High Dimensional Direct Rendering of Time-Varying Volumetric Data

Jonathan Woodring* Chaoli Wang† Han-Wei Shen‡

The Ohio State University

Abstract

We present an alternative method for viewing time-varying volumetric data. We consider such data as a four-dimensional data field, rather than considering space and time as separate entities. If we treat the data in this manner, we can apply high dimensional slicing and projection techniques to generate an image hyperplane. The user is provided with an intuitive user interface to specify arbitrary hyperplanes in 4D, which can be displayed with standard volume rendering techniques. From the volume specification, we are able to extract arbitrary hyperslices, combine slices together into a hyperprojection volume, or apply a 4D raycasting method to generate the same results. In combination with appropriate integration operators and transfer functions, we are able to extract and present different space-time features to the user.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

Keywords: time-varying data, hyperslice, hyperprojection, integration operator, transfer function, raycasting, volume rendering

(a) (b)

Figure 1: Depicting time evolution in photography and art. (a) *Jumping off a Chair*, Etienne-Jules Marey. (b) *Nude Descending a Staircase (No. 2)*, Marcel Duchamp.

1 Introduction

When analyzing a time sequential event, we are likely to think of it as single frames of events in our mind's eye or how the event progresses in time as we imagine it. Artists, photographers, and the early pioneers of cinematography explored the co-mingling of space and time. Etienne-Jules Marey's chronophotographic image of a figure jumping off a chair, seen in Figure 1(a), and Marcel Duchamp's painting, *Nude Descending a Staircase (No. 2)*, seen in Figure 1(b), are two such examples. Culminating in the late 19th and early 20th century, works like these were a result from the aggregation of influence from the birth of cinema, the works of Marey, Muybridge and Eakins, Futurism and Cubism art styles, and the redefinition of space and time by scientists and philosophers. From this imagery, we are able understand the motion and flow of the figures moving over time.

Traditionally, we have two methods of visualizing time series data. We can take a snapshot of the data, by taking a particular time step in a time series and visualizing that volume. It is just a single frame from a sequence of images, and like a 2D slice from a 3D volume, while useful, it is out of context from the entire time sequence. We can also generate an animation from the time series data. This is also useful, but we have to rely on our memory and cognitive abilities to tie together spatio-temporal relationships. Inspired by the works of Duchamp and his peers, we seek to visualize time-varying volumetric data in a different manner.

What we provide is an alternative method to viewing time evolving data. Rather than considering space and time as separate components, the data are treated as a four-dimensional data field. In this manner, we can do high dimensional direct rendering of the data. In the first section following the related work, we will discuss our method to perform projection in four dimensions. This is conceptually broken into two parts, the specification of a image hyperplane and the parallel projection of hyperplanes. When we refer to the image hyperplane, we are referring to a 3D hyperplane in 4D space which we project to, similar to how we project 3D space to a 2D plane to visualize volumes. The result of our technique generates a volume that is the projection of hyperplanes along a 4D projection vector, which can be rendered using traditional volume rendering techniques. Then, we will describe different categories of hyperplanes and how to interpret the projection of each type of the hyperplanes. The interpretation of the projection is dependent upon the integration operator and transfer functions that we use in 4D projection. From these operators, we are able to generate a 4D hyperprojected image volume that exposes different space-time relationships to the user.

2 Related Work

Previous work on time-varying visualization that relates to our work primarily focuses on visualization of higher dimensional objects, time-varying feature tracking and transfer function design.

*e-mail:woodring@cis.ohio-state.edu

†e-mail:wangcha@cis.ohio-state.edu

‡e-mail:hwshen@cis.ohio-state.edu

Already in the 19th century, mathematicians had fashioned models to show the sequence of hypercube slices in various directions [Banchoff 1990]. Nowadays, interactive computer graphics puts us into direct visual contact with slices of 4D cubes. It is up to us to learn how to interpret these images, and to try to overcome the limitation of our own 3D perspective[Cohen 2000]. Over the past decade, many approaches have been proposed to visualize higher dimensional objects. Hanson, Heng and Cross [Hanson and Heng 1992a; Hanson and Heng 1992b; Hanson and Cross 1993] 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 field could be viewed as 3D terrains. A 4D illumination model was developed where 3D Phong lighting model was extended to 4D, with tetrahedra as the basic rendering primitives. Special care was taken to enhance objects with renderable properties so that they are renderable in the embedded dimension.

The *Hyperslice* approach [van Wijk and van Liere 1993] proposed by van Wijk and van Liere uses a matrix of orthogonal 2D slices as the basic visual representation of a multi-dimensional function. Although this approach enables the user to view multi-dimensional space in a simple and intuitive way, it could be difficult to reconstruct a complete mental image of the data from the separate multiple slices for higher dimensions because at most two dimensions are considered for a single slice. Bajaj *et al.*[Bajaj et al. 1998] developed an interface that provides "global views" of scalar fields independent of the dimension of their embedded space and generalized the object space splatting technique into a hyper-volume splatting method. Texture mapping hardware is utilized to directly render *n*-dimensional views of the global scalar field. In essence, visualization of high dimensional objects was the primary interest for those methods, therefore no explicit temporal feature tracking was attempted.

To track time-varying features, researchers have proposed various methods to establish correspondence between data in different time steps. Silver and Wang used spatial overlap criteria [Silver and Wang 1997] to track time-varying interval volumes evolving in time for structured and unstructured data. Important temporal events such as bifurcation can also be detected [Samtaney et al. 1994]. Banks and Singer [Bank and Singer 1995] used a predictor-corrector method to reconstruct and track vortex tubes from turbulent time-dependent flows. Reinders *et al.*[Reinders et al. 2001] designed the event graph viewer in a linked combination with a 3D feature viewer to assist visualization and exploration of time-dependent data.

Research on transfer function design has been mostly focused on time-invariant volume data. Kindlmann *et al.*[Kindlmann and Durkin 1998] proposed a semi-automatic algorithm to detect the material boundaries based on first and second derivatives of the scalar data. A function that maps from data values to the distances to the boundaries is used to assist opacity assignments. Kniss *et al.*[Kniss et al. 2001; Kniss et al. 2002] later followed up the work in [Kindlmann and Durkin 1998] with an intuitive user interface and introduced the concept of dual-domain interaction. For time-varying data, Jankun-Kelly and Ma [Jankun-Kelly and Ma 2001] proposed a method to reduce the number of transfer functions across the time sequence by merging the coherent segments. Multiple transfer functions for a time-varying data are not frequently used, but could be effective for certain scenarios.

The VideoCube [Microsoft Research Graphics Group 2000] application developed at Microsoft allows one to load an AVI file as a volume, and play back the movie sampling space and time in different ways. A single cutting plane is provided for interactively viewing single space-time slices of the video. A more recent work on *Chronovolumes* by Woodring and Shen [Woodring and Shen 2003] proposed a way that integrates time-varying volume data through time and produces a single view that captures the essence of multiple time steps in a sequence. In this paper, we generalize their idea by treating time-varying data as a four-dimensional data field and allow users to integrate the data along arbitrary directions in space and time using high dimensional slicing and projection techniques.

3 Projection in Four Dimensions

Our goal is to understand spatio-temporal characteristics of time-varying volumetric data using 4D projection. To perform projection in four dimensions, we approach the concept by specification of hyperplanes in 4D space and projection of these hyperplanes. The concept of slicing volumes can be readily extended to 4D fields, which is sometimes called *hyperslicing*. Though there is a conceptual similarity between 3D slicing and 4D hyperslicing, it is less intuitive to specify the location and orientation of a hyperslice in 4D space. Given that, we have devised an improved method for interactive specification of arbitrary hyperslices.

Based on our hyperslicing results, we go one step further by performing projection of the hyperplanes. We will discuss the different categories of hyperslices, and the projection that is performed. How we generate interpretable images is dependent upon the integration operators and transfer functions that we use to project in 4D space. With the operators that we have designed, we are able to generate images that are easy to interpret and enable users to understand the spatio-temporal characteristics of the time-varying volumetric data.

3.1 Arbitrary Hyperslicing Specification

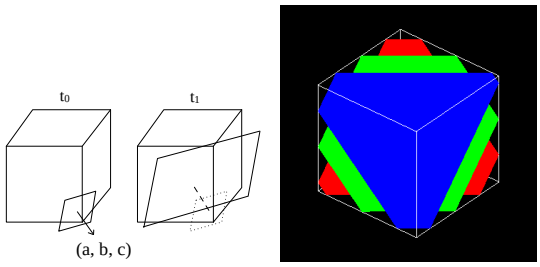
A hyperplane is a $n-1$ subspace of a *n*-dimensional space. Given that we are working with 4D spaces, space and time, hereafter when we refer to a hyperplane, we are speaking of a 3D subspace of a 4D space. A 4D hyperplane can be defined by equation in the *coordinate* form:

$$ax + by + cz + dt + e = 0 \tag{1}$$

Although it is possible for the user to provide the numerical values of the coefficients (a,b,c,d,e) for the hyperplane in Equation 1, compared to 3D slicing, specifying a hyperplane to slice a 4D volume is much more difficult due to our limited ability to visualize objects in four dimensions. There are other forms for the hyperplane, such as the *vector* form $n \cdot (P - P_0) = 0$, which allows the user to specify position and orientation, but it is still difficult for the user to conceptualize what is being sliced. In the following, we describe our approach to address this issue with the coordinate form hyperplane representation.

In Equation 1, if we fix the time, for instance let $t = \tau$, then the hyperplane equation reduces to $ax + by + cz + C = 0$, where $C = d\tau + e$ and is constant. This is a 2D plane in 3D space at a particular instance in time, which can be easily visualized. Based on this idea, given a hyperplane equation, we can compute as well as visualize the hyperslice by computing the intersection of the hyperplane with the underlying time-varying volume at different time steps, and then stack up the resulting 2D slices to form a volume. This is the result of hyperslicing. Two of the local axes of the hyperslice volume span a plane that is orthogonal to the vector (a,b,c), and the other axis is parallel to the time axis t. Since the hyperslice is a volume, we can visualize this volume using regular volume rendering methods in 3D space.

The above algorithm for computing hyperslice provides a intuitive way to help the user specify and visualize a hyperplane, as seen in Figure 2. We can separate the process of specifying a hyperplane into two steps. In the first step, the user specifies the orientation of the intersection plane between the hyperslice and the underlying



(a) (b)

Figure 2: (a) shows the conceptual notion presented to the user on how to specify a hyperslice. The user determines (a,b,c) by setting the orientation for a 2D slice of a time step. d and e is determined by the offset of each slice in space over time. (b) shows the interface in the software. Three time steps are shown, where blue is the slice taken at the first time step, green is the second, and red is the third. All the slices through time are stacked together to form the hyperslice.

volume in spatial domain. In the second step, the user specifies how the intersection plane will move in time along its normal direction. In Equation 1, the first three coefficients, a,b and c, determine the orientation of the intersection plane in space. The coefficients d and e, as well as the value of the time variable t, decide the offsets of the intersection plane at different time steps. For instance, when $t = 0$, the intersection of the hyperplane with the volume has the form $ax+by+cz+e = 0$. And when $t = 1$, the intersection of the hyperslice with the volume becomes $ax+by+cz+d+e = 0$, which indicates that the slicing plane moves along the plane normal (a,b,c) in spatial domain by $d/\sqrt{a^2+b^2+c^2}$ for every time step, assuming a,b, and c are not all equal to zero. A hyperplane can be illustrated in this way that shows the spatial boundaries by polygons, and the temporal boundaries by colors.

3.2 Hyperplane Projection

When visualizing a time-varying volume as 4D space, we extend parallel projection in 3D to 4D projection. In 3D parallel projection, the 2D image plane moves along its normal direction, subsampling and integrating the underlying 3D volume. In four dimensions, we have a 3D image plane that moves along its normal direction, subsampling and integrating 4D space. Thus, the result of the 4D projection is a volume where each voxel in the resulting volume is an integration of space-time samples along the hyperplane normal. To visualize this *hyperprojection* volume, standard volume rendering techniques can be used to project the volume down to a 2D image plane.

The idea of 4D parallel projection can be implemented based on the hyperslicing algorithm described in the previous section. Assuming the image volume is initially set by Equation 1, we can compute a sequence of hyperslices by moving the image volume along its normal direction (a,b,c,d) in 4D space. This is equivalent to updating the hyperplane equation by adding a small constant at every sampling step. That is, for the ith sampling hyperplane, $S_i = ax+by+cz+dt+e+\Delta ei$. With the sequence of properly aligned hyperslices, we can integrate the voxels from the hyperslice sequence to compute the projection result. To ensure a proper sampling rate, Δe should be small enough so that the distance between two consecutive subsamples is less than or equal to one voxel in length. In four dimensions, the diagonal of a hypercube with sides of length 1, corresponding to an even spacing of samples in 4D space, has a length of 2. Half this distance is 1, so the proper sampling rate to get influence from all voxels would be $\Delta e = 1$, assuming that (a,b,c,d) is normalized.

4 Interpretation of 4D Projection

One challenge that remains is the practical use of the projection of 4D space and how to interpret the resulting images. Half of the interpretation depends upon the form of the image hyperplane equation. The other half of the interpretation depends on what integration method is used to project four dimensions down to three dimensions. Useful information promoting the understanding of spatial and temporal behaviors of the data can be derived if suitable integration operators are used to integrate when hyperprojecting. In the following, we first discuss how to interpret the meaning of 4D volume projection based on the hyperplane equation, and then describe the integration operators. We have used several data sets from computational fluid dynamics to show examples of our results. These include the jet, shockwave, delta wing, and vortex time-varying data sets, using 10 time steps in our examples.

4.1 Hyperplane Interpretation

We classify a hyperplane in Equation 1 into three different families. Each of the families can be identified based on the values of the plane coefficients, and has a unique meaning when projecting the 4D space onto the 3D hyperplane:

(1) $d = 0$, i.e., the hyperplane has the form $ax+by+cz+e = 0$;

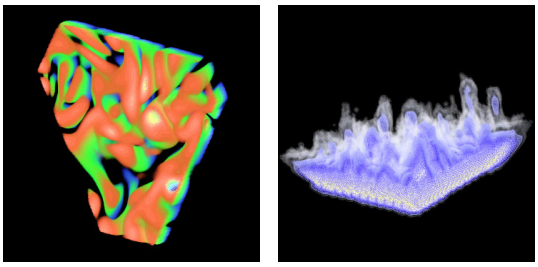
(2) $d \neq 0$, $a = b = c = 0$, i.e., the hyperplane has the form $dt+e = 0$;

(3) $d \neq 0$, $a \vee b \vee c \neq 0$.

(1) d = 0. When $d = 0$, the hyperplane equation reduces to $ax+by+cz+e = 0$, which means that the hyperplane will intersect with the volume at exact the same location for all the time steps. This type of hyperslice will create a volume that contains 2D spatial planes for a fixed position, for every time step. If we visualize a single hyperslice, it will create the effect of several 2D planes stacked along a time axis as seen in Figure 3(a).

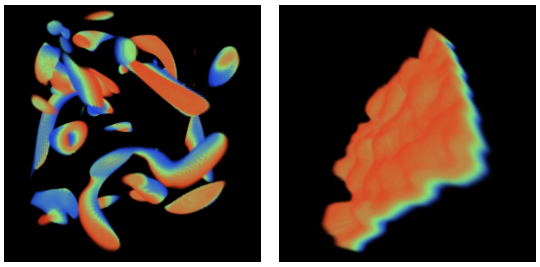
When hyperprojecting along the normal direction $(a,b,c,0)$, since the time increment is zero, each voxel of the hyperslice will accumulate subsamples along the spatial vector (a,b,c). The resulting hyperprojection of the 4D field a volume consisting t slices, where each slice S_τ is a parallel projected volume rendered image along the viewing direction (a,b,c) for the volume at time step τ. The net effect is equal to stacking all the images from rendering a time-varying sequence at one view angle, (a,b,c), for all time steps into a volume. When viewing these hyperprojection volumes, we can observe time evolution along a particular viewing vector, that way the user can observe the change in a viewing profile. One other use is to pass a 2D cutting plane that is perpendicular to the time axis in the image hyperplane. This would give users the ability to animate through time from a particular view by moving the cutting plane back and forward through time.

(2) d ≠ 0, a = b = c = 0. In this case, the hyperplane becomes $dt+e = 0$, or $t = -e/d$. Visualizing the hyperslice of this form is equivalent to visualizing the time-varying sequence at the time step $-e/d$. Without loss of generality, we assume $-e/d$ is a positive number. When hyperprojecting along the normal direction $(0,0,0,1)$, this is equivalent to casting a ray from each voxel in the image hyperplane into the time direction. Every voxel in the final volume after projection represents a fixed location in the original 3D data space, but the integration of that position over time. Conceptually, the result is taking several volumes over time and combining them into one volume. The original spatial depth is preserved, so that when the volume is rendered using 3D visualization techniques, voxels farther from the eye are occluded by ones that are closer. We have found that projecting the 4D space in this way is particularly useful for understanding the space and time behavior



(a) (b)

Figure 3: The vortex data set with hyperplane equation set such that the corresponding rendered hyperslice in (a) shows a 2D data slice in spatial domain changing in time. Color assignment is based on time, with blue being the first and red the last time step. A rendered hyperslice of the concentrate data set (b) shows the data slices evolving with space and time. By setting the normal of the hyperplane orthogonal the movement of the data in 4D space, the growth of the acid component is captured.



(a) (b)

Figure 4: Using alpha composition with the hyperprojection normal of $(0,0,0,1)$, where we project through time. The interval volume starts at blue, moves over time to green, and eventually ends at red. (a) uses the vortex data set, while (b) uses the shockwave data set.

of time-varying data. More discussion and interpretation of projecting these hyperplanes will be given in the next section.

(3) $\mathbf{d} \neq \mathbf{0}$, $\mathbf{a} \vee \mathbf{b} \vee \mathbf{c} \neq \mathbf{0}$. If $d \neq 0$ and one of the other hyperplane normal components is not zero, then space is "shifted" by time. Given the previous two hyperplane families, the oblique hyperplane is the least useful and least intuitive out of the three families. Projection of these hyperplanes proves to be hard to interpret as well. In case **(1)**, the hyperslice is the accumulation of a fixed 2D spatial slice of every time step. In case **(2)**, the hyperslice is the volume at a particular time step. For this case, the hyperplane or volume conceptually created by the story presented to the user has the 2D slicing plane moving in space over time. These hyperplanes are the most difficult to understand and interpret, because space is skewed by time in the hyperplane.

This type of hyperplane can be used to track features in the time-varying data. If a feature located in a 2D plane that moves over time, which can be matched approximately orthogonal to the normal direction of the slicing hyperplane, then we can track the feature in the hyperslice. Figure 3(b) shows such an example. In the concentrate data set, where a planar feature moves along a direction in spatial domain, the user sets the coefficients (a,b,c) of the hyperplane accordingly to slice this feature spatially. By adjusting the coefficient d in the hyperplane equation, the spatial offset of the 2D slices for every time step, we can capture the traces of the data feature.

When we perform 4D projection using this type of hyperplane, we are integrating subsamples at different spaces and times into a final voxel in the resulting hyperprojection volume. Thus, a voxel in the final volume does not represent fixed time step or a fixed position in space, like the previous two cases, respectively. This distortion of space and time makes the volume very difficult to understand, although we have applied this to the track movement of features, like the above example with the hyperslice. Further discussion on how we can use this type of projection will be given in the additive integration method in the next section.

4.2 Integration Operators and Transfer Functions

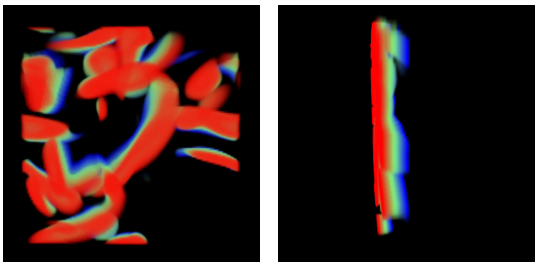
In our 4D projection method, an integration step is needed to combine the information from subsamples in sequence of hyperslices to derive the final voxel color in the image hyperplane. Methods used in regular 3D volume rendering are typically either optically derived or based upon data extraction, such as the minimum/maximum intensity projection method. Since there is not a notion of optically viewing in 4D space, we look towards using integration operators as a form of data extraction. They will provide summary information at the voxel position indicating spatial and temporal characteristics of the data set. In the following, we present several integration operators and their associated transfer functions. The design of the transfer functions is closely tied with the type of integration operators that we utilize. There is an expectation of what type of data that we are searching for and how to summarize that information. The transfer function decides how to code the individual pieces of data, and the integration operator decides how to combine it into a meaningful summary voxel. We note that not all integration operators are meaningful to all of the hyperplane families. Only certain subsets of hyperplanes make sense when using particular integration operators to project the data.

4.2.1 Alpha Composition

Like 3D volume applications, we can use alpha composition in 4D hyperprojection to emphasize a particular range of data values and time steps. The transfer function is designed so that more important values will occlude less important values, via the alpha channel. Values with the highest opacity will be the most dominant in the image hyperplane. The output color channel from the transfer function is used to distinguish among different values the user is searching for.

For example, we have created an alpha compositing transfer function for the hyperplane families of $t = -e/d$. There are two inputs to the transfer function, the scalar value of a sample and the time ordinal value of the sample. The alpha is determined both by the data value and the time ordinal, by having two alpha transfer functions and modulating the results together for the final subsample alpha. In this way the user is able to specify an interval volume they wish to view, and the range of time steps that are of interest. In our examples, Figure 4, the color transfer function outputs colors based on the time ordinal of a subsample, where blue is the oldest time step, red is the latest, and green is median time. We color by time, so that when we project and several time steps are composited together in a single image hyperplane, we are able to distinguish what points in space are contributed by a particular time step. We have found this useful for determining the space-time boundary of interval volumes, and seeing how values move in space over time. In both images, the interval volume moves over time, starting at blue, moving to green, and eventually ending at red.

When we use the hyperplane families that utilize $d = 0$, we project along a vector that samples in space, but not time. For example, the user could specify a hyperplane equation such that the projection vector is $(0,0,1,0)$. The final image volume would result in a series of time evolving images of the data rendered looking down the z axis, using alpha compositing and parallel projection. These 2D images are stacked along an axis which translates to the time axis of the image. Figure 5 shows examples of these, using the



(a) (b)

Figure 5: Using alpha composition, but with the hyperprojection normal of $(0,0,1,0)$. The final image volume is the same as rendering several time steps with a parallel projection looking down the z axis and stacking the images along a t axis. (a) shows when each time step is colored by time. (b) is (a) rotated to show the stacking of the images along the t axis.

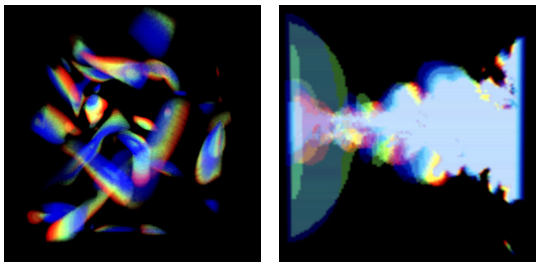
vortex data set again. In the figure, we color each slice by time in order to distinguish each slice from the other slices. If we were to render each time step using traditional raycasting methods, looking down the z axis, using parallel projection, and storing each image in one volume, we would get the same result. By passing a cutting plane perpendicular to the axis which corresponds to time, we can quickly generate an animation looking down the z axis.

4.2.2 First Hit

We also use first hit integration operator in our 4D projection, where the 4D sampling ray will stop at the first encountered subsample that is not completely transparent. When using first hit method, we can distinctly detect the first instance of a feature along the projection vector and additionally can allow the user to view interior features of the hyperprojection volume more easily. This is because first hit will collect the first subsample in space and time as the sample for the hyperprojection. If we had used alpha composition, and used low opacities, voxels in the hyperprojection volume may be opaque because of the accumulation of alpha over the projection vector, and a combination of colors from several subsamples.

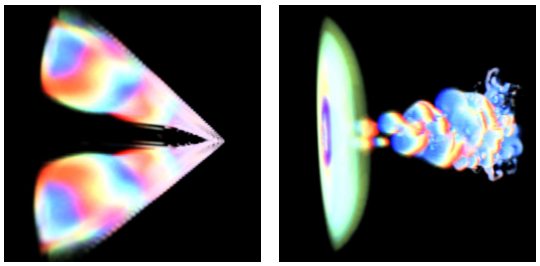
When using first hit in conjunction with the hyperprojection of $t = -e/d$, it allows us to see the space-time boundary of an interval volume showing the first time step that contains the values the user is interested in. It also allows the user to view interior features through foreground elements when several time steps overlap in one image frame, as long as translucent alpha values are specified. An example is shown in Figure 6(a) using the vortex data set. A similar coloring scheme used in alpha composition is used here for time steps, and a low opacity value is specified for each time step. The image shows the first time step that contains the user specified values. We can rotate the image volume and observe internal features through voxels that are close to the eye. If we had used alpha composition like Figure 4(a), the subsamples over time that overlapped in space would have blended together, and created final voxels with a high opacity . We would not be able to see through them spatially nor be able to distinguish the first instance because of color combination.

If we have a hyperprojection of hyperplanes of $d = 0$, we have several images rendered using first hit parallel projection looking down some spatial axis, where the images are stacked along a t axis in the final image volume. Figure 6(b) shows the jet data set using the first hit method with a hyperprojection normal of $(0,0,1,0)$. We have colored the samples by time, so that each time step can be seen in correlation with the other time steps. With a low opacity set for a subsample, we are able to see each layer through the layers that are closer to the eye.



(a) (b)

Figure 6: First hit integration method where (a) uses the vortex data set and a hyperprojection normal of $(0,0,0,1)$, so that we can see distinctly see the first time step that contains the interval volume. With a low opacity for each voxel, we can see through cells. (b) uses the jet data set and a hyperprojection normal of $(0,0,1,0)$, which is similar to having several time steps with first hit rendering looking down the z axis, stacked along a t axis.



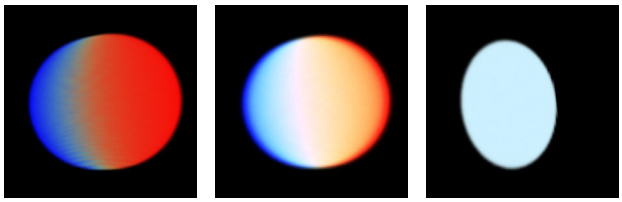
(a) (b)

Figure 7: The delta wing and jet data set, shown in (a) and (b) respectively, using additive integration and the hyperprojection normal of $(0,0,0,1)$. Individual time steps are colored by a linear ramp from blue to green to red. White color indicates that the interval volume occupies that region in space over all time steps. Other colors indicate that fewer time steps occupy that region in space.

4.2.3 Additive

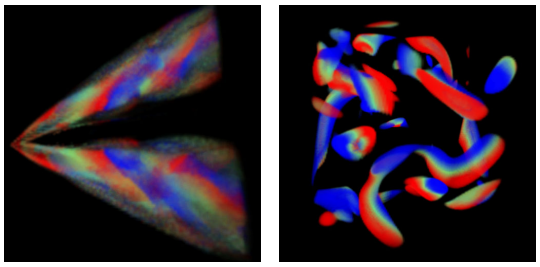
The additive projection method is an unweighted summation of color and alpha channels of the subsamples. We do not use a compositing operator, like what is used in alpha composition. After summation, we normalize all voxel channels by the maximum channel value, to account for the maximum displayable hardware color channel value. With the additive integration operator and an appropriate transfer function, we can show the user where two different subsample values contribute to the projection voxel. The method of showing this feature is by designing transfer function such that when two different valued subsamples are added together, it generates a third unique color value different from normal transfer function output.

When using additive method in conjunction with the hyperplane family of $t = -e/d$, we can show how features in several time steps overlap in space. When we color subsamples by time ordinal using a linear red, green and blue transitioning color ramp, if subsamples from all time periods occupy the same region in space, the final voxel color will be white. This is because a voxel position in the image hyperplane corresponds to a single point in space as it evolves over time. We will get other shades of overlapping occurring too, such as yellow, magenta and cyan, in addition to the three primary colors. Figure 7 shows the delta wing and the jet data set using additive integration. In both, the white regions show where the interval volume overlaps in space for all time steps, while other colors indicate that fewer time steps occupy that region in space in the time sequence.



(a) (b) (c)

Figure 8: A sphere moving over time to the right. (a) The sphere integrated by alpha compositing using a hyperprojection normal of $(0,0,0,1)$. (b) Switching to additive, we can see how the sphere overlaps in time and space. (c) Using a hyperprojection normal of $(1,0,0,1)$ and additive again. Since we see a white sphere, we know that the sphere is moving in that space-time vector, because the 4D projection ray sampled in the same direction along which the sphere is in 4D space.



(a) (b)

Figure 9: The delta wing and vortex data set, shown in (a) and (b) respectively, using maximum intensity integration and the hyperprojection normal of $(0,0,0,1)$. Blue indicates that the value fell over time, red shows that the value rose over time, and green shows that the value rose and then fell in the time sequence.

Additive integration allows us to project the hyperplanes of $d \neq 0, a \vee b \vee c \neq 0$. If we imagine the hyperprojection normal as the direction of a camera moving over time, translated to a 4D space vector, then the hyperslices taken would be what the "viewer" is seeing in a volumetric region relative to its movement. Data values in space that have the same movement in space relative to the camera should occupy the same position in every hyperslice, because their positions do not change in relation to the camera over time. If we were to composite these hyperslice images together and used additive integration, we can detect these overlaps, which corresponds to a form of feature tracking based on the hyperplane projection. An example of this can be seen in Figure 8, where we have a sphere moving left to right over time.

4.2.4 Minimum/Maximum Intensity

We can adopt the minimum/maximum intensity projection used in 3D raycasting for our 4D volume projection. When we integrate the hyperslices together, we choose the sample that has the most extreme value along the hyperprojection ray as input to the transfer function.

When working with the hyperprojection of $t = -e/d$, it allows the viewer to see bimodal change over time within a time evolving data set. For instance, when the users use a transfer function that colors subsamples by time ordinal, they are able to see which time step generated the most extreme value at a point in space. This would show the change in value, whether the value was rising or falling over time. Examples using this technique can be seen in Figure 9, with the delta wing and vortex data set using maximum intensity. We color by time, like in previous examples. For example, the blue areas in the two images indicate that the maximum value occurred at the earliest time and then the value fell over time. The red regions indicate the opposite case, where the value was rising over time, and the green regions show that the value first rose, and then fell over the time sequence.

4.2.5 Average

In addition to operators that do color compositing or select one subsample out of many for the transfer function, we can look at operators that analyze the trend along the projection ray. The average operator averages the subsamples along the projection ray and provides the result to the transfer function.

When used with the hyperprojection of $t = -e/d$, the image hyperplane is the average trend of a position in space over time. The vortex data set is shown as an example of the average operator in Figure 10(a), with the foreground cropped from it. The transfer function used in this example maps the entire data range to a linear ramp of blue, representing small values, to green, to red, representing large values. Hyperprojecting case **(2)** of hyperplane families would show the average value over a spatial vector, where a 2D slice in the final volume is a particular time step.

4.2.6 Deviation

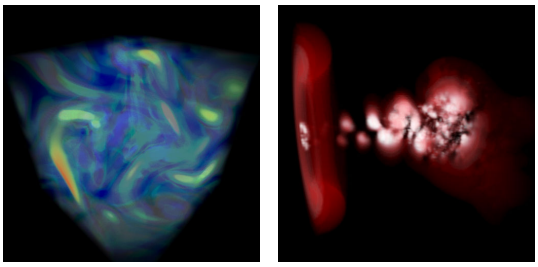
If the user specifies a data value, the deviation operator would give a visual representation to the user on how far values differ from the user specified value along the projection ray. The deviation test could use any kind of deviation operators, and in our example, the standard deviation is used. The operation of the deviation operator is similar to the average operator. As the projection ray collects subsamples, these are used as inputs to a deviation test. In our deviation operator, every subsample is input into a standard deviation test from a user supplied value. The transfer function is a mapping from the computed deviation value along the projection ray to color and opacity.

When the deviation operator is used in conjunction with the hyperprojection of $t = -e/d$, the user is shown how a point in space deviates from the user supplied value over time. An example of the deviation operator used with the jet data set can be seen in Figure 10(b). In this example, when the standard deviation along the projection ray is small, the transfer function returns red, which linearly ramped to white for larger deviation values. A maximum upper bound for deviation values is cut off, which results in transparent regions in the image hyperplane. If we use case **(2)** hyperplane family for projection, the user can see how each time step deviates from a given value over a spatial vector.

4.2.7 Integration Operator Summary

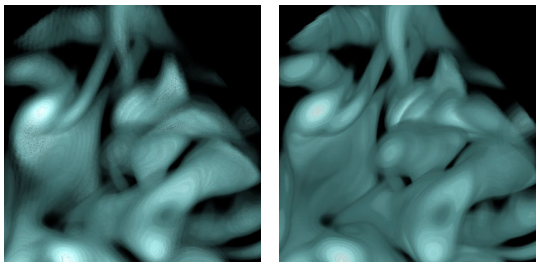
Using our hyperprojection algorithm, users are able to see features that might not normally be visualized with traditional volume rendering techniques. The key benefits that are derived from this visualization is a snapshot of a time varying sequence presented in a single volume. It really depends on the type of integration operator and the projection normal that the users choose for their visualization on what sort of information is presented to them. The previous sections contained the different types of projection, and here we will summary the different integration operators we have presented and the information obtained from them.

The alpha composition operator is the most basic operator for projection. It allows the user to choose what values of interest and time steps they wish to visualize and are presented with those. First hit operator removes the blending aspect from the alpha composition operator and the user is able to see the first instance of a feature distinctly along the projection vector, rather than blending several space-time voxels together. It also gives the capability of translucent hyperprojection volumes, by specifying a low opacity



(a) (b)

Figure 10: (a) shows the vortex data set using the average operator and a hyperprojection normal of $(0,0,0,1)$, from a cropped portion. Subsamples are averaged along the projection ray and the result is the input value for the transfer function. (b) shows the jet data set using the deviation operator and a hyperprojection normal of $(0,0,0,1)$. Subsamples are used in a standard deviation test from a user supplied value. The standard deviation result is used as input for the transfer function where red shows small deviation and white shows large deviation.



(a) (b)

Figure 11: Visualizing a hyperslice from the vortex data set. When zoomed in, the hyperslice with incorrect sampling in (a) loses detailed information compared with the hyperslice with correct sampling in (b).

for subsamples in the transfer function. The additive operator allows the user to see overlap in time and space. When projecting through the time axis with additive integration, the user can see how values overlap in space over time. If we use an arbitrary projection through 4D space, we can use the additive method to follow a feature. Minimum/maximum intensity operator allows the user to see the most extreme value or the time step that contained the most extreme value over the projection vector. This allows the user to detect interest areas and bimodal value change. Extending the idea of minimum/maximum intensity over the projection vector, we can do similar numerical analyses with an average operator and a deviation operator. The user is able to project along a direction and detect the average value or the deviation of a value. Many such analyses along the projection vector could be created, the visualization possibilities are limitless.

5 Implementation Details

The software system for implementing our direct rendering method of 4D space can be separated into two discrete methods, by 4D raycasting or by hyperslice-based projection. The former can be done entirely in software to generate 4D projection, though we use a view-aligned hardware rendering method to rasterize the image hyperplane. The second method can be achieved by a hardware slicing and software projection or a hardware slicing and projection.

5.1 Slicing Implementation

The hyperslicing implementation is straightforward to accomplish in software. Given a camera position, a *look* direction, an *up*, and *over* vector in 4D space, we can set up an orthogonal 3D coordinate system which defines all the points on our hyperplane. The *look* direction becomes the hyperplane normal. To generate the other two coordinate axes for the hyperplane, the cross product in four dimensions [Blinn 2003] requires three vectors to produce a vector orthogonal to the other three. By applying the cross product three times, i.e. $xaxis = up \times over \times look, yaxis = xaxis \times up \times look, zaxis = zaxis \times yaxis \times look$, we construct our hyperplane coordinate system. Using quad-linear interpolation, we can resample the 4D points to lie within our hyperplane by progressing along the 3D hyperplane coordinate axes in 4D space.

We also provide a method for hyperslicing in hardware. Let the dimension of the original 4D space data stored in *xyzt* fashion be $(xdim, ydim, zdim, tdim)$, where the time steps are volumes $Vol_0, Vol_1, \ldots, Vol_{tdim-1}$. Getting a 2D slice out of a 3D volume can be implemented using 3D texture hardware in a manner similar to view-aligned hardware volume rendering: we need to load the data values of each volume Vol_τ $(0 \leq \tau < t_{dim})$ into memory as a 3D texture object and update slicing plane if necessary when rendering the slice. In order to guarantee that the final hyperslice would not be distorted, parallel projection instead of perspective projection is used and the width and height of each 2D image slice is set to be the longest diagonal of the original 3D volume with $length = \sqrt{xdim^2 + ydim^2 + zdim^2}$. In our case, it is desirable to set the viewing direction to be the same as the slicing direction (a,b,c) so that 2D image slices will be aligned automatically in the hyperslice. As a 2D image slice is rendered, its content is read out from frame buffer and stored into the corresponding segment of the hyperslice. The frame buffer is cleared before we render each slice.

An issue that needs to be addressed is the sampling rate along t dimension. If $d/\sqrt{a^2+b^2+c^2} > 1$, then in order to get sufficient samples from the data, we need to insert one or multiple intermediate slices between every two slices, S_τ and $S_{\tau+1}$, where τ is the current time step and $0 \leq \tau < t_{dim} - 1$. Using OpenGL multitexturing and `NV_register_combiners` extensions, this can be done by keeping the consecutive two volumes Vol_τ and $Vol_{\tau+1}$ in texture memory simultaneously and interpolating the intermediate slices. An efficient way to perform this is to swap out only one volume at a time and apply two blending equations alternatively. Figure 11 shows an example of the rendering of two hyperslices with and without correct sampling.

The advantage of this approach is that it can be completely done using graphics hardware and it is quicker than the quad-linear software interpolation approach. However, the sides of each 2D image slice could be larger than *length* due to the power-of-two restriction imposed by OpenGL implementation. This may adversely increase the size of the final hyperslice and create much empty space. Observe that when we increase the size of frame buffer in order to satisfy the constraint, the actual data will not really exceed the boundary of the theoretical one because we always make the viewing direction to be the same as the slicing direction (a,b,c). Thus a convenient way to keep the size of hyperslice from increasing is to read only partial of the image slice with sides delimited by *length* each time when we read the image from frame buffer.

5.2 Projection Implementation

A software implementation for doing 4D raycasting is a straightforward extension of generating a hyperplane in 4D space. The previous section explained how to sample the hyperplane by creating a 3D coordinate system and sampling the 4D space. Given the

definition of an initial view volume from our hyperplane, we perform 4D raycasting by casting a ray from each voxel in the volume through 4D space along the hyperplane normal. We integrate by using our integration functions defined in the previous section. The volume is visualized by any standard volume visualization method.

Alternatively, we can use the hardware hyperslicer to generate several intermediate hyperslices that will be composed together to generate the final image hyperplane. Projection of the volumes is easy if hyperslices are correctly registered such that the same array position in all slices corresponds to the voxels along the 4D projection ray. To project all the slices to the image hyperplane, we compose all samples at the same position in every 3D array to generate the projection voxel. We integrate by iterating through every hyperslice at the same position in space, and collecting subsamples. Again, we can also use a hardware method to project the hyperplane slices into the image hyperplane. Using correctly registered 3D hyperslices, for every hyperslice, we project 2D planes with the same position in every hyperslice into frame buffer. The frame buffer is copied and placed into the final image volume at the position that the 2D slices were taken from the hyperslices. Doing this would be equivalent to performing the projection for all voxels in one 2D plane. After we have performed our 4D projection to a 3D volume, any volume visualization technique can be used to view the hyperprojection. It is non-trivial to implement some of the integration operators in graphics hardware and still some other operators can not be implemented at all in current graphics hardware. Alpha composition and additive operators can be done with dependent texture lookup for the transfer functions, first hit and minimum/maximum intensity might be accomplished with a multi-pass algorithm using the stencil or *z* buffers. The other integration operators may be able to use more recent programmable pixel shader graphics hardware, but it is unlikely to provide division and square root operations, which are needed in the average and deviation operators.

6 Conclusion and Future Work

We have presented a new, flexible method for viewing time-varying volumetric data. The time-varying data are considered to be a 4D data field, rather than separate volumes enumerated by time. In this manner, we can perform high dimensional direct rendering of the data. Our method generates an image volume of 4D space by projecting the data to three dimensions. Despite the difficulty to visualize 4D space, we can help the user better visualize a hyperslice by showing it as a series of 2D slices taken from every time step. The hyperslices are then projected along the hyperplane normal to generate an image hyperplane. The hyperplane equations and their projections can be divided into several categories with different interpretations. We utilize different integration operators and transfer functions and are able to present spatio-temporal features to the user in an intuitive manner. We believe that these methods are useful for exploring time-varying data in an informative way.

Future work includes investigating the tradeoff of memory and speed between software 4D raycasting and hardware slicing. In hardware slicing, several intermediate 3D volumes are generated to composite together. It may demonstrate that a 4D software raycaster is in fact faster than a hardware slicing and projection method, due to graphics hardware RAM limitations and bus speeds. Another interesting aspect to explore would be seeing if specifying an arbitrary projection path would generate informative images. For instance, the user could specify a spline in 4D space, which would be the projection direction. By projecting along this arbitrary path, we may be able to follow features in space and time for tracking. Finally, we need to continue the design of interactive interface for this high dimensional direct rendering of time-varying volumetric data. More GUI tools that allow users to probe the data and get more than visual feedback, and even to specify their own equations for the integration operators, would be immensely useful.

Acknowledgements

This work was supported in part by NSF grant ACR-0118915, NASA grant NCC-1261, Ameritech Faculty Fellowship, and Ohio State Seed Grant. We thank the anonymous reviewers for their helpful comments.

References

BAJAJ, C., PASCUCI, C., RABBIOLO, G., AND SCHIKORE, D. 1998. Hypervolume Visualization: A Challenge in Simplicity. In *Proceedings of 1998 Symposium on Volume Visualization*, ACM Press, 95–102.

BANCHOFF, T. 1990. *Beyond the Third Dimension: Geometry, Computer Graphics, and Higher Dimensions.* Scientific American Library.

BANK, D., AND SINGER, B. 1995. A Predictor-Corrector Technique for Visualizing Unsteady Flow. *IEEE Transactions on Visualization and Computer Graphics 1*, 2, 151–163.

BLINN, J. 2003. Lines in Space Part 1: The 4D Cross Product. *IEEE Computer Graphics and Applications 23*, 2, 84–91.

COHEN, M. F., 2000. Visualization of Everyday Things (http://www.research.microsoft.com/~cohen/Vis2000.pdf).

HANSON, A., AND CROSS, R. 1993. Interactive Visualization Methods for Four Dimensions. In *Proceedings of IEEE Visualization '93*, IEEE Computer Society Press, 196–203.

HANSON, A., AND HENG, P. 1992. Four-Dimensional Views of 3D Scalar Fields. In *Proceedings of IEEE Visualization '92*, IEEE Computer Society Press, 84–91.

HANSON, A., AND HENG, P. 1992. Illuminating the Fourth Dimension. *IEEE Computer Graphics and Applications 12*, 4, 54–62.

JANKUN-KELLY, T., AND MA, K.-L. 2001. A Study of Transfer Function Generation for Time-varying Volume Rendering. In *Proceedings of 2001 International Workshop on Volume Graphics*, 51–65.

KINDLMANN, G., AND DURKIN, J. 1998. Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering. In *Proceedings of 1998 IEEE Symposium on Volume Visualization*, ACM Press, 79–86.

KNISS, J., KINDLMANN, G., AND HANSEN, C. 2001. Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets. In *Proceedings of IEEE Visualization '01*, IEEE Computer Society Press, 255–262.

KNISS, J., KINDLMANN, G., AND HANSEN, C. 2002. Multidimensional Transfer Functions for Interactive Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics 8*, 3, 270–285.

MICROSOFT RESEARCH GRAPHICS GROUP, 2000. VideoCube (http://research.microsoft.com/downloads/VideoCube/VideoCube.asp).

REINDERS, F., POST, F., AND SPOELDER, H. 2001. Visualization of Time-Dependent Data using Feature Tracking and Event Detection. *The Visual Computer 17*, 1, 55–71.

SAMTANEY, R., SILVER, D., ZABUSKY, N., AND CAO, J. 1994. Visualizing Features and Tracking Their Revolution. *IEEE Computer 27*, 7, 20–27.

SILVER, D., AND WANG, X. 1997. Tracking and Visualizing Turbulent 3D Features. *IEEE Transactions on Visualization and Computer Graphics 3*, 2, 129–141.

VAN WIJK, J., AND VAN LIERE, R. 1993. Hyperslice. In *Proceedings of IEEE Visualization '93*, IEEE Computer Society Press, 119–125.

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