P5 are coincident after the projection. Figure 7 gives an example of a 4-simplex which belongs to class 5. Figure 7(a) shows the 4-simplex and figure 7(b) is the projected tetrahedron with non-zero Δt at P4. In class 4, P4 has a non-zero Δt value which is the difference of P4.t and the interpolated t value between P1 and P3. In class 3(a), the new vertex P6 which is the intersection point of the lines P1P3 and P2P4 has a non-zero Δt value which is the difference of two interpolated t values

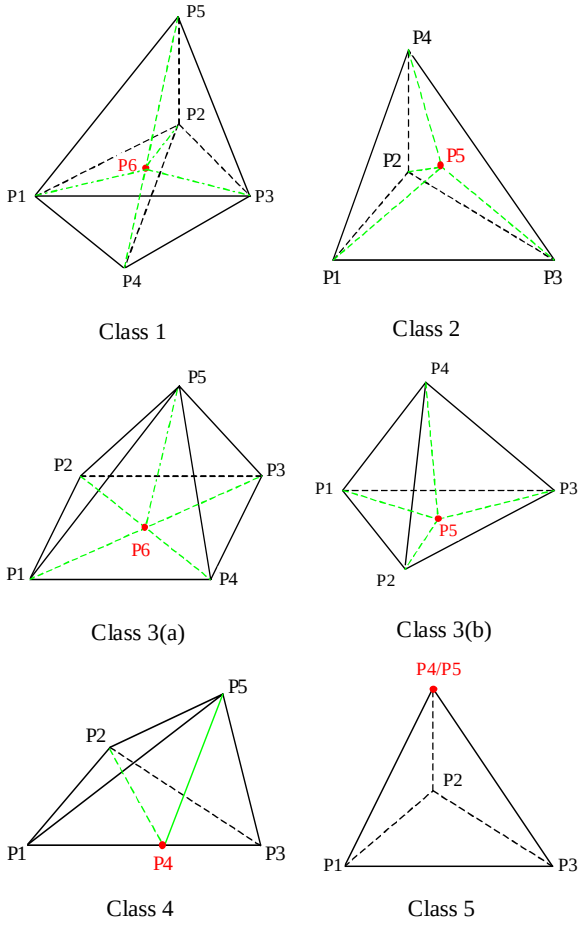


Figure 6. Tetrahedralization of projected 4-simplex

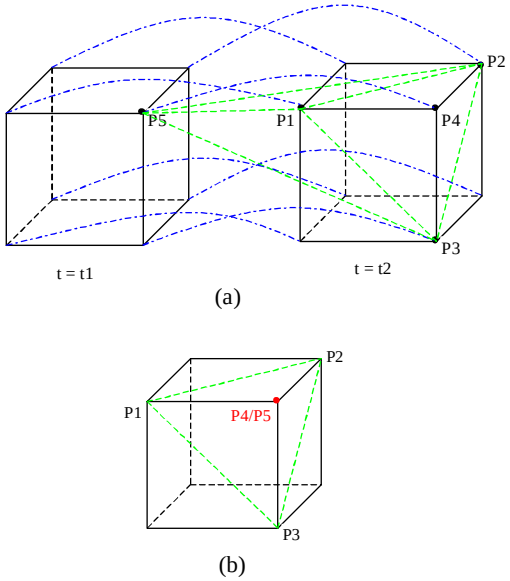


Figure 7. One 4-simplex example of class 5 and its projection

between P1P3 and P2P4. While for class 3(b), the Δt at P5 is non-zero which is equal to the difference of P5.*t* and the *t* value

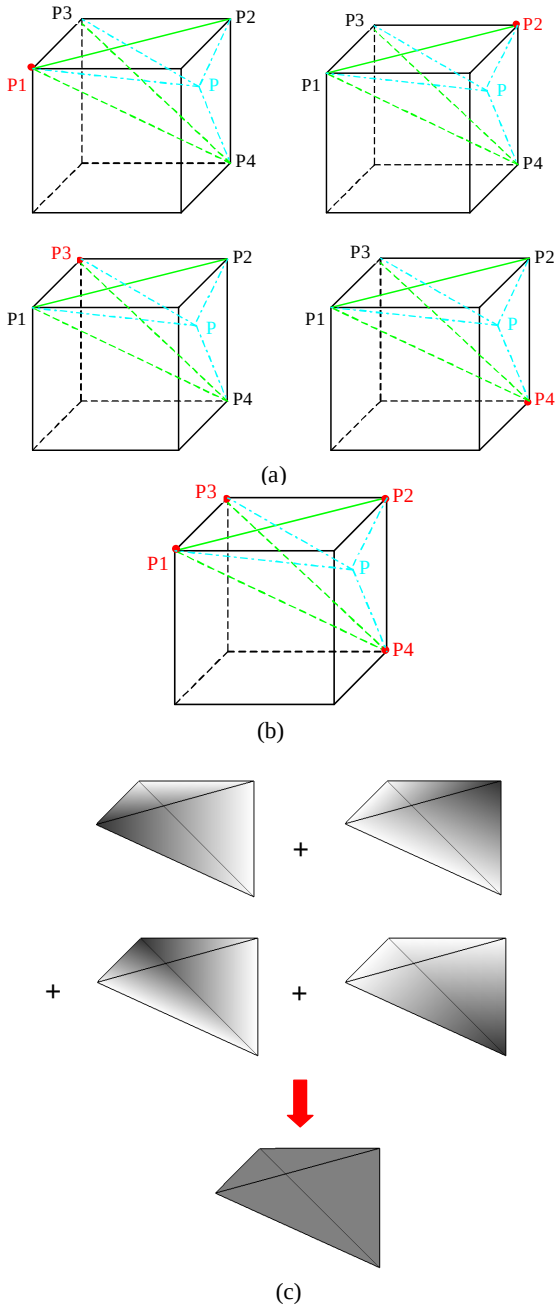


Figure 8. An example of a constant tetrahedron

interpolated inside the triangle P1P2P3. Similarly, for class 2, the Δt value at P5 is the difference of P5.*t* and the interpolated *t* value inside the tetrahedron P1P2P3P4. In class 1, the new vertex P6 is the intersection point of the triangle P1P2P3 and the line P4P5. The Δt value at P6 is the difference of the interpolated *t* value inside P1P2P3 and the interpolated *t* value along the line P4P5.

After determining the vertex with a non-zero Δt for each class, the decomposition of the projected 4-simplices into tetrahedra should make sure that the vertex with a non-zero Δt is one vertex of the decomposed tetrahedra. Now the decomposition becomes a

unique process. The unique decomposition is listed in Table 2 for each class of 4-simplex. For each decomposed tetrahedron, one vertex has a non-zero Δt value and each point inside the tetrahedron has an interpolated Δt value. The Δt distribution inside the tetrahedron also contributes to the final opacity of the rendered tetrahedron.

Table 2. Final decomposition of the projected 4-simplices

Class	Decomposed Tetrahedra
Class 1	P1P2P4P6, P1P3P4P6, P2P3P4P6, P1P2P5P6, P1P3P5P6 and P2P3P5P6
Class 2	P1P2P3P5, P1P2P4P5, P1P3P4P5 and P2P3P4P5
Class 3(a)	P1P2P5P6, P2P3P5P6, P3P4P5P6 and P1P4P5P6
Class 3(b)	P1P2P4P5, P1P3P4P5 and P2P3P4P5
Class 4	P1P2P4P5 and P2P3P4P5
Class 5	P1P2P3P4

Figure 8 shows an example of a constant tetrahedron P1P2P3P4, which is composed of four projected class-5 4-simplices, each with a non-zero Δt value at one vertex (represented as red points). P is any point inside the tetrahedron. By adding the interpolated Δt values from four tetrahedra, every point inside the tetrahedron has a constant Δt (as shown in equation 1 and Figure 8(c)). This is what we expect for a constant tetrahedron which is composed of four projected 4-simplices extracted from a 5D isosurface lookup table.

$$\begin{aligned}\Delta t_p = &\frac{vol_{P2P3P4P}}{vol_{P1P2P3P4}} \times \Delta t_{P1} + \frac{vol_{P1P3P4P}}{vol_{P1P2P3P4}} \times \Delta t_{P2} + \\ &\frac{vol_{P1P2P4P}}{vol_{P1P2P3P4}} \times \Delta t_{P3} + \frac{vol_{P1P2P3P}}{vol_{P1P2P3P4}} \times \Delta t_{P3} \\ &= \Delta t \quad (if \ \ \Delta t_{P1} = \Delta t_{P2} = \Delta t_{P3} = \Delta t_{P4} = \Delta t)\end{aligned} \tag{1}$$

Considering the contribution of the Δt on the opacity, the data in Figure 4 is rendered correctly as in Figure 9.

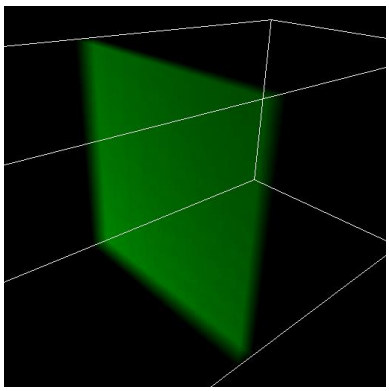


Figure 9. Rendering result of a constant plate

4.2 Projection of 3-simplices to image space

We use an implementation of the Projected Tetrahedron algorithm from Shirley and Tuchman [19] to render the projected tetrahedra from the 4-simplices. The algorithm approximates a tetrahedron using one to four triangles depending on the screen projection of the tetrahedron's vertices. We implement the PT algorithm using a vertex program in programmable graphics hardware [28].

Compared to the projection of the normal tetrahedra, there is one difference in the rendering of the time-varying interval volumes: the tetrahedra here have a non-constant Δt distribution from the projection along the time axis. We can understand this Δt distribution to be a property distribution inside tetrahedra, like density distribution. Therefore, when we calculate the opacity of the projected triangles, we should consider both the contribution of the Δt for the projection along the time axis and the contribution of the Δz for the projection along the z-axis. Since the zero-thickness vertices in PT algorithm do not necessarily have zero Δt thickness, and the vertex with non-zero thickness in PT algorithm may have zero thickness of Δt, so we cannot directly multiply the Δt and the Δz at each vertex and then interpolate it inside the projected triangles. Actually, the bi-variant function should be evaluated at each pixel. That means, we should multiply the interpolated Δt and the interpolated Δz for each pixel inside the projected triangles.

The opacity along a ray is represented as $\alpha = 1 - e^{-\tau \cdot l}$. Here, τ is the extinction coefficient and l is the thickness of the ray inside the tetrahedron. For the rendering of the 4D interval volumes, both the Δt and the Δz contribute to the thickness: $l = \Delta t \cdot \Delta z$. We develop a modified implementation of the Shirley and Tuchman algorithm using the vertex and fragment programs to consider both the contributions of the Δt and the Δz. In the vertex program, we calculate the Δt and Δz for each vertex of the projected triangles, then their contributions to the opacity are multiplied in the fragment program for each pixel.

4.3 Visualization techniques of time-varying interval volumes

In this section, we build upon our work of the computation and projection of the time-varying interval volumes to come up with some visualization techniques for effective visualization of the time-varying volumetric data sets. As discussed in previous sections, the tetrahedra for time-varying interval volumes have Δt distribution from the projection along the time axis and they overlap themselves in 3D space. This causes some occlusion and compositing problems. In this section, we will figure out the suitable rendering techniques for the time-varying interval volumes.

4.3.1 Direct rendering of the time-varying interval volumes with color encoded by the time

We can render the time-varying interval volumes directly from the extracted 4-simplices, using the projection methods as discussed in sections 4.1 and 4.2. Since the time-varying interval volumes actually project to the same three-dimensional space (i.e., it is a self-intersecting volume), no accurate sorting is possible. In this paper, we sort the tetrahedra first by position and then by time. And an additive compositing operator is used to blend the 4-simplices into the image. The color of the vertex is encoded using the time value. For the overlapped vertices (as shown with the red points in Figure 6), the time value is calculated as the average *t* values of the two overlapped vertices.

Figure 10 is an example of the direct volume rendering result of a simple test function comprised of a linear ramp in time. Here, the color at *t=t1* is green, and the color at *t=t2* is red. The color between *t1* and *t2* is encoded between green and red. From this figure, we can see the transition from the green, to the yellow, and to the red as the field moves over time.

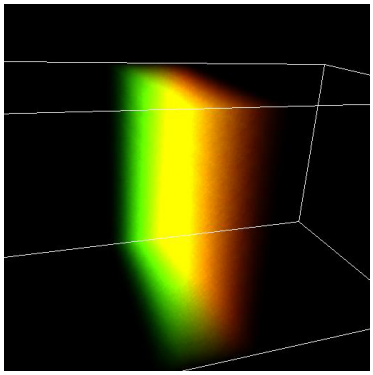


Figure 10. Direct rendering result of a time-varying interval volume

4.3.2 Time-varying interval volumes with the highlighted surface boundaries

Similar to the interval volume with embedded boundary surfaces in [7], we can embed the boundary isosurfaces into time-varying interval volumes to highlight interior features. The boundary surfaces are extracted during the construction of the time-varying interval volumes without extra computation cost, simply by checking if the vertices are on a boundary or not.

For time-varying interval volumes, there are two types of boundaries: volumetric boundaries and surface boundaries. In this subsection, we first consider the surface boundaries. Given a time-varying interval volume defined by two isovalues α and β, and two time steps $t1$ and $t2$, there are four boundary isosurfaces at: (a) $t=t1$ and $f(x,y,z,t)=\alpha$, (b) $t=t1$ and $f(x,y,z,t)=\beta$, (c) $t=t2$ and $f(x,y,z,t)=\alpha$, (d) $t=t2$ and $f(x,y,z,t)=\beta$. Since these surfaces are the boundary of the tetrehedra which compose the time-varying interval volume, these boundary surfaces are rendered together with the 4D interval volume. The four isosurface boundaries are illustrated in Figure 11 in the above order (a) to (d) from left to right. From the figure, we can see how the isosurfaces change with time and with value.

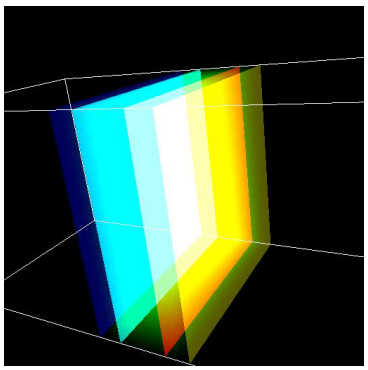
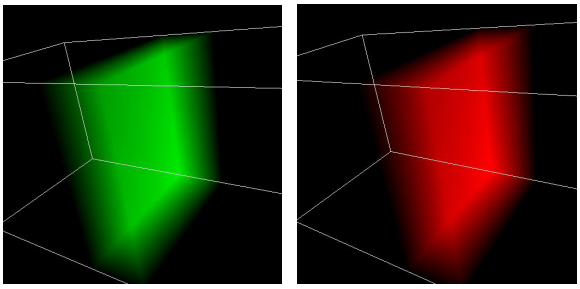


Figure 11. Time-varying interval volume with four isosurfaces highlighted (from left to right: (a) $t=t1$ and $f(x,y,z,t)=\alpha$, (b) $t=t1$ and $f(x,y,z,t)=\beta$, (c) $t=t2$ and $f(x,y,z,t)=\alpha$, (d) $t=t2$ and $f(x,y,z,t)=\beta$)

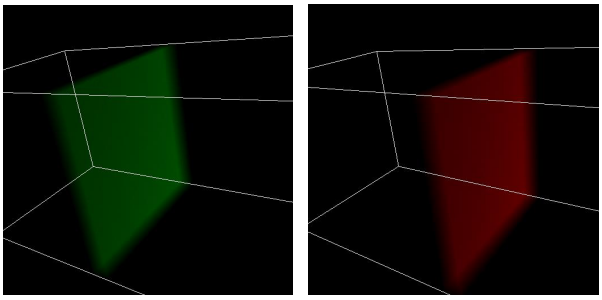
4.3.3 Rendering of volumetric boundaries

The volumetric boundaries are discussed in this section. There are also four kinds of volumetric boundaries for a time-varying interval volume defined by two isovalues α and β, and two time steps $t1$ and $t2$: (a) time-varying isosurfaces at $f(x,y,z,t)=\alpha$ and $t1 \le t \le t2$, (b) time-varying isosurfaces at $f(x,y,z,t)=\beta$ and $t1 \le t \le t2$, (c) interval volumes at $\alpha \le f(x,y,z,t) \le \beta$ and $t=t1$, and (d) interval volumes at $\alpha \le f(x,y,z,t) \le \beta$ and $t=t2$. The four volumetric boundaries are shown in Figures 12(a), 12(b), 13(a) and 13(b), respectively.



(a) (b)

Figure 12. Time-varying isosurfaces at: (a) $f(x,y,z,t)=\alpha$ and $t1 \le t \le t2$, and (b) $f(x,y,z,t)=\beta$ and $t1 \le t \le t2$.



(a) (b)

Figure 13. Interval volumes at: (a) $\alpha \le f(x,y,z,t) \le \beta$ and $t=t1$, and (b) $\alpha \le f(x,y,z,t) \le \beta$ and $t=t2$.

The volumetric boundaries are rendered using the normal projected tetrahedron algorithm, without considering the contribution of Δt on the opacity. Also, a constant color is assigned to each volumetric boundary. Here, a technique similar to Maximum Intensity Projection (MIP) can be employed to sort the interval volumes, not according to the viewing rays, but according to their priorities (the time here) to bring an important interval volume at a specific time step to the forefront. We will show some images using MIP in next section.

5 RESULTS

We apply the rendering and visualization techniques of the time-varying interval volumes explained in section 4 to several datasets. Figure 14 shows time-varying interval volumes of a vortex dataset. The color is encoded using time t: green at $t=t1$, red at $t=t2$, and yellow for overlapped regions between $t=t1$ and $t=t2$. From this figure, we can see how the interval volumes move over time. We can also notice that some new components are generated over time, such as the purely red one in Figure 14. Figure 15 is the interval volumes of the vortex dataset for three time steps. The color mapping with time t is in the following way: blue at $t=t1$, green at $t=t2$, red at $t=t3$, cyan for overlapped regions between $t=t1$ and $t=t2$, yellow between $t=t2$ and $t=t3$. So, for the overlapped region among $t=t1$, $t=t2$ *and* $t=t3$, the color is white using the additive compositing operator. In this figure, areas where contours are appearing over time are predominantly red, while areas that faded over time are predominantly blue. Areas which maintain a high isovalue over time appear white. Figure 16 shows time-varying interval volumes for the NASA Tapered

Cylinder dataset. This dataset is a curvilinear grid in PLOT3D format. Similarly, this figure shows the movement of the interval volumes with the time.

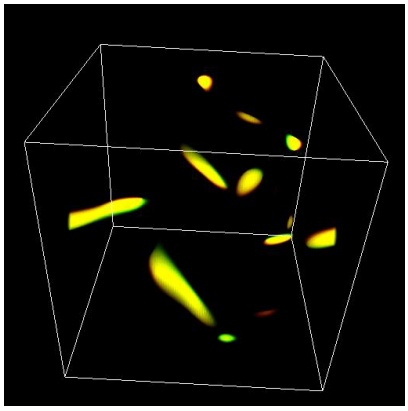


Figure 14. Time-varying interval volumes for vortex dataset (two time steps)

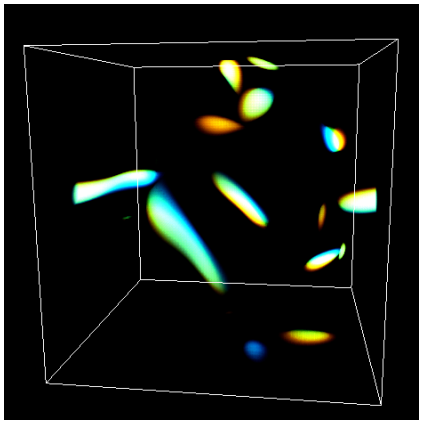


Figure 15. Time-varying interval volumes for vortex dataset (three time steps)

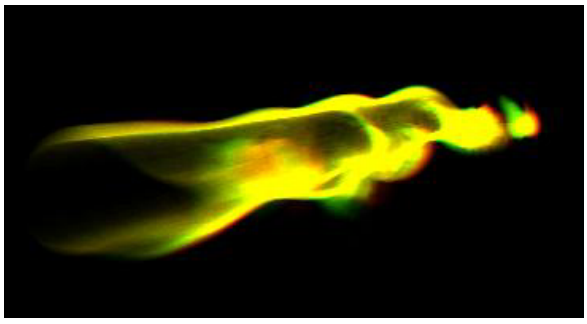


Figure 16. Time-varying interval volumes for the NASA Tapered Cylinder dataset

By rendering two interval volumes at $t=t1$ and $t=t2$ extracted from the volumetric boundary into one image using the MIP technique discussed in section 4.3.3, we can see how the interval volumes move with the time steps. In this way, an important interval volume at a specific time step is brought to the forefront, preventing being occluded by the interval volumes at other time steps. Figures 17 and 18 show two interval volumes at *t1* and *t2* for the vortex dataset and the Tapered Cylinder dataset in one view using the MIP technique. Here, yellow color represents the interval volume at *t1*, and red color is for *t2*. In the two figures, the interval volume at *t2* is given higher priority.

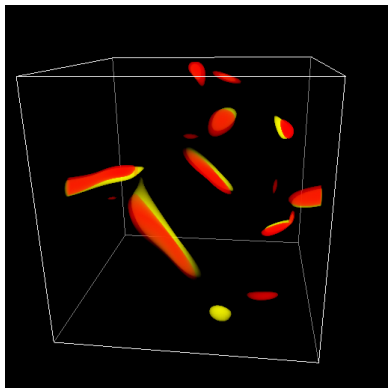


Figure 17. Two interval volumes at *t1* and *t2* for the vortex dataset are rendered using MIP

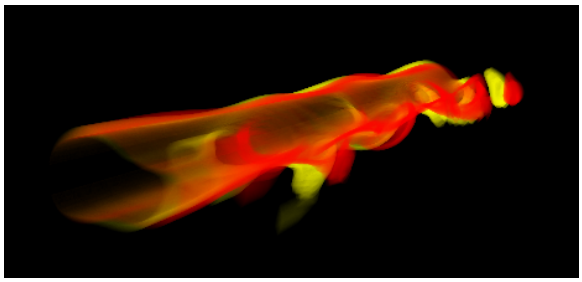


Figure 18. Two interval volumes at *t1* and *t2* for the Tapered Cylinder dataset are rendered using MIP

All the results presented in this paper have been generated using a PC with a QuadroFX 3000 graphics card and a Pentium IV 3.4 GHz processor. The interval volume computation time and the volume rendering time for the datasets are listed in Table 3.

Table 3. 4D interval volume computation and rendering performance

Data set	4D interval volume construction and decompo-sition time	Number of 4-simplices	Number of tetrahedra (with volume)	Rendering time (linear color)
Test function (2x20x10x 10)	75ms	9,720	27,054	30ms
Vortex dataset (2x128x 128x128)	12.2s	319,304	882,044	970ms
Tapered Cylinder (curvilinear 2x64x64 x32)	18.5s	349,624	941,098	1,030ms
Vortex dataset (3x128x 128x128	22.5s	654,846	1,807,460	1,980ms

In the Table 3, the 4D interval volume construction and decomposition time includes the time to calculate the entries of the isosurface lookup table, the time to construct 4-simplices and the time to decompose 4-simplices to tetrahedra. Due to the decomposition of 4-simplices to 3-simplices and the overlapping copies of the 3-simplices in 3D space, there are more tetrahedra for time-varying interval volumes.

6 Conclusions and Future Work

In this paper, we have presented an algorithm for computing time-varying interval volumes in structured and unstructured grids using a fast isosurface extraction algorithm. And we have explained the rendering methods of the 4-simplices by projecting and decomposing the 4-simplices to 3-simplices, and using a modified hardware implemented projected tetrahedron method. In this way, we can render the time-varying interval volumes which integrate multiple time steps into a single view and we can see the movement of the interval volumes over time in one view. Different visualization techniques have been demonstrated for the visualization of the time-varying structured and unstructured data sets.

The current rendering technique uses the hardware implemented projected tetrahedron method [28]. We can use the new PT implementation presented in [15] to improve the quality of the images and we can consider to speed up the projection from 4D to 3D by taking advantage of the modern graphics hardware. Also, the current algorithm can be augmented with feature detection techniques to aid the user in identifying useful/interesting intervals in the field. We also want to extend the concept of constructive solid geometry for multi-attribute data sets to do arbitrary operations like intersections, unions and subtractions.

Acknowledgements

The Tapered Cylinder data set is from NASA's online data set repository. Part of this work was supported by NSF award #ACI-0222903.

References

[1] ALBERTELLI, G., AND R. A. CRAWFIS, *Efficient subdivision of finite-element datasets into consistent tetrahedra*, in Proceedings of IEEE Visualization '97, p.213-219, October 18-24, 1997, Phoenix, Arizona.

[2] BAJAJ, C., V. PASCUCCI, G. RABBIOLO, AND D. SCHIKORE, Hypervolume Visualization: A Challenge in Simplicity, in IEEE Volume Visualization 1998 Symposium, pp. 95-102.

[3] BANKS, D., AND S. LINTON, Counting Cases in Marching Cubes: Toward a Generic Algorithm for Producing Substitopes, In Proceedings of IEEE Visualization 2003, pp. 51-58.

[4] BANKS, D., S. LINTON, AND P. STOCKMEYER, Counting Cases in Substitope Algorithms, IEEE Transactions on Visualization and Computer Graphics, July/August, 2004, Vol. 10, No. 4, pp. 371-384.

[5] BHANIRAMKA, P., R. WENGER, AND R. CRAWFIS, *Isosurfacing In Higher Dimensions*, in Proceedings of IEEE Visualization 2000, Ertl, Hamann, Varshney, Ed., IEEE Visualization Proceedings, 2000, 15-22.

[6] BHANIRAMKA, P., R. WENGER, AND R. CRAWFIS, *Isosurface Construction in any dimension using convex hulls*, IEEE Transactions on Visualization and Computer Graphics, March/April, 2004, Vol. 10, No. 2, pp 130-141.

[7] BHANIRAMKA, P., C. ZHANG, D. XUE, R. CRAWFIS, AND R. WENGER, *Volume Interval Segmentation and Rendering*, in IEEE Volume Visualization 2004 Symposium, pp. 55-62.

[8] FUJISHIRO, I., Y. MAEDA, AND H. SATO**,** *Interval volume: a solid fitting technique for volumetric data display and analysis*, in IEEE Visualization '95, Atlanta, GA, 1995.

[9] FUJISHIRO, I., Y. MAEDA, H. SATO AND Y. TAKESHIMA, *Volumetric data exploration using interval volume*, in IEEE Transactions on Visualization and Computer Graphics, 2 (June 1996).

[10] GUO, B. *Interval Set: A Volume Rendering Technique Generalizing Isosurface Extraction*, in Proceedings of IEEE Visualization '95, Atlanta, GA.

[11] HANSON, A., AND P. HENG, *Four-Dimensional Views of 3D Scalar Fields*, in Proceedings of IEEE Visualization 1992, pp. 84-91.

[12] HANSON, A., AND P. HENG, *Illuminating the Fourth Dimension*, IEEE Computer Graphics and Applications, Vol. 2, No. 4, pp. 54-62.

[13] HANSON, A., AND R. CROSS, *Interactive Visualization Methods for Four Dimensions*, in Proceedings of IEEE Visualization 1993, pp. 196-203.

[14] JI, G., H. SHEN, AND R. WENGER, *Volume Tracking using Higher Dimensional Isosurfacing*, In Proceedings of IEEE Visualization 2003, pp. 209-216.

[15] KRAUS, M., W. QIAO, AND D. EBERT, *Projecting Tetrahedra without Rendering Artifacts*, in Proceedings of IEEE Visualization 2004, pp. 27-34.

[16] LORENSEN, W. E., AND H. E. CLINE, *Marching cubes: A high resolution 3d surface construction algorithm*, in M. C. Stone, ed., *Computer graphics*, 1987, Anaheim, California, July 1987, pp. 163-169.

[17] MAX, N. *Consistent Subdivision of Convex Polyhedra into Tetrahedra*, in Journal of Graphics Tools, 6 (3), 29-36, 2002.

[18] NIELSON, G. M., AND J. SUNG, *Interval volume tetrahedrization*, in R. Y. a. H. Hagen, ed., IEEE Visualization '97, IEEE, November 1997, pp. 221-228.

[19] SHIRLEY, P. AND A. TUCHMAN, *A polygonal approximation to direct scalar volume rendering*, in Volume Visualization Workshop, 1990, pp. 63-70.

[20] The Ohio State University. Isotable generation software. http://www.cse.ohio-state.edu/graphics/isotable.

[21] WEILER, M., M. KRAUS, AND T. ERTL, *Hardware Based View-independent Cell Projection*, in Symposium on Volume Visualization, 2002, Boston, MA.

[22] WEILER, M., M. KRAUS, M. MERZ, AND T. ERTL, *Hardware-Based Ray Casting for Tetrahedral Meshes*, in Proceedings of IEEE Visualization 2003, pp. 333-340.

[23] WILLIAMS, P. *Visibility Ordering of Meshed Polyhedra*, in ACM Transactions on Graphics, 11 (4), 103-126, April 1992.

[24] WILLIAMS, P., *A Volume Density Optical Model*, in IEEE Volume Visualization Symposium, '92, 61-68

[25] WILLIAMS, P., N. MAX, C. M. STEIN, *A High Accuracy Volume Renderer for Unstructured Data*, in IEEE Transactions on Visualization and Computer Graphics 4(1): 37-54 (1998).

[26] WOODRING, J., AND H. SHEN, *Chronovolumes: A Direct Rendering Technique for Visualizing Time-Varying Data*, In Proceedings of 2003 International Workshop on Volume Graphics.

[27] WOODRING, J., C. WANG, AND H. SHEN, *High Dimensional Direct-Rendering of Time-Varying Volumetric Data*, In Proceedings of IEEE Visualization 2003, pp. 417-424.

[28] WYLIE, B., K. MORELAND, L. A. FISK, AND P. CROSSNO, *Tetrahedral projection using Vertex Shaders*, in Symposium on Volume Visualization, 2002, Boston, MA