Incorporating Highlighting Animations into Static Visualizations

Jonathan Woodring and Han-Wei Shen

Ohio State University, Columbus, Ohio

ABSTRACT

Rendering a lot of data results in cluttered visualizations. It is difficult for a user to find regions of interest from contextual data especially when occlusion is considered. We incorporate animations into visualization by adding positional motion and opacity change as a highlighting mechanism. By leveraging our knowledge on motion perception, we can help a user to visually filter out her selected data by rendering it with animation. Our framework of adding animation is the animation transfer function, where it provides a mapping from data and animation frame index to a changing visual property. The animation transfer function describes animations for user selected regions of interest. In addition to our framework, we explain the implementation of animations as a modification of the rendering pipeline. The animation rendering pipeline allows us to easily incorporate animations into existing software and hardware based volume renderers.

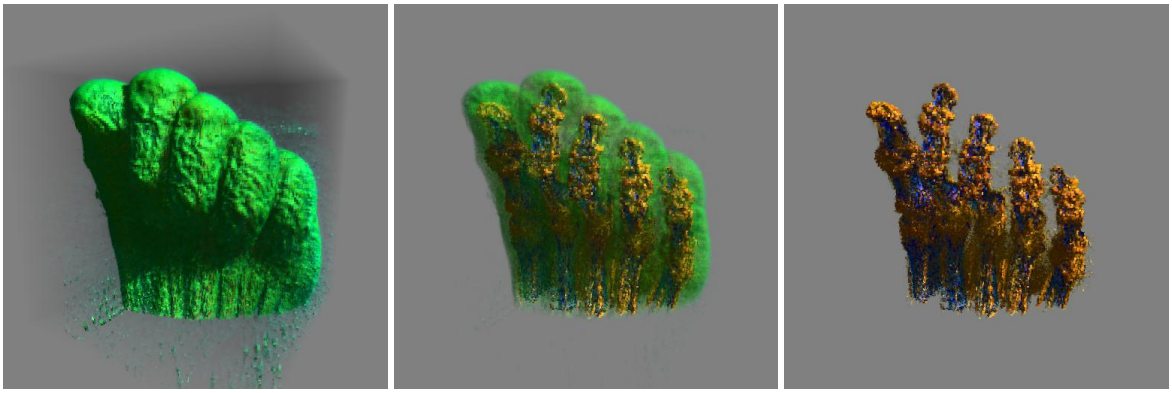

Figure 1. Animation of the opacity of a foot data set.

1. INTRODUCTION

Given a visualization, our goal is to allow the user to render selected focus data in context with an entire data set. Additionally, we want to render the focus data and the context data at a similar level-of-detail. The reason that we want to render both selection sets is to allow the user to see how data compare between focus and context values and positions. The main problem is visual clutter and occlusion occurring from the amount of data that is rendered at once. This is particularly the case in multi-field rendering where many volumes are combined together into one visualized volume. Also, user confusion can occur when she is not able to tell what data belongs to her selection set or which is the context data. The user may not be able to even find her data selection in the visual search.

We need some method to highlight user selected data in contrast with the rest of the data. To reduce visual clutter and emphasize regions, the current method of highlighting data is to reduce the opacity of surrounding data and ramp up the opacity of the selected data. This violates one of our requirements where we wish to render the context at the same level-of-detail. An alternative is that we can use the color transfer function to highlight selected data values. We feel that we can do better than this for the color channel is already overloaded in usage.[1,2]

Furthermore, using the color channel for highlighting does not address occlusion problems. Current visualization techniques solve occlusion problems by reducing the amount of data that is rendered at once. This is done by completely

Jonathan Woodring: woodring@cse.ohio-state.edu Han-Wei Shen: hwshen@cse.ohio-state.edu

Visualization and Data Analysis 2007, edited by Robert F. Erbacher, Jonathan C. Roberts, Matti T. Gröhn, Katy Börner
Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 6495, 649503, © 2007 SPIE-IS&T · 0277-786X/07/$18

removing the occluding data in question, through cutting planes and opacity reduction. This again violates one of our requirements where we wish to show the contextual data with the focus data.

We explore other avenues for highlighting rather than using static opacity reduction or color transfer function manipulations. The human visual system has evolved such that we can easily perceive motion and change in our environment. Even at a standstill, we use self-motion to get a better understanding of our surroundings. We currently use animation in visualization for time varying phenomena and it is quite natural to use in these instances. Users can observe time changing features that might not be immediately apparent from a set of static images, but become overwhelmingly obvious when the same set of images are seen in an animation. We feel that human motion perception capabilities are underutilized in visualization.

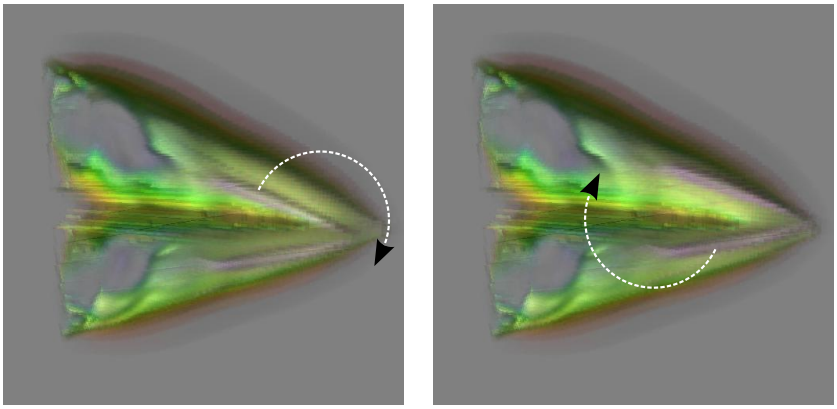

Figure 2. Two frames of an animation where a selected region in the data set moves around in a circular fashion.

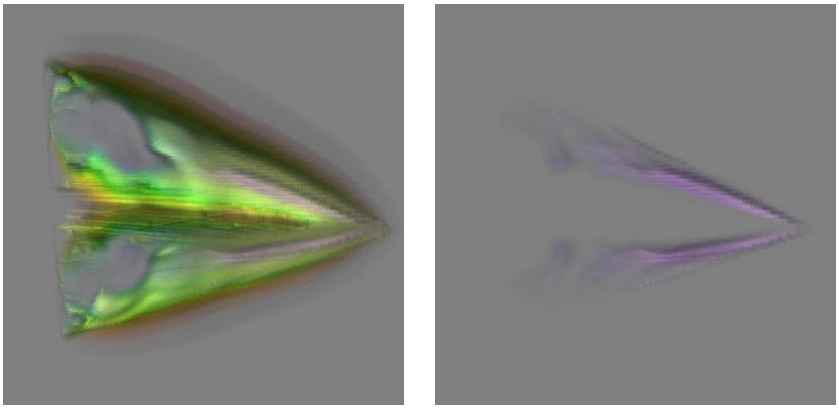

Figure 3. The opacity of the data set changes over the course of an animation, revealing the data that the user is interested in.

In Figure 2, we can highlight a region of data the user is interested in by moving the selection on the screen. Motion helps highlight objects by making them stand out, and the user can immediately find the data she is interested in. If this was a static image, it might take some time for the user to locate the regions that satisfy her query. This speeds up the visualization interaction time because motion is preattentive, and it immediately pops out to the user.[3] A simple analogy describes this: a deer that stands still in a forest is hard to see, but once the deer moves it becomes immediately obvious. In particular, Bartram et al. noted in a study on using motion as a filter, that some users stated that non-moving entities "fell out" of the frame.[3]

Furthermore, the user might not be able to locate the regions without motion all because of clutter, blending, and occlusion. As a side-effect of moving data, studies have shown that the user can make better decisions when given spatial

data in motion, rather than using only static perspective or even static stereo rendering.[4] The change in lighting and shading through object movement and light movement can be key perceptual indicators for shape and curvature.[5,6]

We can also animate the opacity function so that interesting regions are revealed. Figures 1 and 3 are examples where the data that the user is interested in is held at a constant opacity, while the rest of the data set has its opacity fluctuated over the course of the animation. By interactively revealing the user selection over time, she can get a sense of how her selection relates to the context of the data set. This is along with the user actually being able to locate her data by deoccluding it.

The unifying concept in our approach is an animation transfer function (ATF), which is similar to a color transfer function. The ATF maps data and animation frame indices to changing visual properties. To incorporate animation into the visualization, the user chooses different data regions by value or space, and also selects the animations that are applied to the data regions. As the visualization animation elapses, the position and/or opacity of selected regions change over time, providing highlighting through animation. Our ATF framework describes how the mapping of the animations is performed. For the implementation, we show how we can change the visualization pipeline to incorporate our highlighting animations into visualization. This change to the rendering pipeline can be easily incorporated into existing implementations without radical redesign of the pipeline.

2. RELATED WORK IN VISUALIZATION

The basis of augmenting visualizations with animation extends the transfer function. Transfer functions in visualization were first detailed by Levoy.[7] Our transfer functions will utilize an additional input parameter, in addition to a data point, which is the index for an animation frame. Data vectors and animation frame indices will map to visual property vectors.

There are numerous papers related to animation in scientific visualization, and it would be difficult to list all of them. Most are related to time-varying visualization and the acceleration of animation. Some past research has used animation in visualization outside of time-varying data. ISA, IBFVS, and related techniques developed by vanWijk and Laramee use animated textures to show flow in vector fields.[8] Reeves introduced particle systems, which has been used as a technique for visualizing vector fields through the animation of points and point sprites.[9] Lum and Ma have used particle animation to generate visualizations that show structure and shape from visual motion.[10]

Recently, there has been an interest in the role of perception in scientific visualization. Weiskopf recently examined the use of color and perception in motion.[11] Huber and Healey performed a user study on the perception of motion in visualization in relation to flicker, direction, and velocity.[12] Related to our work, Correa and Silver created transfer functions that animate a path or data set traversal.[13] The technique locally alters the transfer function during animation to highlight features such as vascular flow.

3. BACKGROUND INFORMATION IN MOTION AND PERCEPTION

In order to justify our work, Bartram and Ware have a multitude of studies showing the use and effectiveness of motion in information visualization.[14] Motion can enhance visualization for multiple reasons.[1,2,15] Motion is perceived at a low-level that is done in parallel with other perceptual tasks. It is preattentive such that the perception does not take an active mental effort.

Other graphical channels have a finite levels of granularity and they are already overloaded with information.[1,2] Only about 7 to 10 color hues can be fully distinguished in the narrow foveal range of vision. Studies have shown that color and shape are not easily conjoined for representation of values, and the search time can be significantly increased.[14] On the other hand, Nakayama and Silverman showed that motion can group moving elements together allowing users to search moving groups individually.[16] This would allow users to conjoin search, such as looking for particular colors in a moving group. In examining moving objects for a particular shape or color, the search is linear in the number of objects.[3,17] Driver et al. also had similar results, showing that objects can be grouped together using coherent oscillating motion patterns.[18]

By using motion as a filter, selected values with motion will stand out.[3] We are mostly shape and color blind outside of our foveal range.[19,20] The use of motion allows objects to be more readily detected outside of the fixation area. In a study where there were different types of alert mechanisms to a main task, motion is detected at a quicker response time and with fewer errors, compared to shape and color.[17,20]

Motion also helps in determining form, such as shading changes through movement can be used to interpret shape and curvature.[5,6] The kinetic depth effect allows us to determine structure and shape through the movement of an object in

space.[10] In a study where users had to make decisions about 3D graphs, the graphs in motion with perspective rendering had fewer errors and quicker response times than static perspective or stereo views.[4]

There is guidance on what types of motion are appropriate. Simple motions are preattentive, while more complex motions require effort to filter moving objects.[3] Differences in detection and error rates are based upon the amplitude of the motion, while phase and frequency have no appreciable effect.[17,20] There is evidence to support that there is granularity in the frequency of the motions, where up to four to six frequencies can be distinguished. There is a practical granularity of two in phase difference where motions with 90 degree phase are easily distinguishable from each other.[1,17,20]

4. ANIMATION TRANSFER FUNCTIONS

The animation transfer function framework supports many types of change in visual appearance beyond motion, such as change in color, texture, opacity, and lighting. We will primarily focus on animating position and opacity after we introduce our framework and pipeline. Animations for the paper can be located at **http://www.cse.ohio-state.edu/∼woodring/ATF** and the use of motion and opacity animation is described in a later section. Please forgive the quality of the animations. The animations were generated by a prototype system with nearest neighbor filtering, so the animations may seem grainy at times. Additionally, the capture system to record the movies also interferes with actual frame rate of the the real system.

A transfer function maps points from one space to another space. The most familiar of these is the color transfer function where there is a mapping from computational space to color space. This is usually implemented as a color lookup table. In rendering, there is also an additional, usually implicit, positional mapping from data coordinates to screen coordinates. This can be thought of as a positional transfer function or space transformation. To support animation, we will provide an animation transfer function. The animation transfer function will partially replace the opacity or classification transfer function and the positional transform. Later, we will describe how to incorporate our highlighting animations that do not create a complete reimplementation of a rendering pipeline.

The ATF is a function, $\mathbf{A}$, that maps a data set vector, represented by a function $\mathbf{D}(\hat{x})$, and a animation frame index, t, to a visual property space, a vector $\hat{v}$, seen in the following Equation 1. $\mathbf{D}(\hat{x})$, parameterized by $\hat{x}$, generates vectors that represent the data values of a data set, such as field values, color, opacity, position, gradient, etc. $\hat{v}$, the mapped vector for a particular $\mathbf{D}(\hat{x})$ and t, can have many different animated attributes, such as color, position, and opacity. It represents the appearance of data on a particular animation frame. We will have the ATF generate opacity and positions for data values so that we can provide motion and opacity animation. This means that the vector v is a two-tuple containing (a, p) where a is the alpha value and p is the position for a data point $\mathbf{D}(\hat{x})$ at animation frame t.

$$\mathbf{A}(\mathbf{D}(\hat{x}),t) = \hat{v} = (a,p) \tag{1}$$

An example equation that animates the opacity for regions can be seen in Equation 2. The user specifies what values are to be animated, which is incorporated into the ATF description. For example, the user can select a data value range based on a **box** function. Those values of interest will have the same animation applied to it so that we provide a highlighting mechanism. Our ATF design for this animation will be that if the data set value is within the user **box** function, then we use opacity of the data point $\mathbf{D}(\hat{x}).a$ as the output opacity. Otherwise, if a data point is outside of the **box** function, its opacity is modulated by a sine function. The sine function is parameterized by the animation frame index to create our animation. The function will smoothly and periodically increase and decrease from one to zero to one, scaling the original opacity. Regions that the user is interested in, contained in the **box** function, will have a constant opacity, and all other regions will fade into and out of view. We do not animate the position in this instance so the position of the data point remains the same as $\mathbf{D}(\hat{x}).p$ in the output vector.

$$\begin{aligned} \mathbf{s}(v) =& (\mathbf{sin}(v)+1)*0.5 \\ \mathbf{A}(\mathbf{D}(\hat{x}),t) =& if : \mathbf{box}(\mathbf{D}(\hat{x})) = 0 \\ & then : (\mathbf{s}(t)*\mathbf{D}(\hat{x}).a, \mathbf{D}(\hat{x}).p) \\ & else : (\mathbf{D}(\hat{x}).a, \mathbf{D}(\hat{x}).p) \end{aligned} \tag{2}$$

Now we will show an ATF example that generates positional motion in a visualization. Equation 3 causes selected values of interest to orbit in planar circle around their original position. Like before, in this example, we let a user define a region of interest through a **box** function. If a value is not a selected, being outside of a **box** function, then its position will not change during the animation. We will use the input position of a data point for the output position for the function in that case. Otherwise, when value is within the **box** function, we give it motion by animating its position. The original position of the data point, $\mathbf{D}(\hat{x}).p$, is offset by a animated vector that is a point that moves in a planar circle. This is generated by sine and cosine functions that are parameterized by the animation frame index. As the animation progresses, it creates a circular offset pattern. Our actual implementation for circular motion is more complex than Equation 3, because we provide view aligned circular motion.

$$\begin{aligned}\mathbf{A}(\mathbf{D}(\hat{x}),t) = & if : \mathbf{box}(\mathbf{D}(\hat{x})) = 1 \\ & then : (\mathbf{D}(\hat{x}).\hat{a}, \mathbf{D}(\hat{x}).\hat{p} + (\mathbf{cos}(t), \mathbf{sin}(t), 0)) \\ & else : (\mathbf{D}(\hat{x}).\hat{a}, \mathbf{D}(\hat{x}).\hat{p})\end{aligned} \tag{3}$$

5. ANIMATION PIPELINE

We do not change the opacity of a point in the above case, nor did we change the position of a point at the same time of animation of the opacity. It is possible to combine the two into one animation, by joining the two ATFs into one ATF. Furthermore, it is possible for the user to define multiple regions of interest, and have different animations for each region. This leads us to our discussion on how to combine our animations with a practical implementation in our visualization pipeline. The previous section has given us a framework on how to animate data. Like with color transfer functions, we have a mathematical framework in describing how a transfer function works, but the actual implementation in the pipeline is usually performed with a color lookup table.

5.1. Single Field Animation Pipeline

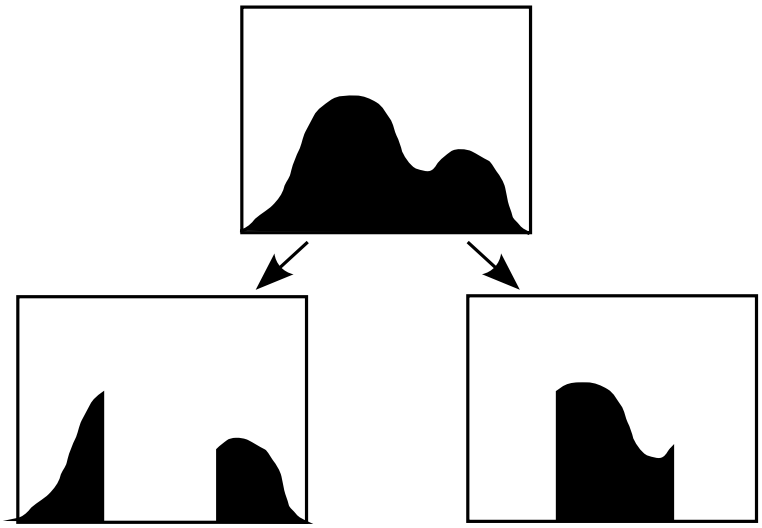

Figure 4. An opacity transfer function for a data set is split into two separate functions based on the user selection. A split opacity function defines the data points that have one user defined animation.

To explain our rendering pipeline, we use an example with single field data where the user selects one region of the data she wishes to animate, either by position or by opacity. Which remains there is some data that is not animated, so that there are two data selection regions. We assume that there is a opacity function that determines the opacity for data points in the data set.

For our example, the opacity function is separated into two opacity functions by the user selection, as seen in Figure 4. A split opacity function corresponds to the user selected data portion and the animation on the selection. Every data point will be sampled multiple times, once for each animation defined, but will only appear once in the final render. This is because we have separated the opacity function by animations and every data point only has positive opacity in one separate opacity function. Another way to think about this is that by dividing the opacity function, we split the data volume into separate rendered volumes which can be animated independently and composited together.

In our example, for the first split opacity function, the selected data points retain their original opacity value, while all other points are set to zero opacity. To create this, the opacity function is modulated with the function that defines the user selection to mask out the portion that is not animated. This creates one volume for that animation. In the other opacity function, the converse is true. The remaining points not selected by the user have positive opacity and the rest are transparent, which rendered is the volume containing the rest of the data.

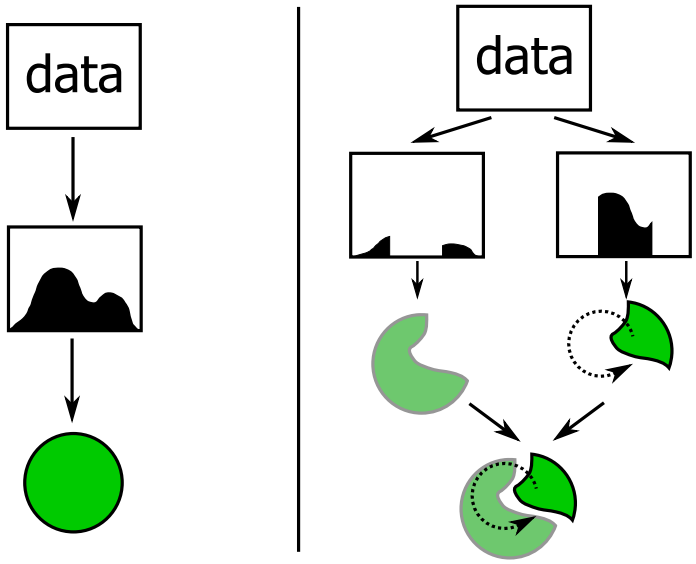

Figure 5. The traditional rendering pipeline is seen on the left. Data is sampled, the color and opacity transfer function is applied for each sample, eventually resulting in a visualization. To support animation, we split the opacity transfer function into different regions of animation, do multiple samples at a point in space, and applying animation to each the samples.

During rendering for our example, the data is looked up twice when sampling a region in space, once for each animation or opacity function. An example figure of our modified visualization pipeline is shown in Figure 5 on the right, where the traditional pipeline is shown on the left. We have divided the opacity transfer function into regions of different types of animation, which in this case divides the volume into two volumes. The animation that is defined for a data selection is applied to the corresponding sample which results in animating that volume portion.

Our main reason for dividing the opacity function is so that only data points that have a selected animation will a positive opacity for that animation. For each animation, we animate the position of the entire data volume, moving all data points for an animation. The data points that are selected to move by that animation have positive opacity, while all other samples are transparent. Therefore, we mask out the data points that do not correspond to a selected animation and only the region the user has selected is animated.

For opacity animation, we only need to animate the corresponding separate opacity function. In our example with the two color samples for the two animations, they are composited together to generate one sample at a point in space.[21–24] By doing this, we perform 3D composition of the animated volume portions, rather than 2D composition of images, to provide correct depth occlusion in the animation.

This can be extended to multiple animations simultaneously, by segmenting the opacity transfer function into multiple opacity transfer functions. Each one of the separate opacity functions corresponds to a user data selection and animation

of that region. We perform a data sample for each opacity function, so that if there are N animation regions then there are N opacity transfer functions and N samples at a point in space. The animation is applied for each sample and all of the samples are composited together for one final sample.

5.2. Multi-variate Animation Pipeline

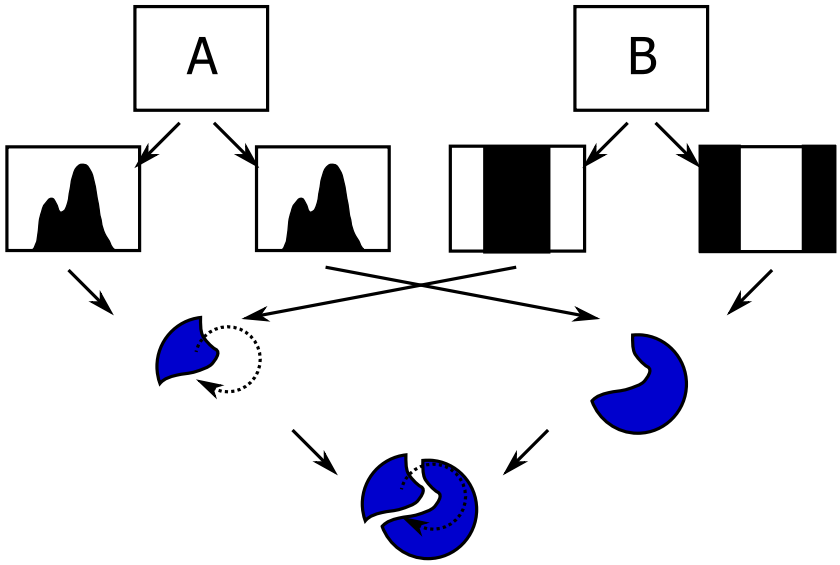

Figure 6. We have two data fields for multi-variate data, where the user wishes to show field A with color and animate regions based upon field B. Field A has a color and opacity transfer function, while field B has a constant opacity transfer function defined on it. The animation of field B segments B's constant opacity transfer function, and opacity samples taken from B are used to modulate the color and opacity of A.

We make a slight modification to our pipeline to support animation for multi-variate, time-varying, or comparative data where one field is represented by the color channel and another field is represented by motion patterns or opacity animation. Examples of this can be seen later in Figure 8 and 9. For our example in Figure 6, we have two data fields, *A* and *B*. The color transfer function is used to represent the values and visualize field *A*. We will use the data in field *B* to animate the appearance of the field *A*. This way the user can see the conjoining of values of *A* and *B*, through motion and color. By inspecting animated data points, a user can see values in *B* by an animation pattern with the colors that are the values of *A*. The color transfer function and opacity function for field *A* in the animation pipeline does not change as it is defined by the user. For field *B*, we define an opacity function of one for the entire field, which is separated into multiple opacity functions for animation.

We need to perform the positional animation that is defined on *B* for *A*. If we do not apply the positional animation on *A*, then *A* does not appear to move at all. We have to positionally animate *A* as much as *B* is moved, because values of *B* are shown by the movement of *A*. We are applying a motion pattern from the values of *B* and transferring that to animate the apperance of field *A*. We sample *A* to obtain a color point and samples are taken from field *B* to obtain an opacity point. Likewise, the positional or opacity animation is applied to the sample from *B*. The opacity samples taken from field *B* are then modulated with the corresponding color sample taken from field *A*.

The modulation of opacity taken from *B* with the color of *A* creates an intersection volume. For our example, we are intersecting two alpha volumes from *B* with the two color volumes obtained from *A*. The alpha volumes of *B* are created from the separation of *B*'s opacity function by the animation selection. This opacity masks the animation of the color volumes from *A* with the alpha volumes from *B*. This is so that *A*'s animated appearance is only done with the selected data points specified on field *B*. Like the single value animation, we composite the samples together to form one sample at a point.

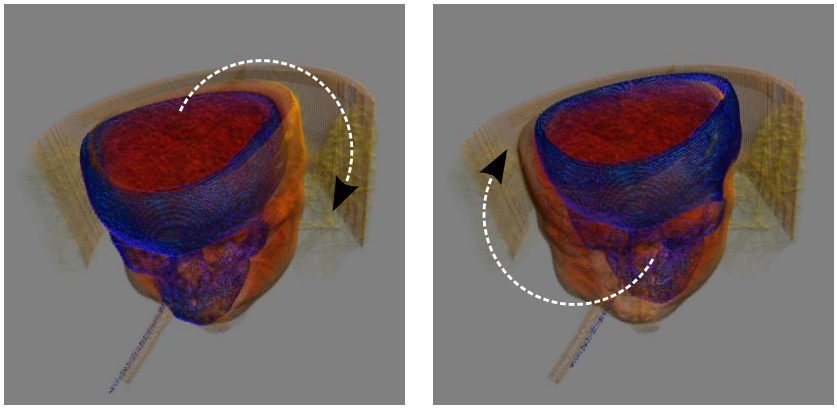

Figure 7. The selected skull region of the head moves in a circular motion, thereby highlighting the region of data.

6. MOTION HIGHLIGHTING

For highlighting using motion, the user makes a query by choosing a data range or spatial range of interest. The selected values are animated through the animation pipeline, as seen in Figures 2 and 7. Compared to a static data set without animation, the motion allows a user to find values without searching because the data pops out to the user.[3] Due to perceptual mechanisms, the selection that is in motion can be examined independently from the rest of the data.[16] This is because the motion provides a low-level visual filtering and grouping of moving and non-moving elements.[18] The filtering gives the user a focus + context visualization support of the selection, as she can compare the selection with the rest of the data.

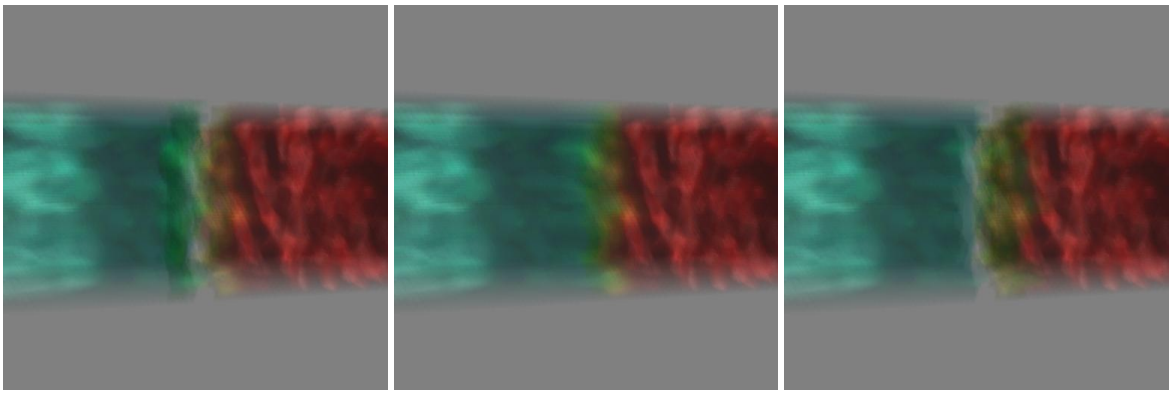

Figure 8. A value range in a previous time step (not shown) is animated in this time step. This shows where the data value occurred in the previous time step in relation to the current time step.

The user can use animation so that it highlights the difference between two data sets. For example, the user may compare two different simulation runs of the same initial condition or two different time steps in one simulation. An example is shown in Figure 8 where the user selects a value range in a previous time step. The corresponding spatial locations in the currently visualized time step are animated to show the correlation. Through the motion animation, the user does not have to visually search for the correlating points.[3] When there are dual viewports showing two comparative views, the data can be animated in both views. When animating the points in both views, the points would be visually grouped together perceptually.[18] The user is then able to examine the differences and similarities between the series of points in both views, and she is able to ignore the contextual data not in motion.[16]

Motion can also be used in highlighting multi-variate and time-varying data set analysis. Given a data set with several fields of information, one data field or time step can be depicted by motion or animation of the data points to highlight

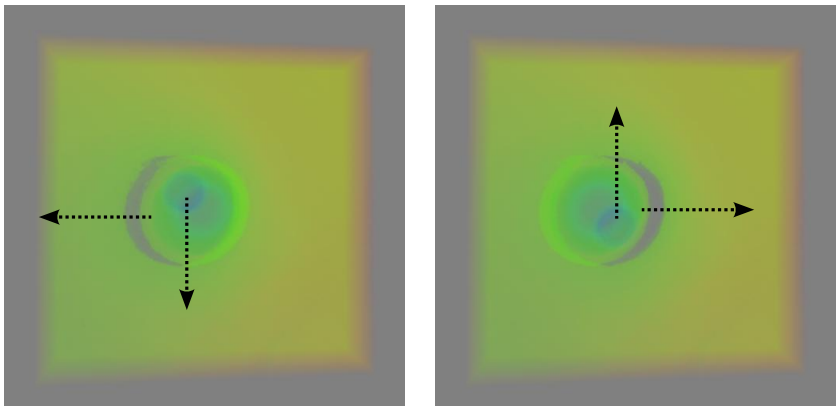

Figure 9. A two variable data set that has a pressure component and a cloud cover component. One variable is shown in color, while selected ranges in the other variable have motion.

that data. For example, in Figure 9, it is a data set with cloud cover and pressure values. The pressure is displayed via a color transfer function. Cloud cover value ranges are shown by animation with different motion patterns. For example, one cloud cover range has an oscillating up-down motion, and another range has a left-right motion. Since color and motion can be conjunctive, the user can find data points that satisfy both channels of information.[16] Users are able to accomplish this by highlighting the region of interest with motion. Then, her visual capabilities are able to perceptually filter out one moving group corresponding to a value range.[3,18] From the data that is in motion, the user is able to examine the colors for a movement pattern.[16] For example, the user could examine the up-down pattern corresponding to one particular cloud cover range, perceptually separating it from the context data. Then from that selection range, she is able to examine what pressure values are within that cloud cover range.

7. OPACITY HIGHLIGHTING

One of the biggest problems in scientific visualization is occlusion, because much of our data is three dimensional. Occlusion is a particularly a problem when combining multiple volumes together for multi-field and time-varying data sets.[21–24] By animating the alpha channel, we can provide an animated focus + context visualization, as seen in Figures 1 and 3. The user is interested in a particular data value range of the data set, but cannot see that value range fully or at all due to occlusion. Instead of completely removing the occluding data, we are able to show the data selection in relationship to the rest of the data set by animation. The opacity values for regions that are not the focus of the user are faded in and out of view over the course of animation. The period of the change can be altered to the user's liking determining how long she sees the entire data set, the focus values, and the length of the transition between the two. This animation allows the user to see the selected values, but also get a sense of how it relates to the rest of the data.

8. IMPLEMENTATION

Our test software is written using OpenGL 1.5 and the GLSL ARB fragment shader extensions. The hardware used is an AMD Athlon 64 3500+ with an nVidia 6800 GT 256MB. The frame rates varied from 5 fps to 20 fps depending on whether real-time shadows were enabled and the level of zoom. Our rendering speed is limited by the fill rate and texture memory size of the graphics card, and not by the animation. Data set sizes used in our tests ranged from 128x128x128, 256x256x64, to 128x128x64, with one or two variables. The volume rendering algorithm uses view aligned quadrilaterals and 3D texturing hardware. Up to four texture units are used; one to two for the data set and normals, one for the accumulating shadow buffer, and one for the color and opacity maps. The user selects a data range and and the type of animation to be applied to the selection.

The animation is mostly performed by the graphics hardware during the fragment processing stage. The techniques either use texture transform manipulation (position highlighting) or dependent texture lookup manipulation (opacity highlighting). For example in opacity highlighting, the separated transfer and opacity functions are uploaded as a dependent

texture. On every frame, the opacity transfer function is altered for effected regions and re-uploaded to the graphics hardware. This causes the selected regions to fade in and out of view by animation, by looking up the opacity in the dependent texture, which is animated by the CPU and uploaded to the graphics hardware on every frame.

When using positional motion to animate, simple, rigid motions are supported[3] which makes it easy to implement using texture matrices for positional offset sampling. Texture transform manipulation is used to implement rigid positional motions of the data this way. The opacity transfer function is separated into different motion regions. Motion on each frame is stored in one of the multiple texture matrices as an inverse offset mapping from the slicing quadrilaterals to 3D texture space. For a selected motion region to move, the texture transform for a region (**glMatrixMode(GL_TEXTURE)**) is changed according to the user's desired motion to highlight that data. The user's selection is masked by the separated opacity functions, and the actual motion is implemented by **GL_TEXTURE** matrix updates on every grame. So we are changing a transformation matrix on every frame for the graphics hardware to implement positional highlighting.

The new texture matrices are passed to the vertex program on each frame, one for every motion type, from which it generates multiple texture coordinates for a vertex. When rasterizing the slicing quadrilaterals, the fragment program performs multiple texture lookups for the animations on the volume data to acquire separate sample points. It performs a dependent texture lookup for each of the data points to get the color and opacity for every animation. The multiple samples are composited together to form the final fragment, which may be from multiple data points overlapping in space due to positional movement.

To determine the final fragment color, from the separated opacity functions or user defined regions, we use alpha blending to combine the multiple samples together. This is because for every fragment, we sample the volume X times for X animations, and combine the results together. There is one rendering pass, but X samples for X animations. Remember from the previous section, a data point only has one animation applied to it. Even though a data point is sampled multiple times, a data point only appears once in an animation because only one sample has a positive opacity due to the separated opacity functions.

In implementation, there is very little that has to be changed in the visualization pipeline in either case for opacity highlighting or position highlighting. This results very little data that has to be uploaded to the graphics hardware on every frame. We do not have to manipulate the original data set in any way, just the dependent texture and texture transformation matrices in the hardware implementation. In theory, for a ray caster, we would only need to animate the opacity function and provide offset to the ray(s) on every frame. We would need to provide multiple rays (a bundle of rays) for the separated opacity functions, and combine the results of the multiple rays, by alpha blending, like in the hardware implemenation.

Our implementation is viable for user interaction and real time updates. The ATF technique can also be used for pregenerated animations, but we feel that it is more effective if the user is allowed to turn off animations or switch to different animation types interactively, while still being able to change other visualization parameters. When this technique is layered upon other visualization systems, it is especially important that the user can interactively use the animation mechanisms freely with other techniques that she is familiar with.

9. CONCLUSION

Motion and animation is a powerful technique that is under utilized in visualization. The animation transfer function is our framework that is used to add highlighting animations to visualization. It builds upon the concept of the transfer function, where an ATF maps data values and animation frame indices to changing visual properties. In particular, we animate the position and opacity of data to highlight data selections. We have presented the framework of an ATF, but also the practical implementation of it in the visualization pipeline. This implementation is easy to integrate into existing pipelines, and is capable of real time animation of the data.

We use the animation for rendering data selections in context with the entire data. Through highlighting by motion, the user can pick out regions that she is interested in and focus on them. This is because motion is preattentive and can make things immediately pop out without search.[3] Highlighting by animation can also be used in multivariate and comparative analysis. Animation can also be used as a technique to deocclude and highlight unseen regions of interest. By fading context data out of view, the user can see the data she is interested in and how it relates to the context of the entire data set.

We have set up the general framework for animating static visualizations and there is room for future work in highlighting through animation. Our implementation uses simple, rigid motions, which is supported by the perceptual literature,

but possibly more deformable motions may have use. In Correa and Silver's animations,[13] they use space, continuity, and locality to enhance visualizations, and we would like to explore this avenue as well. User feedback may drive the animation as well, depending on user interaction with the animation and previous frames of animation. Finally, transfer functions primarily deal with local data values emerging into a global picture. It may be possible to use global analysis to generate animations.

REFERENCES

1. L. Bartram, "Can Motion Increase User Interface Bandwidth in Complex Systems?," in *Proceedings of 1997 IEEE Conference on Systems, Man and Cybernetics*, **2**, pp. 1686–1692, IEEE Computer Society Press, 1997.
2. L. Bartram, "Enhancing Visualizations with Motion," in *Proceedings of the IEEE Symposium on Information Visualization, 1998*, pp. 13–16, IEEE Computer Society Press, 1998.
3. L. Bartram, C. Ware, and T. Calvert, "Filtering and Integrating Visual Information with Motion," in *Proceedings of IEEE Symposium on Information Visualization, 2001*, p. 2002, IEEE Computer Society Press, 66–79.
4. C. Ware and G. Franck, "Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions," *ACM Transaction on Graphics* **15**, pp. 121–140, April 1996.
5. D. Akers, F. Lossasso, J. Klingner, M. Agrawala, J. Rick, and P. Hanrahan, "Converying Shape and Features with Image-Based Relighting," in *Proceedings of IEEE Visualization 2003*, pp. 349–354, IEEE Computer Society Press.
6. L. Wanger, J. Ferwerda, and D. Greenberg, "Perceiving Spatial Relationships in Computer-Generated Images," *IEEE Computer Graphics and Applications* **12**, pp. 44–58, May 1992.
7. M. Levoy, "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications* **8**, pp. 29–37, May 1988.
8. R. Laramee, J. van Wijk, B. Jobard, and H. Hauser, "ISA and IBFVS: Image Space-Based Visualization of Flow on Surfaces," *IEEE Transactions on Visualization and Computer Graphics* **10**, pp. 637–648, November/December 2004.
9. W. Reeves, "Particle systems–A Technique for Modeling a Class of Fuzzy Objects," in *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 359–375, ACM Press, 1983.
10. E. Lum, A. Stompel, and K.-L. Ma, "Kinetic Visualization: A Technique for Illustrating 3D Shape and Structure," in *Proceedings of IEEE Visualization 2002*, pp. 435–442, IEEE Computer Society Press, 2002.
11. D. Weiskopf, "On the Role of Color in the Perception of Motion in Animated Visualizations," in *Proceedings of IEEE Visualization 2004*, pp. 305–312, IEEE Computer Society Press, 2004.
12. D. Huber and C. Healey, "Visualizing Data with Motion," in *Proceedings of IEEE Visualization 2005*, pp. 527–534, IEEE Computer Society Press, 2005.
13. C. Correa and D. Silver, "Dataset Traversal with Motion-Controlled Transfer Functions," in *Proceedings of IEEE Visualization 2005*, pp. 359–366, IEEE Computer Society Press, 2005.
14. C. Ware, *Information Visualization*, Morgan Kauffmann Publishers, 2004.
15. L. Bartram, "Perceptual and Interpretative Properties of Motion for Information Visualization," Technical Report CMPT-TR-1997-15, School of Computing Science, Simon Fraser University, 1997.
16. K. Nakayama and G. SIlverman, "Serial and Parallel Processing of Visual Feature Conjunctions," *Nature* **320**, pp. 264–265, 1986.
17. C. Ware and R. Bobrow, "Motion to support rapid interactive queries on node-link diagrams," *ACM Transactions on Applied Perceptions* **1**, pp. 3–18, July 2004.
18. J. Driver, P. McLeod, and Z. Dienes, "Motion Coherence and Conjunction Search: Implications for Guided Search Theory," *Perception & Psychophysics* **51**(1), pp. 79–85, 1992.
19. L. Bartram, C. Ware, and T. Calvert, "Moving Icons: Detection and Distraction," in *Proceedings of the IFIP TC.13 International Conference on Human-Computer Interaction*, pp. 157–165, IOS Press, 2001.
20. L. Bartram, C. Ware, and T. Calvert, "Moticons: Detection, Distraction and Task," *International Journal of Human-Computer Studies* **58**(5), pp. 515–545, 2003.
21. J. Woodring and H. W. Shen, "Chronovolumes: A Direct Rendering Technique for Visualization Time-Varying Data," in *Proceedings of International Workshop on Volume Graphics '03*, pp. 27–34, 2003.
22. J. Woodring, C. Wang, and H. W. Shen, "High Dimensional Direct Rendering of Time-Varying Volumetric Data," in *Proceedings of IEEE Conference on Visualization '03*, pp. 417–424, 2003.

23. J. Woodring and H. W. Shen, “Multi-variate, Time-varying, and Comparative Visualization with Contextual Cues,” *IEEE Transactions on Visualization and Computer Graphics* **12**, pp. 909–916, September/October 2006.
24. W. Cai and G. Sakas, “Data Intermixing and Multi-volume Rendering,” in *Proceedings of EuroGraphics ’99*, pp. 359–368, 1999.