Volume Graphics (2003)
I. Fujishiro, K. Mueller, A. Kaufman (Editors)

Chronovolumes: A Direct Rendering Technique for Visualizing Time-Varying Data

Jonathan Woodring[1†] and Han-Wei Shen[1‡]

[1] Department of Computer and Information Science, The Ohio State University, Columbus, Ohio

Abstract

We present a new method for displaying time varying volumetric data. The core of the algorithm is an integration through time producing a single view volume that captures the essence of multiple time steps in a sequence. The resulting view volume then can be viewed with traditional raycasting techniques. With different time integration functions, we can generate several kinds of resulting chronovolumes*, which illustrate differing types of time varying features to the user. By utilizing graphics hardware and texture memory, the integration through time can be sped up, allowing the user interactive control over the temporal transfer function and exploration of the data.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Viewing algorithms

1. Introduction

Time varying scientific data traditionally has two methods of visualization. The first method, being normal volume rendering or isosurface extracting time steps of interest and rendering those individually. In this manner, the continuity and connection between time steps is minimized. The information of a single time step is present, but the context of the surrounding earlier and later time steps is not immediately available, except by comparing rendered time steps. The second method is to create an animation from the data set. While context of the surrounding time steps is provided, it relies on the viewer's memory of what happened to tie together spatial relations. Also, it may be too time consuming for a transfer function update and re-render to interactively explore the animation in real time.

We propose an alternative method for viewing time varying data, related to late 1800 photographic methods. During this period, Etienne-Jules Marey was studying the human body, and was aiming to discern the laws that drove human physiology. He manufactured graphing machines that were able to measure forces and movement over time, but the machines were not adequate to studying locomotion completely. "He wanted to depict in a single image all the relationships occurring both between one body part and each of the others and between one body part and the body as a whole at each of several instants of a specific movement executing during a discrete unit of time and in a specifically defined and constant space. [1]" In 1878, Eadweard Muybridge captured images of a horse moving over time with multiple cameras, where the horse was captured separately on individual frames. Inspired by Muybridge's photography and the advancement of camera technology, he was able to manufacture a camera system that was able to take multiple series of images within a single plate. Dubbed chronophotography, seen in Figure 1, Marey was able to capture the progression of a body over time and space at regular intervals within a single image frame from a single point of view.

If we were to apply Marey's chronophotography method to computer visualization, it would be stated: given a sequence of time varying images, the resulting image is a combination of all time steps of interest. Our method is an advancement of this, where: given a sequence of time varying *volumes*, the resulting volume is a combination of all time steps of interest. We have several benefits of being able to study the data with this method. First of all, we can see every time step of interest at once, with weighted interest. This provides context of meaning from earlier and

† woodring@cis.ohio-state.edu
‡ hwshen@cis.ohio-state.edu

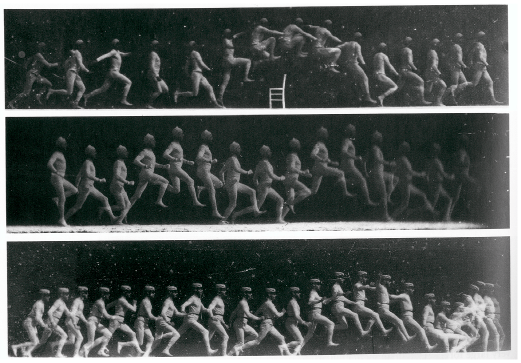


Figure 1: *Marey's chronophotography.*

later time steps, and we can reason about spatial relationships between structures as they progress over time. Also, since this is a single volume, unlike an animation, we don't have to rewind and forward the animation to see the progression. Secondly, by combining volumes we have an advantage over chronophotography such that depth information is preserved. We are able to rotate the volume and structures are properly depth occluded. Lastly, we are able to make rapid updates to the transfer function and interactively explore the time series.

The basic functioning of rendering a time varying volume follows. The user designs a transfer function for time and data and then the volume is integrated through time. For every voxel in the final viewable volume, a single voxel is the result of integrating through every time step at that position in space. The resulting volume is then rendered with traditional volume raycasting techniques. The added step we have introduced is integration through time in the raycasting pipeline. In the time integration step, we will introduce several different time accumulation methods for rendering, in addition to feature enhancement. Finally, hardware acceleration will allow us to quickly and interactively update the transfer function.

2. Related Work

Previous work on time-varying visualization primarily focuses on data compression, acceleration of visualization techniques, and time-varying feature tracking. For data compression, Guthe and Straßer [2] used 3D wavelet transformation, coefficient encoding, and motion compensation to achieve efficient volume compression and decompression. Lum *et al.*[3] used Discrete Cosine Transform and vector quantization techniques to compress multiple time steps of each voxel into an 8 bit index. The quantized volume is loaded into the texture memory, and a dynamic color palette is used at run time to decode the compressed data and animate the volumes. More recently, Sohn *et al.*[4] combined wavelet transform and MPEG compression scheme to encode blocks that contain significant features. Isosurface seed sets are also encoded to achieve interactive data browsing. Neophytou and Mueller [5] proposed a space-time point rendering technique. Four dimensional Body-Centered Cartesian (BCC) grid is used to provide a more efficient sampling.

To accelerate the visualization computations, various algorithms were proposed. For volume rendering, Shen and Johnson [6] proposed a differential volume rendering algorithm which performs difference encoding and can selectively cast rays only into those voxels that change values dramatically in time. Shen *et al.*[7, 8] proposed a Time-Space Partitioning Tree data structure to store the temporal variation based on the coefficient of variation in the data. Partial images rendered from data blocks are reused if little temporal variation is observed. Anagnostou *et al.*[9] used a statistical hypothesis testing technique to measure voxel variances and incorporated the technique into the shear-warp volume rendering algorithm. For isosurface extraction, Sutton and Hansen [10] extended the Branch-on-Need Octree (BONO) algorithm to manage time-varying data for isosurface extraction. Shen [11] proposed a temporal hierarchical index tree to arrange the extreme values of each data cell based on their temporal variation to speed up isosurface extraction.

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 [12] to track time-varying isosurfaces evolving in time for structured and unstructured data. Important temporal events such as bifurcation can also be detected [13]. Banks and Singer [14] used a predictor-corrector method to reconstruct and track vortex tubes from turbulent time-dependent flows. van Walsum *et al.*[15] designed a feature viewer using iconic visualization techniques to assist visualization of important temporal events.

Little attention has been paid to direct visualization of 4D data. Hansen *et al.*proposed a method [16, 17] for four dimensional illuminations. In their method, three dimensional Phong lighting model was extended to four dimension, 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. Bajaj *et al.*[18] generalized the object space splatting technique into a hypervolume splatting method that can be implemented by texture mapping hardware. In essence, visualization of high dimensional objects were the primary interest for those methods, therefore no explicit temporal feature tracking was attempted.

Research on transfer function design has been primarily focused on static volumes data. A common practice is to use a single transfer function for the entire time sequence. Kindlmann *et al.*[19] 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.*[20] later followed up the work in [19]

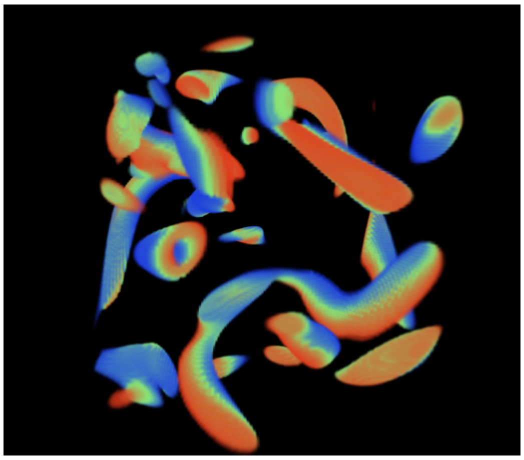


Figure 2: *Alpha composition of 10 consecutive time steps of the Vortex data set. The isosurface progresses from blue to green to red over time.*

with an intuitive user interface and introduced the concept of dual domain interaction. For time-varying data, Jankun-Kelly and Ma [21] 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 can be effective for certain scenarios.

3. Integration Through Time

In normal raycasting, a ray is cast from the eye through a pixel on the image plane. The ray then travels through the prospective volume, using an integration function as it passes through voxels along the ray to accumulate colors. The final color of the pixel is the result of all the color along the ray as defined by the integration function and transfer function.

To visualize data from multiple time steps in a single image, a projection of the four-dimensional data to a renderable lower dimensional space is obviously necessary. Since our primary interest is to understand the temporal evolution of time-varying data, it is desirable that this projection preserves the spatial relationship of data features such as locations and visibility in the embedding three-dimensional space. This is to ensure that the user can reason about feature shapes and positions easily. To achieve this, we first use an orthographic projection to transform the four-dimensional data to a three-dimensional volume. The transformation is to project the four dimensional voxel data (x,y,z,t) along the t (time) axis to a three-dimensional "image plane". Each "pixel" in the resulting "image" is in fact a voxel, which has a value from integrating $\mathbf{v}(x,y,z,t)$ along time.

Hereafter we call the projection volume a *chronovolume*. To visualize the chronovolume, we can use regular volume rendering methods to project it to a two-dimensional image plane from arbitrary viewing directions. Is is only necessary to reintegrate along time when changing the transfer function, to recreate the chronovolume.

```
while(true) {
  if(transfer function was updated) {
    integrate through time
    build chronovolume
  }
  if(view position has changed) {
    raycast chronovolume
  }
}
```

Unlike volume integration in the three-dimensional spatial domain, there is no "correct" way of performing the time integration when constructing the chronovolume. This is because there is no analogous concept of time-integration in the physical world. The design of time integration operation should depend on the goal of visualization. Different integration schemes will pick out different time-varying features. We have implemented several integration functions. In the following, we describe the functions in detail.

3.1. Integration and Transfer Functions

When using raycasting to render a volume, there is an array of integration functions that can be used. We can utilize these integration functions when applied to integration through time. In the following, we describe the use of alpha composition, first temporal hit, and maximum/minimum integration functions for creating a chronovolume. Each of these functions provides a particular insight to the time-varying data. What follows is the description of the different algorithms and the example images of the techinques referred to in Figure 3 are on the color plate. Color is an important feature to the chronovolume images, and viewing them in gray scale lacks much of the information that is presented in the images.

3.1.1. Alpha Compositing

One method to integrate data when casting a ray through time is to adopt the idea of alpha blending commonly used in volume rendering. To achieve this, we define an opacity function over time and an opacity function over data. The opacity of a single time step value before being integrated into the result value is a modulation of time opacity and data opacity. The time opacity allows us to emphasize certain time steps, while the data opacity allows us to emphasize certain data values. The blending order of integration through time follows the time sequence with two alternatives - we can integrate the voxels from the oldest time steps to the newest time steps (forward integration), or from the newest time step to the oldest time step (backward integration). When forward integration is used, earlier time steps can occlude older time steps, hereafter called *temporal occlusion*. On the other hand, when the backward integration is used, the opposite effect is seen. The user can determine what weight or importance to give to a single time step by changing the alpha factor for a time step. When the user does so, more important time steps can be visible, with fading to transitory time steps within the context of past and present of the important time step. By using a hat alpha function over time, for example, the user can isolate a single time step, and smoothly fade to earlier and later time steps. An inverted hat alpha function allows the user to distinguish two time steps and smoothly fade toward a median time step.

Our transfer functions minimally have two inputs, data value and time ordinal, so the color of a single time step voxel will depend on its time step and data value. We have separated the transfer function into two portions, time and data, where the output color of a value at a particular position in time and space is a modulation of time color and data color. In the case of a color transfer function for time, we have used a linear red to green to blue transition from newest to oldest time steps. This allows for three distinct periods, roughly corresponding to "past", "present", and "future". A warm-cool color space may be appropriate when operating with two distinct time periods and transitory in-betweens. Using a luminance transfer function for time, would allow to user to focus on a single time step, while lesser important time steps would fade to black. While it is possible to modulate the data color transfer function with the time color transfer function, the modulation of colors muddles information, as it is near impossible to discern the combination of colors from data and time. It is more recommended to use either the data color transfer function or the time color transfer function, but not both simultaneously, by setting the one modulation color to a constant color. Given the proper interface, the user could interactively toggle between data transfer function and time transfer function to see both spaces. In all of our examples, we have only used temporal coloring.

In Figures 2 and 3(a), we show examples of using alpha blending through time. Here, we can see several isosurfaces as they progress. In Figure 2, 10 consecutive time steps are time alpha composited; there is a great deal of overlap in space between the time steps. In Figure 3(a), three spaced time steps over a period of 20 are time composited together; there isn't quite as much overlap in this image, as the time steps are spaced apart. Blue indicates the oldest time step, while red indicates the most recent time step. Green is a time step that is the median between the newest and the oldest time step. In both images, we can see how the isosurfaces progress over time and space.

A position in space may have several time periods that have an opacity that is non-zero. By providing an importance value, via the alpha channel, to a time step, we can see what time step or steps occupied a voxel over time. More important time steps will dominate, thus these time steps will have their color appear in the chronovolume. For example in Figure 2, the red time step dominates all others, while blue is the least dominant. We get an overlay effect where the isosurface moves from blue to red. In Figure 3(a), we have spaced out the time steps so we can distinctly see each time step. Visualizing chronovolumes generated with different integration order and opacity functions in a comparative manner can assist us to understand the spatial relationships of certain data features in different time steps, and thus reason about how the features evolve.

3.1.2. First Temporal Hit

When, first temporal hit is used, as we integrate through time, the final color for a chronovolume at a voxel is determined by the first voxel that we encounter with non-zero opacity, depending on the integration order. When we choose the first time step that meets this criteria, we stop integration through time. This is different from a first hit technique used in the spatial domain, where the first hit along a view vector is completely opaque. Here, a voxel does not need to be opaque, because we are doing first hit in the temporal domain. Spatially, there may be interesting features we would like to see, thus a voxel might not be fully opaque will allow the user to see through first temporal hit voxels. The rendering order also determines what is first hit, whether it be newest to oldest, oldest to newest, or some importance ordering. By importance ordering, the user can specify which time step would be encountered first when integrating through time, thus more important time steps would be checked first.

When doing a first hit in the temporal domain, rather than the spatial domain, there are features in one time step that may be behind another time step, depth-wise from the viewer. First hit allows the user to apply a low opacity to a single time step so that the user can see through a time step in space. If we were using alpha blending through time, and applied a low opacity to a time step, it may be dominated by another time step, and thusly temporally occluded or blended. In addition, when using alpha blending, opacity builds up very quickly if many time steps overlap the same region in space. By using first temporal hit, we can definitively say whether or not a certain time step occupied a region in space over time sequence, without getting the blending effect with other time steps. First hit allows the user to distinctly isolate a time step while allowing the user to see spatially through a time step.

The first hit method to integrate through time can be seen by Figure 3(b). Time steps are distinct from each other, and we can see farther objects through objects that are nearer to the viewer because we have used a low opacity. If we had used alpha composition through time, regions where time steps overlap would have had a blending of time steps, and a higher opacity, potentially occluding objects spatially.

3.1.3. Additive Colors

Sometimes it is necessary to know whether features at different time steps overlap, or by how much they overlap. Images generated using the first hit or alpha blending integration method mentioned previously, however, do not provide such cues. This is because at any given point in the image we can only see features from one time step due to temporal occlusion, or it is very difficult to deduce how features in different time steps are overlapped from the alpha blended colors. To solve this problem, instead of using alpha compositing, we can use an additive color integration method.

To achieve this, as we integrate through time, we take a color summation of all time steps at the same position, and do a normalization step at the end to account for the maximum value ranges of graphics hardware. Now instead of time steps occluding each other, when several time steps occupy the same space and we use a meaningful color transfer function, it can lead to combination colors which clearly show how features in multiple time steps overlap. We use time transfer function which linearly transits from red to green to blue. For the regions where features in all time step overlap, a white color is produced. This can be useful when trying to study several time steps simultaneously, as they do not occlude each other, but show up as a color which signifies their overlap. Alpha blending or first hit is more useful when trying to study a single time step in relation to other time steps, since a time step can temporally occlude other steps when integrating through time.

In Figure 3(c) and 3(d), they were rendered with an additive composition technique. In our previous two examples first hit and alpha compositing, when several time steps occupied the same region it was usually dominated by one single time step. When we use this technique, when several time steps overlap the same region of space it appears as white. For example in Figure 3(c), we have used the same vortex data set. Previously, when two or more time steps occupied the same position in space, only one time step was really discernable. Now, when all time steps overlap in space, they appear as white, or possibly appear as cyan, magenta, or yellow, if only a few time steps occupy the same space. We can see near the left hand side on Figure 3(d) that there is a great deal of overlap in time, so this means isosurface is relatively unchanged in this region over the time sequence. However, near the right hand side of the image, we can see distinct time steps, meaning there is significant progression of the isosurface. This is very useful in seeing overlap between time steps, and whether the isosurface has remained in the same relative region over time.

3.1.4. Minimum/Maximum Intensity

Visualizing the min/max values throughout the entire time sequence for every voxel in the volume in a single picture allows us to pick out the extreme value over the complete time series easily. When coloring each voxel in the chronovolume by the time step that has the min-max value, it allows the user to see changes over time. The user can see, in a particular region, which time step had the extreme value over a certain time sequence. As we integrate through time at a position, we select the time step that had the extreme value over the entire sequence. That time step and value is used to look up the value in the transfer function and becomes the value for the voxel at that position in the chronovolume.

Figure 3(e) shows the maximum intensity technique. With the same color transfer function over time that we have been using, where blue is the oldest and red is newest, a blue area indicates that the value has decreased over this time sequence. Likewise, a red area indicates that the value increased over this entire time sequence, and a green area indicates that the value increased then fell over time. With minimum/maximum intensity, at a glance we can see over the time sequence the general change in value, and which particular time step it was that had the maximum or minimum value. Granted, we do not detect and display multiple changes over time for the user; at this point we make an assumption of a single rise and fall in value.

3.1.5. Edge Enhancement and Other Feature Enhancements

We can also apply feature enhancement to our time integration techniques. Edge enhancement has proven to be a valuable tool to scientific visualization. To apply a feature enhancement, in our rendering pipeline, we do the detection and enhancement during the time integration step. To accomplish this, there are additional parameters to the transfer function. In previous examples, the transfer function had only been dependent on the data value and time ordinal. Edge enhancement adds gradient to the transfer function and silhouette edge enhancement also takes into account the view vector. For example, we have used silhouette enhancement in conjunction with alpha blending through time. As we integrate through time, we do a silhouette edge detection at every time step and apply the enhancement to the voxel for that particular time step and position. Note that this means when there is a change in viewer position, when we are doing silhouette enhancement, time integration must be recomputed, because the transfer function depends on the view vector in relation to the gradient for every time step. If we used basic edge enhancement, it would not be necessary to recompute the time integration when view position changes, because the gradient for a position in time and space is does not change with the view.

Figure 3(f) shows an application of silhouette enhancement over 3 time steps. Without the silhouette enhancement, not a great deal of detail can be seen on the surface. When silhouette enhancement is turned on in Figure 3(f), now we can see some additional information. In this data set we can see how the surface details change and move as time progresses. While the user has the ability to rotate the volume interactively, since we're combining many time steps together in one viewable volume, the user may need some aid to determine spatial depth while looking from a certain view. Silhouette enhancement can aid the user to determine depth in these instances.

3.2. Data Organization

In preparing the time varying volume for time integration and display, the data needs to be reorganized in an optimal manner for time integration. Traditionally, when a time series data is being stored, it is usually stored in a *xyzt* fashion, where *x* runs the fastest and *t* runs the slowest. In building a chronovolume, there are sequential accesses to consecutive data values in the same position but with varying *t*. This does not utilize spatial cache coherence, because if we integrate through time, there will be large address jumps in the array to access temporally sequential values.

For maximum efficiency in a software time integration, we order the data in a *txyz* fashion, where *t* runs the fastest. To minimize memory usage, instead of storing the entire time varying data set, we can store linear arrays of a single fixed positions with varying t. Thus, we only need to keep time steps of interest in core. Also in this manner, we only need to incrementally add or subtract relevant time steps to the array as needed, when the user wishes to move through time. The drawback to this is there are many linear arrays, which may need to be managed by the dynamic memory manager if the maximum length of time sequences is not fixed.

When using hardware rendering, it is more efficient to store the data in a *xytz* fashion, where *x* runs the fastest. The reason for this is because of efficient 2D texture operations in graphics hardware. Thus, when doing time integration, we use 2D texture slices that contain all *x* and *y* for a fixed *t* and *z*. Again, we only need to keep relevant time step textures in memory that are needed. In the following, we describe the hardware implementation of time integration.

3.3. Hardware Acceleration

If the integration through time is done is software, an update in the transfer function can be slow to display. We feel that interactive control is an important aspect of visualization software. To be able to give the user interactive control over the transfer function and immediate feedback, we can speed up the time integration by using graphics hardware.

To create a chronovolume using graphics hardware, we compute a 3D texture in *z* slices. To compute a *z* slice, in the case of alpha blending, for all time steps with the same *z* position, we alpha blend 2D textures into a pbuffer. When a texture is blended with the pbuffer, we are computing a portion of the time integration for some *t*, for all positions (x,y) with the same *z*. When all time steps are blended, the pbuffer is copied into the appropriate *z* slice in the final 3D texture, which is the chronovolume.

We utilize multitexturing units and dependent texture lookups. For our set up, we used a nVidia GeForce4 Ti4200, OpenGL 1.4, and nVidia's NV_TEXTURE_SHADER OpenGL extension. As mentioned in the previous section, we order the data in an *xytz* manner, that way we can use 2D one-component data textures. One texture unit is used for the data, and a second texture unit is used as a dependent texture in which the data transfer function is stored. When texture coordinates are generated, the data texture output is used as texture coordinates to the data transfer function texture to lookup the transfer function value for that data value. Only two texture units are used; in the GeForce4, that leaves two more texture units free to use for additional information like gradient textures. The time transfer function value for a texture is applied by setting the modulation color for the entire texture quad, because when the texture is rasterized, *t* is constant across the texture.

```
Activate a pbuffer as the render area
Turn blending on
Set the blend function
Bind the data transfer function
  dependent texture to texture unit 1
Pipe the output of texture unit 0
  to the input of unit 1
Pipe the output of texture unit 1
  to pbuffer
Set texture unit 0 as the
  active texture unit
for(all z) {
  Clear the pbuffer
  for(all t) {
    Bind data texture (t, z)
      to texture unit 0
    Set the quadrilateral color
      to time transfer function (t)
    Render an orthographic
      quad to pbuffer
  }
  Copy the pbuffer to slice z
    of the 3D texture chronovolume
}
```

With this, we can quickly integrate over time and create our chronovolume. We have implemented time alpha compositing and additive color in graphics hardware. For additive color, the time transfer function value also needs to be premultiplied by a normalizing factor to account for color clamping in the graphics hardware and the normalization step for additive color. Additionally, the blending function needs to be set appropriately to do addition rather than alpha blending. We have not implemented minimum/maximum intensity and first hit in hardware, but an outline follows for an implementation for graphics hardware. First of all, the data would need to be stored as a two-component texture, where the second component carried what time step the data value was from. To achieve min-max intensity or first hit, we rasterize data textures to the pbuffer, without using the transfer function, utilizing the stencil buffer to cull away data fragments. The stencil buffer would serve as a mask that stored either a boolean value for first hit, or the most extreme value

Data	*Size*	*Software*	*Hardware*
vortex	10x128x128x128	3.826 s	.1986 s
	20	7.536 s	.2649 s
	30	11.426 s	.3440 s
jet	10x128x128x128	3.810 s	.1832 s
	20	7.563 s	.2322 s
	30	11.412 s	.2990 s
dens	10x512x64x64	3.789 s	.1638 s
	20	7.457 s	.2156 s
	30	11.318 s	.2795 s
delta	10x128x128x64	1.895 s	.0905 s
	20	3.735 s	.1148 s
	30	5.659 s	.1430 s

Table 1: *The computation time for time integration in software compared to hardware. Computation time is expressed in seconds.*

for min-max intensity. If an incoming data fragment does not pass the stencil test, then it is not written to pbuffer. The pbuffer would be then to used as the data texture. This data texture would be used to do two dependent texture lookups, the data transfer function texture and time transfer function, also stored as a texture. The two texture fragments then would be modulated together using register combiner hardware, via NV_REGISTER_COMBINERS extension to produce a fragment in the z slice of the chronovolume.

For hardware acceleration, we get quite a significant speed up for time integration. In our hardware setup, we used an Athlon XP 1700, 512 MB, and a nVidia GeForce4 Ti4200 64 MB. For 10 time steps, we get about 19 times speed up of calculating the time integration with hardware compared with software time integration. When 30 time steps are integrated, a constant overhead involved in the graphics rasterization is less of a factor in the time taken, and we get approximately 39 times speed up of the time integration. The timing results can be seen in Table 1.

4. Parallel Views and User Interaction

When exploring a data space, we have found it to be useful to visually present and switch between parallel views of the rendering, with different time integration methods, to get a more complete understanding of what is taking place over the time series. In our use, it was not uncommon to switch between the additive method and the alpha compositing to find overlapping areas in time. We frequently reversed the time integration order so we could see both images of newest to oldest and oldest to newest to investigate the differences between the two. The sum of the parts is greater than the parts individually when the user is presented with the various time integration methods.

For example, in Figure 3(g), the isosurface moves from blue to green to red over 3 time steps. We get a good sense of motion and the actual distance in space between isosurfaces over time at a glance. When we switch to an additive view, as seen in Figure 3(h), we can see exactly how much the time steps overlap in space. Figure 3(g) gives us a sense of the amount of movement, and 3(h) gives us a sense of the amount of overlapping occupation. If we switch to our third view, Figure 3(i), we can see how the values are actually changing over time and what regions are contributing to the movement of the isosurface.

We have also found that when the user is altering the transfer function, fine tune control is very necessary when manipulating the alpha transfer values in chronovolume rendering, even more so than traditional raycasting. Since multiple time steps contribute to a single voxel now, opacity builds up very quickly on a single voxel when using alpha blending, which leads to a very opaque chronovolume where interior and depth features are obscured. To remedy the problem, the user must be given fine tune control, or the ability to “zoom in” on the transfer function, to edit the function at a very small scale. In addition to micro control over alpha values, since many time steps can still be in the same space in chronovolume, a necessary tool to give the user is a slicing plane to examine the interior of features that have progressed.

5. Conclusion and Future Work

We have developed an algorithm for rendering time varying data sets. It is an exploration of direct rendering of space-time data and different from previous methods of using animations and viewing individual time steps. The immediate benefit that we see in this system is that for data the surrounding context of adjacent time steps is immediately viewable. In both animation and the individual time steps, the viewer has to shuffle forward and backward through time to see the transition that is taking place. With the entire time sequence at once, the viewer can see what is taking place over space and time in a single frame and view the space-time boundary of features. We can also explore other features such as change in value over time and spatial overlap quite easily. With hardware acceleration, the user has the ability to interactively change the transfer function to explore the domain more readily.

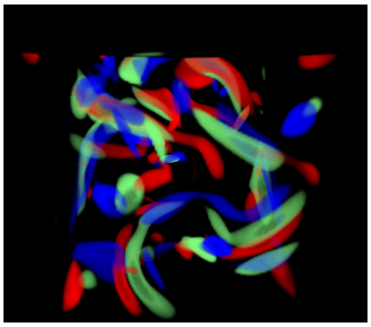
There are possibilities that could even further improve chronovolume visualization. First of all, improvements can be made to the data management schemes. There needs to be work done on managing large time varying data sets in relation to this rendering scheme and keeping the ability to interactively change the transfer functions. First and foremost, there is the issue about shuttling textures in and out of the main memory and texture memory. Some methods that could be used to reduce the traffic would be compression and incremental update.

Next, there can be improvements made to the user interactivity of the software. The user interface needs to be modified toward this new 4D rendering to allow the viewer to move quickly through the entire time sequence and select the areas of interest. Some measurement tools and probes that are adapted to working in this multi time space would also be beneficial to the user.

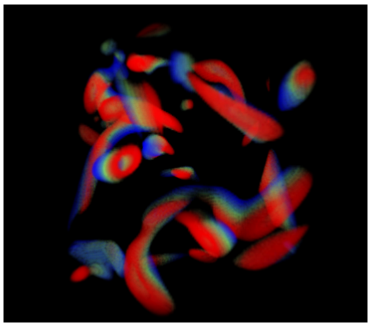
Finally, there can be enhancements made to the rendering capability more than just silhouette and edge enhancement. Other rendering methods that are useful to visualizing time varying data could be discovered. When so many time steps contribute to one image there tends to be quite a bit of clutter in the final image if the user is not careful with manipulating the transfer function. Better integration and transfer functions and more enhancements to reveal important data would be useful to the end user.

References

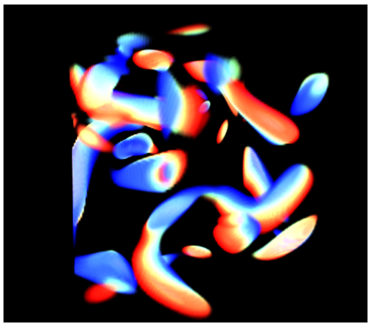
1. M. Braun. *Picturing Time: The Work of Etienne-Jules Marey*. The University of Chicago Press, 1992.
2. S. Guthe and W. Straßer. Real-time decompression and visualization of animated volume data. In *Proceedings of Visualization '01*, pages 349–356. IEEE Computer Society Press, Los Alamitos, CA, 2001.
3. E. Lum, K. Ma, and J Clyne. Texture hardware assisted rendering of time-varying volume data. In *Proceedings of Visualization '01*, pages 263–270. IEEE Computer Society Press, Los Alamitos, CA, 2001.
4. B. Sohn, C. Bajaj, and Siddavanahalli. Feature based volumetric video compression for interactive playback. In *Proceedings of 2002 Symposium on Volume Visualization*. ACM SIGGRAPH, 2002.
5. N Neophytou and K. Mueller. Space-time points: 4d splatting on efficient grids. In *Proceedings of 2002 Symposium on Volume Visualization*. ACM SIGGRAPH, 2002.
6. H.-W. Shen and C.R. Johnson. Differential volume rendering: A fast algorithm for flow animation. In *Proceedings of Visualization '94*, pages 188–195. IEEE Computer Society Press, Los Alamitos, CA, 1994.
7. H.-W. Shen, L.J. Chiang, and K.L. Ma. A fast volume rendering algorithm for time-varying field using a time-space partitioning (TSP) tree. In *Proceedings of Visualization '99*. IEEE Computer Society Press, Los Alamitos, CA, 1999.
8. D. Ellsworth, L. Chiang, and H.-W. Shen. Accelerating time-varying hardware volume rendering using tsp trees and color-based error metrics. In *Proceedings of 2000 Symposium on Volume Visualization*. ACM SIGGRAPH, 2000.
9. K. Anagnostou, T. Atherton, and A Waterfall. 4d volume rendering with the shear warp factorisation. In *Proceedings of 2000 Symposium on Volume Visualization*, pages 129–137. ACM SIGGRAPH, 2000.
10. P. Sutton and C. Hansen. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (t-bon). In *Proceedings of Visualization '99*, pages 147–153. IEEE Computer Society Press, Los Alamitos, CA, 1999.
11. H.-W. Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In *Proceedings of Visualization '98*, pages 159–166. IEEE Computer Society Press, Los Alamitos, CA, 1998.
12. D. Silver and X. Wang. Tracking and visualizing turbulent 3d features. *IEEE Transactions on Visualization and Computer Graphics*, 3(2), 1997.
13. R. Samtaney, D. Silver, N. Zabusky, and J Cao. Visualizing features and tracking their revolution. *IEEE Computer*, 27(7):20–27, 1994.
14. D. Bank and B. Singer. A predictor-corrector technique for visualizing unsteady flow. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):151–163, 1995.
15. T. van Walsum, F. Post, D. Silver, and F Post. Feature extraction and iconic visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):111–119, 1996.
16. A. Hansen and R. Cross. Interactive visualization methods for four dimensions. In *Proceedings of Visualization '93*, pages 196–203. IEEE Computer Society Press, Los Alamitos, CA, 1993.
17. A. Hansen and P. Heng. Illuminating the fourth dimension. *IEEE Computer Graphics and Applications*, 12(4):54–62, 1993.
18. C. Bajaj, C. Pascuci, G. Rabbiolo, and D. Schikore. Hypervolume visualization: A challenge in simplicity. In *Proceedings of 1998 Symposium on Volume Visualization*, pages 95–102. ACM SIGGRAPH, 1998.
19. G. Kindlmann and J. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of 1998 Symposium on Volume Visualization*, pages 79–86. ACM SIGGRAPH, 1998.
20. J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of Visualization '01*, pages 255–262. IEEE Computer Society Press, Los Alamitos, CA, 2001.
21. T. Jankun-Kelly and K. Ma. A study of transfer function generation for time-varying volume rendering. In *Proceedings of Volume Graphics Workshop 2001*, pages 51–65, 2001.



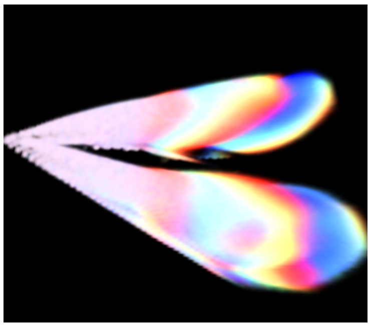
(a) Alpha composition over 3 time steps selected over 20 steps of the Vortex data set. Time steps are picked spaced apart so that they do not overlap, but progression over time can still be seen.



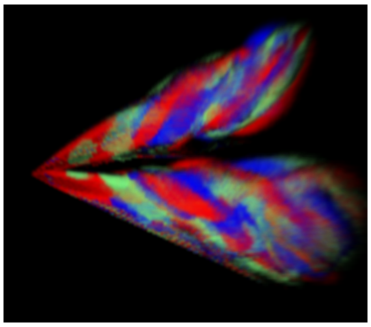
(b) First hit integration over 10 consecutive time steps of the Vortex data set. A time step has a low opacity and with first hit, the time steps are distinct and can be seen through spatially.



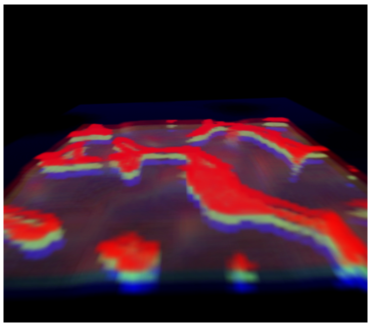
(c) Additive integration over 10 consecutive time steps of the Vortex data set. White regions indicate where all time steps intersect the same spatial location.



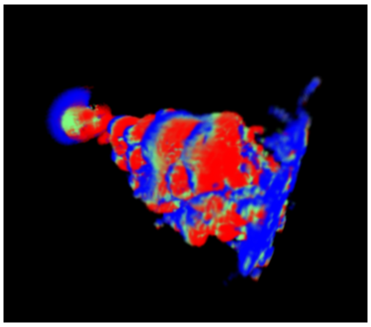
(d) Additive integration over 10 consecutive time steps of the Delta wing. White regions indicate where all time steps intersect the same spatial location.



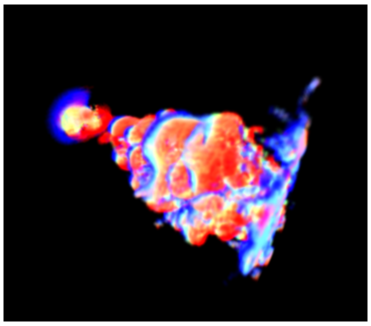
(e) Maximum intensity integration over 3 time steps selected over 10 of the Delta wing. Coloration indicates which regions had the maximum value over the time series, and thus indicates data value change over time.



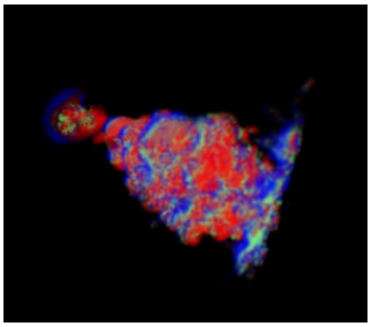
(f) Alpha blending with silhouette edge enhancement over 3 time steps selected over 30 of the Dens data set. The surface features can be seen on the planar surface as it progresses over time.



(g) 3 time steps of the Jet data set with time alpha compositing. The volume indicates that the data was in the blue region, moved to green, and then red.



(h) 3 time steps of the Jet data set with time additive integration. White regions indicate that several time steps occupy the same region in space over the series.



(i) 3 time steps of the Jet data set with time maximum intensity integration. This is an indication of the change in value over time in the time series.

Figure 3: *Chronovolume images.*