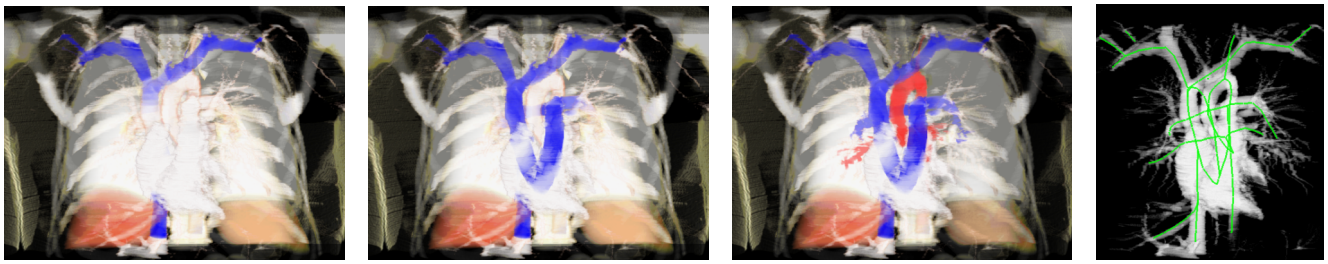
Dataset Traversal with Motion-Controlled Transfer Functions

Carlos D. Correa*and Deborah Silver†

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey
Piscataway, NJ 08855-0909



(a) (b) (c) (d)

Figure 1: Sequence of traversal of blood vessels in the visible human dataset as an illustrative visualization. Here, flow of blood is simulated via a transfer function. (a) Venous blood is represented as a blue transfer function coming through the vena cava. (b) The blood flows through the pulmonary artery. (c) A second transfer function in red represents arterial blood coming out through the aorta (d) Different paths used for traversal (in green).

Abstract

In this paper, we describe a methodology and implementation for interactive dataset traversal using motion-controlled transfer functions. Dataset traversal here refers to the process of translating a transfer function along a specific path. In scientific visualization, it is often necessary to manipulate transfer functions in order to visualize datasets more effectively. This manipulation of transfer functions is usually performed globally, i.e., a new transfer function is applied to the entire dataset. Our approach allows one to locally manipulate transfer functions while controling its movement along a traversal path. The method we propose allows the user to select a traversal path within the dataset, based on the shape of the volumetric model and manipulate a transfer function along this path. Examples of dataset traversal include the animation of transfer functions along a pre-defined path, the simulation of flow in vascular structures, and the visualization of convoluted shapes. For example, this type of traversal is often used in medical illustration to highlight flow in blood vessels. We present an interactive implementation of our method using graphics hardware, based on the decomposition of the volume. We show examples of our approach using a variety of volumetric datasets, and we also demonstrate that with our novel decomposition, the rendering process is faster.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

Keywords: Dataset traversal, illustrative visualization, volume manipulation, animation, transfer functions

*cdcorrea@caip.rutgers.edu
†silver@caip.rutgers.edu

1 Introduction

Although volume rendering is a very well researched field, the effective exploration and visualization of volumetric datasets remains a challenging area. One of the difficulties of exploring volume models is the simultaneous visualization of different structures while preserving the context, due to occlusion, clutter and noise. In this paper, we present a technique for illustrative visualization based on the exploration of a dataset via a transfer function that can be moved along a path. We call this *dataset traversal*, i.e., traversing a dataset along a specified path. This is commonly used in medical illustration to produce animations, highlight features or enhance the rendering of a dataset. For example, Figure 2 shows frames from an animation of arteriovenous malformations [2]. Arterial blood is highlighted as animated red arrows during the first frames of the animation, and the last frames show the flow of venous blood in blue. While this is extremely illustrative, one cannot rotate or slice this image. In this paper, we show how this type of technique can be implemented on 3D data with the ability to rotate and slice interactively.

The method we describe in this paper uses a skeleton path, an

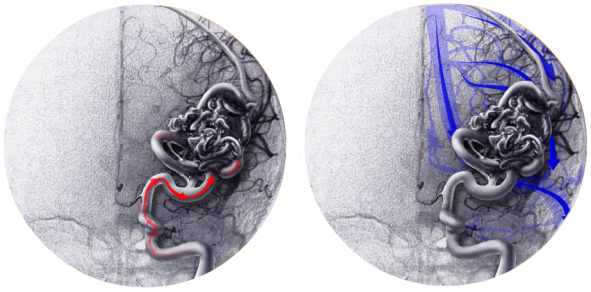


Figure 2: Two frames from a medical illustration animation of Arteriovenous Malformations. In the left figure, red arrows are used to animate flow of arterial blood. In the right figure, blue arrows represent venous blood (courtesy of Meaghan Brierley. Copyright ©, 2000, url: `http://brainavm.oci.utoronto.ca/swf/intro.html`)

abstraction of the volumetric model, as the control structure for traversal of the dataset. This enables the user to explore different parts of the dataset, while preserving the context of the visualization. Applications of dataset traversal are effective visualization of convoluted structures, such as aneurysms, or the simulation and animation of flow in vascular datasets. It can also be used for surgery and medical treatment pre-planning, or as a teaching tool, to allow a student to discover new features. There is also a basis in cognition that suggests that this type of traversal could aid in comprehension and shape understanding. Psychological evidence suggests that traversing a complex dataset seems to be an essential part of understanding 3D shape. The cognitive process obtains local explanations along the object as one traverses it to finally arrive at a global explanation [21]. Furthermore, skeletons are essential attributes of *geons*, simple and primitive volumes which are the basic elements of object recognition, according to Hummel and Biederman [12]. Techniques for navigation of datasets have been proposed for, e.g., virtual colonoscopy, where the user's viewpoint is traversed along a path. Such techniques can be described as *inside-out* (ego-centric) visualizations of the dataset. Here we propose an *outside-in* (exo-centric) visualization, where the exploration is enabled by moving a transfer function, in addition to independent control of the user's viewpoint.

2 Related Work

The methodology presented in this paper is related to a number of technologies in scientific visualization. These include spatial transfer functions [5], focus+context visualizations, interactive rendering using commodity hardware and virtual colonoscopy [11, 28]. As a visualization that conveys motion, it is related to *kinetic visualization* [18], where a particle system is animated along the surface of a static volumetric model to convey shape. In our approach, we help to convey structure with the traversal of a transfer function along the *interior* of the volumetric model.

This work relates to the *spatial transfer functions* defined in Chen et al.[5]. A spatial transfer function is defined as a function that applies a geometrical transformation of every point p in euclidean space. A *temporal spatial transfer function* is defined as a geometrical transformation over time. The work here can be thought of as a temporal spatial transfer function, applied along a defined path, or a "*motion-controlled transfer function*". This transfer function defines a scalar value, such opacity and color, as a function of time and the position of a voxel. In contrast, the spatial transfer function in Chen et al.[5] does not move the transfer function along a path, but rather applies a transfer function to the *transformed* voxels, according to some geometrical transformation.

The exploration of volume datasets sometimes requires the ability to visualize a focused portion of the volume while preserving some view of the rest of the volume (context). This is often known as *focus + context* visualization. There are many ways to provide this type of visualization, including segmentation [9], cutting planes, importance-driven [27] and selective volume rendering [23], illustrative techniques [3, 17], features maps [19], and warping techniques such as fish-eye views [14]. Cutting planes are very important in 3D visualization, but they can be difficult to position accurately, especially for convoluted structures such as an aneurysm. Mueller et al.[19] present an approach to modify the transfer function of a segmented region by migrating the density range of the region of interest. Volume exploration is possible through *feature maps*, which associate a set of interesting features with a particular density range, so that they can be isolated smoothly from other similar features. This approach becomes impractical for the interactive traversal of a transfer function, since a new region of interest must be computed at each frame. Furthermore, the scalability of this approach is limited to a few features, because of transfer function complexity.

Importance-driven rendering [27] provides an alternative for focus+context visualization for both segmented and non-segmented datasets. Each voxel is assigned an importance value, which translates to a given visibility priority during a two-pass rendering process. One can think of our method as a way of defining importance on a portion of the volume. And certainly, importance-based rendering can be implemented on top of this method.

In this paper, we decompose a volume to access it via a transfer function. There has been work in volume decomposition, most of which is for rendering speed-up and texture compression [15, 16, 22, 13, 20]. In our work, texture decomposition help us explore advanced rendering capabilities. Li and Kaufmann [15] and Li et al. [16] devise a mechanism to speed up the rendering process by decomposing the texture into axis-aligned boxes. The partition enables empty space skipping and occlusion clipping. Engel et al.[7] use a different approach to reduce the number of slices needed for rendering, using slabs rather than slices. Pre-integrated classification is used to improve image quality. Jankun-Kelly et al.[13] and Park et al. [20] use a k-d tree decomposition of axis-oriented boxes to speed up visualization of adaptive mesh refinement data. Unlike these approaches, we decompose the volume along a selected path. Because of the geometry decomposition along the bones, our approach enables the traversal of a transfer function at interactive rates, for segmented and non-segmented volumetric datasets.

The use of skeleton paths to guide traversal of datasets has been proposed for other applications, such as virtual navigation and virtual colonoscopy [11], also readily available as commercial products such as in GE [10], Siemens [25] and Viatronix [26]. In such applications, the path is a set of connected *centerlines* that are located at a distance from the surface appropriate for collision-free navigation [29]. This type of traversal can be described as an *inside-out* visualization, where the user's viewpoint is traversed along the volumetric dataset. In contrast, our approach is an *outside-in* visualization, where the transfer function is traversed along the path, while the user's viewpoint is independent of the traversal. Although centerline algorithms are usually required for skeleton determination, it is not a requirement for transfer function traversal. As an aid for visualization the path for traversal might be along certain structures of the volume, e.g., bones, vascular structures, and not necessarily along the centerline of the volume.

The use of skeletons has been proposed in Singh and Silver [23] and Singh et al. [24] to allow interactive manipulation of volumetric models. Singh[23] showed that it is possible to apply a different transfer function to a particular bone in the skeleton. Here, we generalize that approach and define any path as a control structure for guiding a transfer function. This can be combined with the ability to rotate and manipulate a volume as an aid to effective visualization.

The rest of the paper is organized as follows: Section 3 describes the theory for this work and the different processes involved in traversal rendering. We then describe different challenges of applying this process, such as obtaining smooth transitions and applying multiple traversals. Section 4 shows results of our approach with a number of volume datasets, as well as performance results. Section 5 describes how our approach can be used to provide focus+context visualization. Finally, Section 6 presents conclusions and future work.

3 Theory

This paper presents a methodology for dataset traversal using motion-controlled transfer functions. We define *dataset traversal* as the process of traveling along a given region of the dataset for the purpose of visualization. This traversal is usually performed on certain regions or features of interest, that may or not be segmented from the rest of the dataset. In order to guide this traversal

along certain regions, we make use of a *skeleton path*, which is an abstract representation that captures the essential shape of the volumetric model, or some of its interesting features. In general, traversal paths form an undirected graph, where each link is usually referred to as a *bone*, and each vertex is called a *joint*. An individual traversal is a walk within that graph from a given source to a destination bone. Alternatively, a traversal may be defined as the subtree generated by a graph search algorithm, such as *depth-first search (DFS)* or *breadth-first search (BFS)*. At each stage or frame of the traversal, a number of bones are selected, referred to as the *traversed bones*.

Once we apply a different transfer function to the voxels in the traversed bones, it is possible to traverse the entire dataset by moving along the path. For this reason, we refer to such transfer functions as *motion-controlled transfer function*.

The attributes for color *RGB* and opacity α of a voxel can be computed as $A_{RGB\alpha} = \Phi(p,t)$ [5], where p is the position of the voxel in euclidean space and t is time. Notice that in this case, since no geometric transformation is performed, the function Φ maps from R^4 (euclidean space and time) to a $RGB\alpha$ tuple. We can extend this to the case where geometrical transformations can be applied to the points in the dataset, as in Chen et al.[5]. We discuss this case later in section 3.5.3

Before traversal-rendering of a dataset, a preprocessing stage is necessary. The preprocessing stage takes a volumetric model and computes a texture decomposition along the bones of its skeleton, or a user specified path structure. After preprocessing, the user can select a path, either automatically or interactively, and rendering is performed using standard 3D texture graphics. Rotation, slicing and animation of the volume can be performed interactively, while traversing the transfer function. These stages are described further below.

3.1 Path structure specification

The first step of the preprocessing stage is to determine an overall path structure from the volumetric object, from which the traversal path is chosen. This structure is a shape abstraction that defines the essential topology of an object. Path structures can be defined interactively by the user or determined automatically, e.g., using a skeleton. This skeleton can be extracted using a number of methods, and any type of skeleton path is suitable for this application. For this paper, we used the methods in Gagvani [8], which is a volume thinning algorithm that obtains a centerline representation of the volume, and in Chuang [6], which uses potential fields. The different parameters controlling the skeletonization process can affect the granularity of the animation. Examples of skeletons are shown in Figures 8(b), 11(a), 12(a), and 13(a). For the case of amorphous data, skeletons are usually more difficult to obtain. However, it is still possible for the user to select a path given the appropriate interaction widgets. Here, we assume that any path, skeleton-based or user-defined, is valid for traversal.

3.2 Texture Decomposition

Traditional texture-based volume rendering uses view-aligned polygons, known as *proxy geometry*, to slice the volume. Once the texture coordinates for the vertices of those slices are computed, the graphics pipeline finds the correct texture via 3D interpolation. This is depicted in Figure 3(a). An improvement to this mechanism is proposed in Li et al.[16, 15], by decomposing the texture into axis-aligned subvolumes, using octrees or binary space partitioning. A speed-up is obtained in sparse volumes because it is possible to skip empty spaces that otherwise must be transferred into the graphics memory. As noted in Li et al.[16], an excessive number of subvolumes may hinder the performance, thus it is necessary to group a number of subtextures into larger 3D textures. In

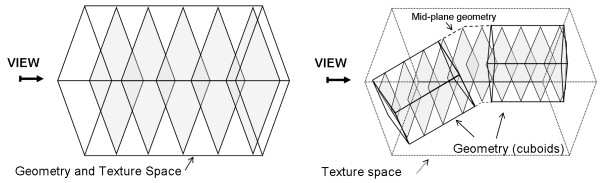


Figure 3: Geometry and texture decomposition. (a) Standard 3D texture (no decomposition). (b) Geometry decomposition into cuboids along the bones of a path. Note that the texture space remains as a single large 3D texture.

our work, decomposition is possible by means of the skeleton path. Unlike the approach in Li et al.[15], we decompose the proxy geometry, rather than the texture space. The sliced geometry consists on a set of cuboids whose axes are oriented along each of the bones (Figure 3(b)). Similar to texture decomposition, this improves the rendering rate by skipping empty space during slice compositing. However, because the decomposition is performed along the shape of the volumetric model, this makes it now possible to create meaningful visualizations of the dataset. For larger datasets that do not fit into texture memory, or sparse volumes, it may be desirable to also partition the texture space.

In this paper, we follow the approach of geometry decomposition and a single 3D texture for the cases when the texture fits into graphics memory. In addition to using cuboids, the joints between bones are modeled using mid-plane geometry [4]. This type of geometry is simple enough to support interactivity, and prevents possible cracking of the dataset when manipulating the bones, as described in Singh et al.[24]. It is important to note that for adjacent geometries, the common plane is composited twice. This results in clear artifacts in the rendering. To resolve this, we use the stencil buffer. Each slicing plane renders its own stencil value so that no voxel is composited twice in the same slicing plane. This also resolves artifacts when cuboids overlap, e.g. during manipulation of the model. Furthermore, using the stencil buffer we can achieve focus+context views, as described in Section 5.

3.3 Selection of the Traversal Path

After texture decomposition, it is possible for the user to determine the path for the transfer function. This can be done either automatically or interactively. Automatic traversal can be performed by applying a recursive search strategy on the graph structure that constitutes the skeleton. Depth first search (DFS) and breadth-first search (BFS) are two common methods. For example, BFS can be used for simulating flow of blood in arteries and veins. Some user interaction may be needed to select the root of the traversal. An-

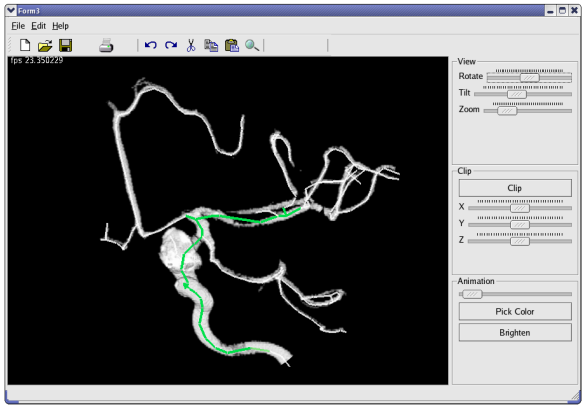


Figure 4: Path selection in the VolEdit program. The green lines highlight the selected path.

other strategy is to let the user select a source and destination bone and compute the path resulting from running a shortest path distance algorithm. Figure 4 depicts a screenshot of the VolEdit [24] application for volume manipulation, as the user selects a traversal path. The bones are highlighted as an aid for the user to select the right bones, and the traversal path is rendered with a different color.

3.4 Standard 3D Texture Rendering

The rendering stage is an extension of standard texture-based volume rendering. Instead of axis-oriented cubes, we sample the bone-oriented cuboids along the view direction and blend the slices in a back-to-front fashion. (Figure 3(b)).

In order to correctly composite the slices of each cuboid, it is necessary to ensure that slices are rendered back to front along the view direction. This is achieved in a two-pass rendering algorithm with the aid of a data structure that indexes polygons with respect to their depth coordinate. First, the slices from each cuboid are stored in the back-to-front data structure. After all cuboids have been processed, the polygons in the data structure are traversed in order and rendered. Since the depth bounds are known and the sampling distance is fixed, this can be done in linear time. This is essential for interactive volume rendering, since re-slicing and re-sorting must be performed when the viewpoint changes or when the user manipulates the bones ([24]).

Transfer functions are usually applied as a lookup color table. This table defines the color and transparency associated with each density value of the volume. This can be performed with the OpenGL command `glColorTable`, but recent graphic cards do not support this extension, and transfer functions are applied through dependent lookups, using texture shaders or fragment programs. Here, we need to apply a different transfer function to different regions of the skeleton path, as described in the following section.

3.5 Traversal Rendering

Once a traversal path has been selected, it is necessary to render the traversed bones with a different transfer function. Let us consider a given bone b, $b \in B$, where B is the set of bones in the path, as shown in Figure 5(a). A connectivity function $C : R^3 \mapsto B$ maps the position of a voxel to a particular bone, i.e., $C(p) = b$.
A dataset traversal can be defined as a function T that maps an instance in time t with a set of bones in the path B_t. That is, a dataset traversal is defined as: $T : R \mapsto B$, such that $T(t) = B_t \subseteq B$

Then, the transfer function for a given voxel can be defined as

$$A_{result}(p,t) = \begin{cases} A_i, & C(p) \in T(t) \\ A_0, & \text{otherwise} \end{cases} \quad (1)$$

where A_i is the transfer function selected for traversal i, $i \in \{1,2,...,M\}$ with M the number of traversals, and A_0 is the original transfer function applied to the dataset.

For hardware rendering, however, we do not solve Equation 1 for each voxel. Rather, we apply it to the individual cuboids as they are processed into view-aligned slices. The back-to-front data structure for the slices is extended to maintain the source bone for each slice. When the slice is being rendered, a proper transfer function is found according to a simpler equation:

$$A_{result}(b,t) = \begin{cases} A_i, & b \in T(t) \\ A_0, & \text{otherwise} \end{cases} \quad (2)$$

This equation assigns the traversal transfer function if bone b corresponds to one of the bones currently being traversed, or the original transfer function otherwise.

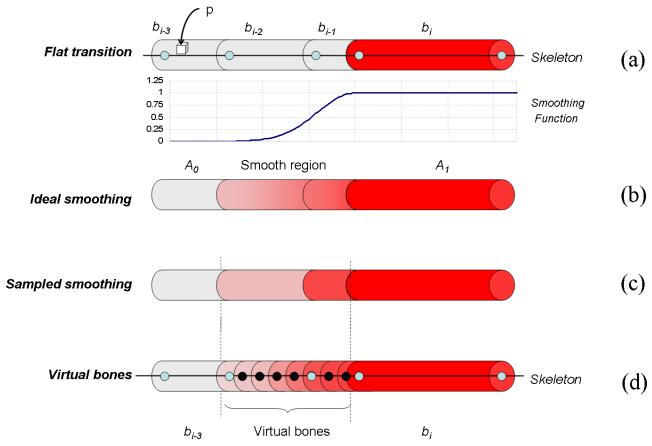


Figure 5: Smooth transitions between transfer functions. (a) Flat transition between bones b_{i-1} and b_i (b) Ideal smoothing using a gaussian function over the falloff bones b_{i-2} and b_{i-1}. (c) Sampled smoothing of the gaussian function (d) Smoothing is improved by adding virtual bones to even out the sampling function

3.5.1 Smooth Transitions

The method described above results in clear discontinuities between the traversal and the dataset transfer functions, as seen in Figure 5(a). Although this may be required for certain applications, such as the simulation of the flow of a substance into the dataset, other applications might require the transition to be smooth. One may also want to smooth along the width of the path segment, so that voxels closer to the center of the bone appear different than the ones in the outside.

In the first case, the transitions along the direction of the traversed bones can be smoothed via interpolation. Let us denote the transfer function for traversal i as A_i and the original transfer function as A_0. Then, for the bones in a boundary of the traversed bones, called the *falloff region*, an intermediate transfer function $A_{i,\alpha}$ is computed as:

$$A_{i,\alpha} = \alpha A_i + (1-\alpha)A_o, \quad (3)$$

which performs linear interpolation. Gaussian smoothing can be obtained by mapping the interpolant α into $1 - g_\sigma(5\sigma\alpha)$, where $g_\sigma(x) = e^{\frac{-x^2}{2\sigma^2}}$ is a zero-mean gaussian filter with standard deviation σ. The 5σ scaling factor is used to approximate the gaussian interpolant to 1 when $\alpha = 1$. Note that when $\alpha = 1$, the transfer function corresponds to A_i, and when $\alpha = 0$, to A_0. By setting different values for the falloff region and σ, it is possible to obtain traversals of different size and different visualization effects. For instance, a sufficiently large traversal with a BFS order can be used to simulate a flood-fill of a dataset.

This solution works on a per-cuboid basis. Whenever a cuboid is processed, the proper transfer function is obtained depending on the position of the bone in the traversal and applied to the corresponding slices. This results in a transfer function applied uniformly to an entire bone, that is, the voxels within each cuboid will be rendered with the same transfer function. For a sufficiently large number of bones, the transition may be smooth, but in a close-up view of the volume, or for large bones, it results in a flat transition, as the sampling of the blending function becomes noticeable (Figure 5(c)). Furthermore, this approach results in poor transitions for bones of varying length.

Some applications may require the smooth transition to be of constant length, regardless of the length of the traversed bones. For this purpose, we propose the use of virtual bones, which split the traversal bones into smaller bones of constant length. We call them virtual bones since they are not part of the original path structure,

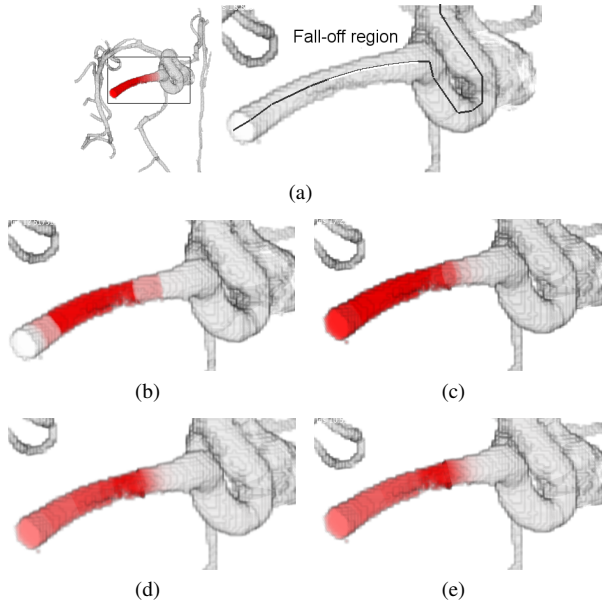


Figure 6: Smooth Transfer function transition for the aneurysm dataset. (a) Falloff region (b) Flat transitions (c)-(e) smoothing with 15, 30 and 60 virtual bones

and are only needed for the region of interest. Splitting the entire path into equally long bones would be unnecessary and would result in a decreased performance. Splitting can be performed just before the slicing stage. The polyline formed by the bones in the falloff region is subdivided into segments of length d, where d is the sampling size of the blending function used for the transition. The virtual bones create small cuboids along the skeleton path. After the new cuboids are found, the new virtual bones can be rendered with the same algorithm described above. Figure 5(d) illustrates the use of virtual bones to improve the traversal rendering. Figure 6 shows the result of applying different techniques for the "aneurysm" dataset, over the fall-off region highlighted in Figure 6(a). Figure 6(b) shows the flat transition resulting of sampling the blending function at every bone. Figure 6(c)-(e) shows the smoothing obtained by adding 15, 30 and 60 virtual bones, respectively.

Two different implementations of the above formulation can be realized. The first one uses a set of lookup tables for each traversal. Once the slices are rendered, the transfer function is applied by enabling the right texture or lookup table. Another implementation can be obtained with the use of dependent textures and fragment shaders. Instead of having different textures for each transfer function, a single texture can be used. A dependent lookup is used to retrieve the appropriate color using the voxel's density, the traversal itself (normalized to the interval $[0,1]$) and the interpolant (α) as texture coordinates. For this implementation, it is not necessary to include virtual bones, since the α value can be interpolated along its length to achieve the desired falloff.

3.5.2 Multiple Traversals

Many applications may require multiple simultaneous traversals. This can be easily implemented by extending the above techniques for looking up into multiple sets of transfer functions. Having additional transfer functions poses a new problem. When two traversals intersect, an appropriate transfer function must be applied to represent the desired effect of transfer function intersection. Different schemes can be applied. For instance, one may simply use one of the transfer functions for the intersection bones, having the effect of one traversal simply overlapping the other. However, for animation it may be necessary to blend the intersecting transfer functions. One example is the animation of blood flow. It is common to represent arterial blood with a red color and venous blood in blue. Blood which contains a mix of both (as it may occur for certain cardiovascular disorders), are usually represented in shades of purple [1]. One mechanism to produce blended transfer functions is to precompute several intermediate transfer functions for different levels of overlapping. However, this is impractical when the number of traversals grow.

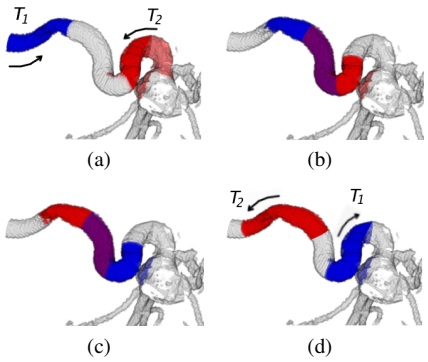


Figure 7: Multiple traversals of the aneurysm dataset. Transfer function T_1 (blue) is traversed from left to right, while T_2 (red) is traversed in the opposite direction (a). The two traversals overlap and are blended (purple region) (b) The two traverals continue their path (c) until they no longer overlap (d).

Here, we propose a solution that requires no additional preprocessing of the transfer functions and runs at interactive rates. When two transfer functions must be rendered for the same cuboid, we interlaced them in the slices. That is, we apply transfer function A_1 to odd-numbered slices and A_2 to even-numbered slices. In our experiments, it turns out to be as fast as applying a single transfer function and only adds a constant overhead the the render time. However, this approach may produce rendering artifacts because each transfer function is applied to half of the slices. For example, artifacts may arise when the transfer functions negate themselves. Some of these problems may be solved by over-sampling the slicing on the highlighted bones. Note that this strategy can be extended to the case of n overlapping transfer functions. A transfer function A_i is applied to the slices whose numbering is equal to i mod n.

Multiple traversals prove to be a useful mechanism to simulate flow and represent different stages of a certain process. For simple transfer functions, the interleaving mechanism proves to be very effective for blending two or more transfer functions. For instance, Figure 7 shows four frames from a traversal of two transfer functions. Notice the blending of the colors in the overlapping region. This blending is a useful tool in animation and illustration to maintain the effect of movement and smooth navigation of a transfer function along a path. Other examples are shown in Figures 1 and 13(b).

3.5.3 Rotation and Slicing

As with traditional texture-based volume rendering, one may rotate or clip the entire volumetric dataset. Rotation of the dataset (or the user's viewpoint) requires the re-computation of view-aligned slices. Global clipping of the entire dataset can be performed easily using the OpenGL command `glClipPlane`. More interesting is the ability to rotate and slice locally on individual cuboids. The rotation of a single bone enables the user to manipulate and animate the dataset and to reconfigure or re-pose its features. This combined with the traversal of transfer functions helps to visualize complex volumetric structures. Note that in the case of local rotation, reslicing is only needed for the modified cuboids. The decomposition

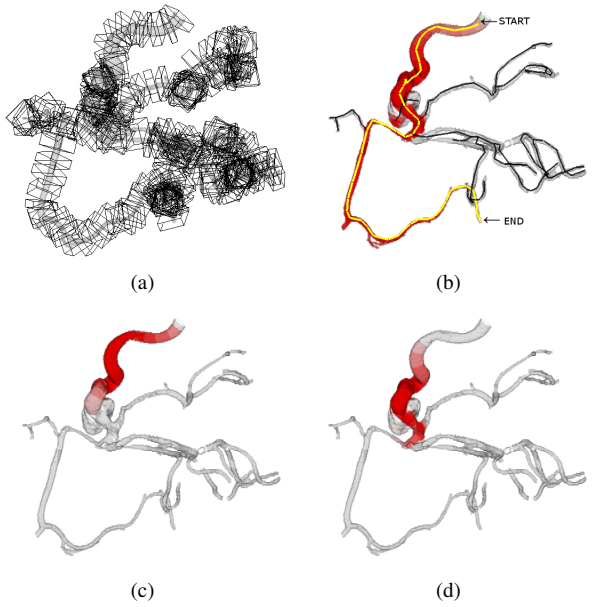


Figure 8: Traversal of the aneurysm dataset. (a) Cuboid decomposition (b) Selection of traversal. The bones in yellow indicate the selected path (c)-(d) Two frames from the aneurysm traversal of a red-colored transfer function

of the model into cuboids also allows clipping or slicing individual parts of the dataset. This can also be accomplished using enabling the OpenGL clip planes on the desired cuboids. The clipping of individual cuboids is useful for rendering hidden parts of a dataset while maintaining the surrounding features intact.

3.5.4 Parameters and user control

A dataset traversal is a multidimensional object defined in terms of the following parameters per path: rendering style R (Direct Volume Rendering DVR, Volume Shading SHD or Isosurface ISO), the transfer function itself T, type of traversal tt (DFS, BFS or PATH), start and end points P_0 and P_1, length L, length of falloff region L_f, and speed s. Other possible parameters (not shown here) include width along the bone and importance. Users control these parameters with interaction widgets. Some of these parameters are incorporated into widgets, and we are currently devising interaction widgets for the others. In this paper we focus on the definition and implementation of dataset traversals.

4 Results

We have experimented with our approach on a number of datasets, including the "aneurysm" (256^3), the "colon" (205x133x261), the "bonsai" (256^3), the "lobster" (256^2x64) and the "visible human", at low resolution (128x512x64). The traversal paths were obtained from a skeleton of 197, 47, 108, 38 and 22 bones, respectively. Our test configuration consists of a Pentium 4 1.7 Ghz PC with 512 MB RAM, equipped with a GeForce 3 with 64MB. The results shown here were obtained with a 700x600 viewport, using 3D textures with a slice distance of 0.5 voxel.

Figure 8 shows an example traversal of the aneurysm dataset. The original dataset is highly convoluted, and it is difficult to understand the 3D structure even when rotating the dataset. When the traversal transfer function is applied, the highlighted region helps to disambiguate the different branches and vascular structures. Figure 8(a) shows the cuboid decomposition resulting from our approach. In this case, the cuboids are of constant width, but this is not the general case. Cuboids are connected with mid-plane geometries [4], not shown in the figure. Figure 8(b) shows the different paths of the aneurysm dataset, with the traversal path highlighted in yellow. Figures 8 (b-d) show several stages of the traversal (R = DVR, T=red TF, tt = PATH, $L = 10$, $L_f = 3$, $P_0 = start$, $P_1 = end$), The traversal is rendered at approximately 20 fps.

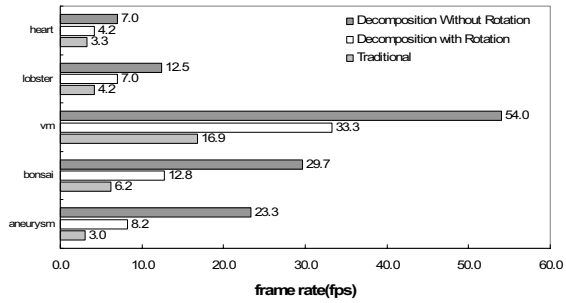


Figure 9: Frames rate for different datasets using various methods: Our decomposition approach without rotation, decomposition with rotation, and without geometry decomposition (Traditional)

Another example is the traversal of the human colon, as depicted in Figure 11(b) (R = DVR, T = yellow-ish TF, tt = PATH, $L = 11$, $L_f = 5$). The traversal follows a single path, as depicted in the first image. A different transfer function, in yellow, helps to identify different parts of the volume. Figure 12(b) shows a traversal sequence of the visible male dataset (R = SHD, T = green bones, tt = DFS, $L = 42$, $L_f = 6$). In this case, a DFS traversal is computed for traversal, as depicted in the first frame of the sequence. A transfer function in green is used to visualize the bones by making certain voxels transparent. Due to smoothing, this sequence has the effect of slow disintegration of the skin and tissues to reveal the bones. Simple phong illumination is obtained using the OpenGL fragment shaders to compute the modulation of a infinite light with the volume gradient. Figure 13(a) shows a traversal of the bonsai dataset using multiple paths and transfer functions. Each of these has different values for parameters T, L, L_f, s, P_0 and P_1, with R=SHD, and tt= PATH. The overlapping transfer functions are blended using the method discussed in section 3.5.2, and results in a holiday light effect, as shown in Figure 13(b). These examples can be shown in the video that accompany this paper, and at `http://www.caip.rutgers.edu/~cdcorrea/traversal`

4.1 Performance

One of the applications of our approach is the improvement on rendering rate due to geometry decomposition along the bones. This results in speed up for sparse datasets, as shown in Figure 9. For traversal of datasets, it may not be necessary to change the viewpoint. In fact, traversal of a transfer function aids to the visualization of convoluted structures without the need to rotate the dataset. This can be exploited by the rendering algorithm to allow fast traversals without the added computation of re-slicing of the volume. Figure 9 shows these results. We compare the rendering rate for a number of datasets for our decomposition approach without rotation (i.e., only traversal), with rotation, and with no decomposition. Rendering of multiple traversals has a constant overhead per traversal. In our experiments each additional traversal incurred in an overhead of 1.5 to 2 ms.

5 Focus+Context Views

Traversal paths do not need to be connected or non-overlapping. In fact, multiple (disconnected) path structures for the same volumetric model, can be used to highlight different features of the dataset while providing context. Consider the case of the visible human

heart, as depicted in Figure 1. This dataset is a portion of the visible human dataset. A skeleton has been found for the heart and the arteries after segmentation and used as the path structure, in order to simulate flow of blood (*focus*). The resulting cuboids can be rendered from this skeleton (only the heart and vessels are rendered). However, one may want to also render the torso, in order to provide *context*. Applying a second skeleton path to the remaining features accomplishes this. A transfer function for this new path can be chosen so as to allow the visualization of the vessels inside. Traversals are used to illustrate the flow of venous blood through the inferior and superior vena cava into the heart and through the pulmonary arteries. Similarly, another set of paths guide the animation of arterial blood flowing through the aorta. Because the cuboids from the two skeletons overlap along the arteries and veins, the corresponding voxels will be composited twice. One way to prevent this is by choosing carefully the transfer functions so as to avoid the overlapping. Another mechanism is to use the OpenGL stencil buffer. After slicing the cuboids, every slice is tested against the stencil buffer such that the test fails whenever another slice along the same Z depth was rendered first. This effectively avoids compositing the overlapping slices. For proper rendering of the traversal, the slices from the context traversal should be rendered before the others (focus). This is true if the sorting algorithm for the back-to-front compositing process is *stable*. Figure 10(a) shows the process of slice interleaving for a set of two skeletons. Figures 10(b) and 10(c) show the results of interleaving with and without the stencil buffer. We can achieve an importance-driven style rendering [27], by leaving the stencil buffer on for all slices between the viewer and the selected bones (Figure 10(d)).

6 Conclusions and Future Work

We presented a methodology for interactive traversal of volumetric datasets via a motion-controlled transfer function. Dataset traversal refers to the exploration of a set of features in a dataset along a path. This is common in medical illustration to represent flow, enhance interesting parts and improve visualization. In this paper, we move the transfer function along a path, by applying the transfer function to the voxels connected to the bones in the path.To make this interactive, we decompose the volume along a path into cuboids. We have shown that this decomposition helps to obtain a rendering speedup compared to traditional texture-based volume rendering. Furthermore, traversal rendering provides a mechanism to visualize convoluted structures without the need to rotate the dataset, which.results in a higher rendering speedup. We have also shown that our approach can be used to obtain focus+context visualizations. This is possible by traversing multiple overlapping skeleton paths at the same time, and combining the results in the rendering pipeline with the stencil buffer. Our results presented here show that traversal rendering can produce improved visualizations for interactive exploration. As part of the future work, we plan to incorporate our approach with the framework for spatial transfer functions in Chen et al.[5]. Other directions for future work are the extension of our decomposition approach to support massive datasets and futher improvements to the rendering process. We plan to extend our method to enable the traversal of importance along a path, in a similar fashion to the method for transfer functions.

7 Acknowledgements

We would like to thank Meaghan Brierley for the images of the Arteriovenous Malformations animation [2]. We gratefully acknowledge the support from NSF(0118760) and the United States Air Force (Brooks).

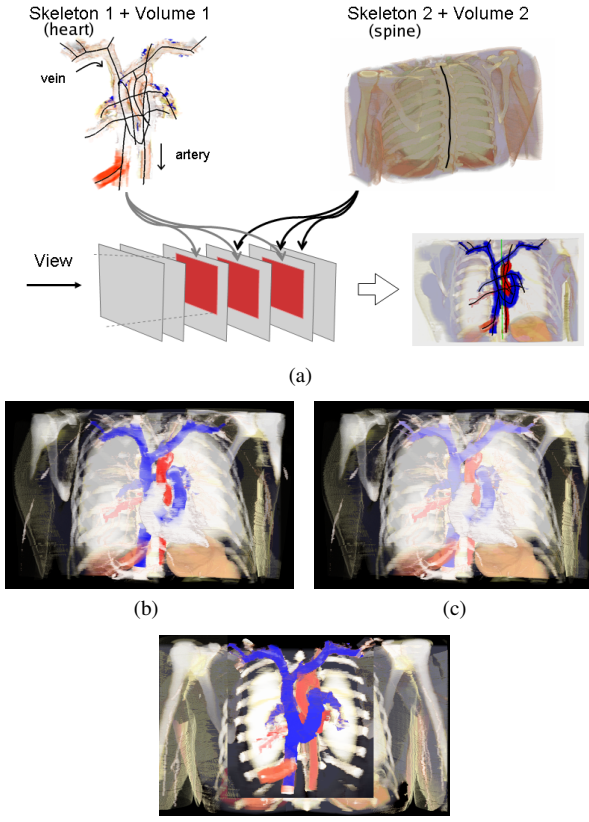


Figure 10: Juxtaposition of skeletons and volumes for focus+context visualization of the visible human heart. (a) Two different skeletons (one for veins and arteries, one for the torso) are used to merge slices from different features. (b) Correct compositing using the stencil buffer in (a) (c) Compositing without the stencil buffer. (d) Cutaway view of the vessels using the stencil buffer along the entire view direction.

References

[1] American Health Association. Pulmonary atresia, 2005. http://www.amhrt.org/presenter.jhtml?identifier=1303.

[2] M. Brierley. Arteriovenous malformations, 2000. http://brainavm.oci.utoronto.ca/swf/intro.html.

[3] S. Bruckner, S. Grimm, A. Kanitsar, and M. Gröller. Illustrative context-preserving volume rendering. In *Proc. EuroVis 2005*, 2005.

[4] J. E. Chadwick, D. R. Haumann, and R. E. Parent. Layered construction for deformable animated characters. In *Proc. SIGGRAPH '89*, pages 243–252. ACM Press, 1989.

[5] M. Chen, D. Silver, A. S. Winter, V. Singh, and N. Cornea. Spatial transfer functions: a unified approach to specifying deformation in volume modeling and animation. In *Proc. Volume Graphics '03*, pages 35–44. ACM Press, 2003.

[6] J. Chuang, C. Tasi, and M. Ko. Skeletonization of three-dimensional object using generalized potential field. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(11):1241–1251.

[7] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proc. HWWS '01*, pages 9–16. ACM Press, 2001.

[8] N. Gagvani and D. Silver. Parameter-controlled volume thinning. *CVGIP: Graph. Models Image Process.*, 61(3):149–164, 1999.

[9] M. Hadwiger, C. Berger, and H. Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proc. IEEE Visualization '03*, pages 301–308. IEEE Computer Society, 2003.

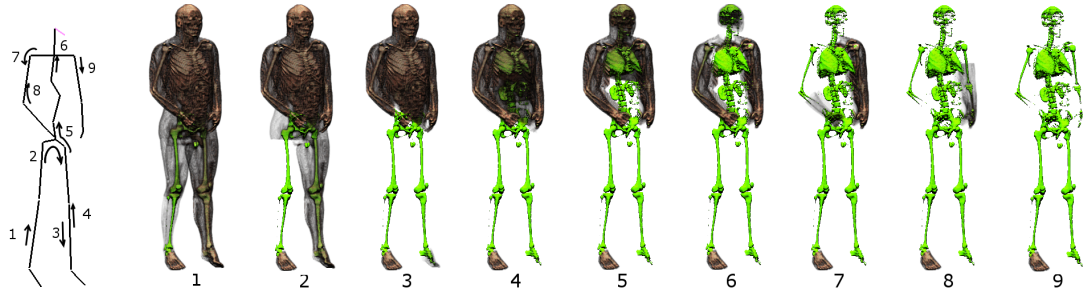
[10] GE Healthcare, 2005. http://www.gehealthcare.com.

[11] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual voyage: interactive navigation in the human colon. In *Proc. SIGGRAPH '97*, pages 27–34. ACM Press/Addison-Wesley, 1997.

[12] J.E. Hummel and I. Biederman. Dynamic binding in a neural network

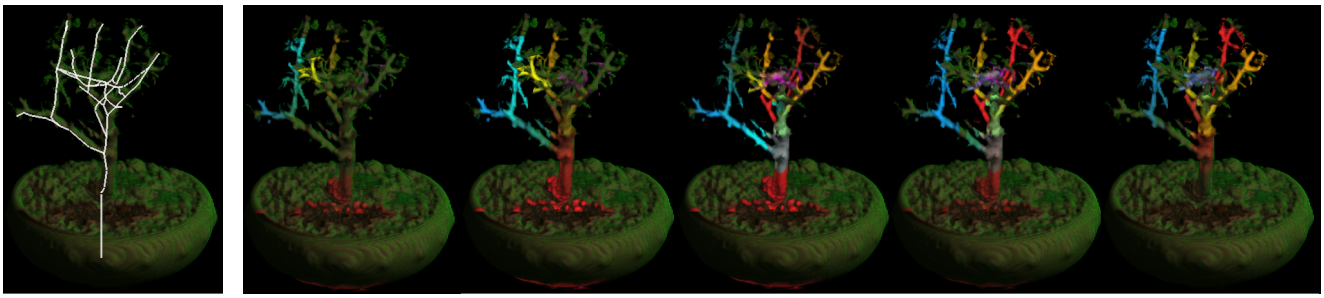
(a) (b)

Figure 11: Traversal of the colon dataset. (a) Skeleton and traversal path (b) A transfer function in yellow along the path.



(a) (b)

Figure 12: Traversal of the visible male dataset. (a) Skeleton and DFS traversal path. (b) Frames from the traversal sequence.



(a) (b)

Figure 13: Multiple traversals on the bonsai dataset (a) Skeleton (b) Blending of multiple traversals produce a *holiday-lights* effect

for shape recognition. *Psychological Review*, 99(3):480 –517, 1992.

[13] T. J. Jankun-Kelly, O. Kreylos, K. Ma, B. Hamann, K. I. Joy, J. Shalf, and E. W. Bethel. Deploying web-based visual exploration tools on the grid. *IEEE Comput. Graph. Appl.*, 23(2):40–50, 2003.

[14] E. LaMar, B. Hamann, and K. I. Joy. A magnification lens for interactive volume visualization. In *Proc. Pacific Graphics PG'01*, page 223. IEEE Computer Society, 2001.

[15] W. Li and A. Kaufman. Texture partitioning and packing for accelerating texture-based volume rendering. In *Proc. Graphics Interface 2003*, pages 81–88, 2003.

[16] W. Li, K. Mueller, and A. Kaufman. Empty space skipping and occlusion clipping for texture-based volume rendering. In *Proc. IEEE Visualization '03*, 2003.

[17] A. Lu, C. Morris, D. Ebert, P. Rheingans, and C. Hansen. Non-photorealistic volume rendering using stippling techniques. In *Proc. IEEE Visualization '02*, pages 211–218. IEEE Computer Society, 2002.

[18] E. B. Lum, A. Stompel, and K. Ma. Using motion to illustrate static 3d shape- kinetic visualization. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):115–126, 2003.

[19] K. Mueller, S. Lakare, and A. Kaufman. Volume exploration made easy using feature maps. In *Scientific Visualization: Extracting Information and Knowledge from Scientific Data Sets*. Springer-Verlag, 2005.

[20] S. Park, C. L. Bajaj, and V. Siddavanahalli. Case study: interactive rendering of adaptive mesh refinement data. In *Proc. IEEE Visualization '02*, pages 521–524. IEEE Computer Society, 2002.

[21] J.R. Pomerantz. In the mind's eye: Julian hochberg on the perception of pictures, film, and the world. *Piecemeal Perception and Hochbergs Window: Grouping of Stimulus Elements over Distances*, 2004.

[22] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume on standard PC graphics hardware using multi-textures and multi-stage rasterization. In *Proc. HWWS '00*, pages 109–118. ACM Press, 2000.

[23] V. Singh and D. Silver. Interactive volume manipulation with selective rendering for improved visualization. In *Proc. IEEE Symposium on Volume Visualization and Graphics '04*, pages 95–102. IEEE Computer Society, 2004.

[24] V. Singh, D. Silver, and N. Cornea. Real-time volume manipulation. In *Proc. Volume Graphics '03*, pages 45–51. ACM Press, 2003.

[25] Siemens Medical Solutions, 2005. http://www.medical.siemens.com.

[26] Viatronix, 2005. http://www.viatronix.net.

[27] I. Viola, A. Kanitsar, and M. Eduard Gröller. Importance-driven volume rendering. In *Proc. IEEE Visualization '04*, pages 139–145. IEEE Computer Society, 2004.

[28] F. Vos, I. Serlie, R. van Gelder, F. Post, R. Truyen, F. Gerritsen, J. Stoker, and A. Vossepoel. A new visualization method for virtual colonoscopy. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2001*, pages 645–654. Springer Verlag, 2001.

[29] M. Wan, F. Dachille, and A. Kaufman. Distance-field based skeletons for virtual navigation. In *Proc. IEEE Visualization '01*, pages 239–246. IEEE Computer Society, 2001.