Feature Selection of 3D Volume Data through Multi-Dimensional Transfer Functions

Sangmin Park [a] Chandrajit Bajaj [a]

[a] *University of Texas at Austin*

Abstract

Direct volume rendering maps data values to visual properties such as transparency and color through transfer functions. Traditional multi-dimensional functions are generated based on a 2D histogram of function value and gradient magnitude. When two different features overlap in the 2D histogram, the traditional transfer functions cannot visually distinguish the features, since overlapped areas have similar visual properties. In this paper, we describe a new multi-dimensional transfer function that enables visual differentiation of features even in the case when two different features overlap in the 2D histogram. Furthermore, we provide details of an implementation of our transfer function on modern programmable graphics hardware.

Key words: Volume Rendering, Transfer Functions
PACS:

1 Introduction

Direct volume rendering enables visualization of volume data without first resorting to the geometry of triangles. It simply maps each volume element (or voxel) to visual properties such as opacity and color and then composites the properties into an image. The conversion is done via transfer functions. Good transfer functions produce good images that accentuate interesting structures in volume data while removing uninteresting regions. However, it is difficult to construct good transfer functions. Kindlmann's survey [7] reviews many versions of transfer functions and divides them into the two categories of data-driven and image-driven functions. In this paper, we extend the results of [13] (a data-driven multi-dimensional transfer function) with more detailed exposition on feature selection and effectively demonstrate that our multi-dimensional transfer functions are significantly better to visually distinguish overlapped features than traditional transfer functions.

Related Work

Two-dimensional (2D) transfer functions of function value (v) and gradient magnitude (g) were suggested by Levoy [11]. He also proposed semi-transparent multiple surface visualization. The Kindlmann's survey [7] explains many kinds of transfer functions, but only a few multi-dimensional transfer functions have been developed. As transfer function domains are extended to two and three dimensions, the function yields more power in feature selection [9] [10]. For the same reason, it is difficult to set values manually. Nevertheless Kniss *et al.* [10] suggested a convenient user interface widgets for setting three-dimensional (3D) transfer functions. Users, however, still needed to decide voxel transparencies from the displayed information of the two-dimensional (2D) histogram of v and g, while selecting interesting regions.

Kindlmann and Durkin [8] suggest the semi-automatic transfer function generation method for boundary visualization through 3D histogram volume inspection. The method uses second derivatives to automatically compute boundary thickness (σ). Based on σ, voxel transparency is computed in the 2D subspace defined by v and g. Their method has the following shortcomings: if two different voxels have the same values of v and g, then they will be set to the same transparency, even though the voxels belong to different features. In other words, when features overlap in the 2D histogram of v and g, all overlapped portion are assigned the same transparency. Distance maps have been used for volume rendering in [5] and [6] for the visualization of binary segmented volume data. Ziou and Tabbone's survey [15] shows several ways to remove phantom edges.

The remainder of the paper is organized as follows. In section 2, we present a boundary detection method for 3D volumetric data. In section 3, we explain how to compute the distance from a voxel to a boundary using the boundary detection method. Based on the distance, we suggest new opacity functions in section 4. In section 5, we provide details of the implementation of our transfer function in modern graphics hardware along with several applications on volumetric datasets Finally, we suggest some extension of our work in section 6.

2 Boundary Detection in 3D Volume Data

Several boundary detection (or edge detection) algorithms have been published in the image processing and pattern recognition literature. One of the traditional edge detection techniques is Canny's method that finds the local maxima along the gradient direction [2]. Most common edge detection schemes consist of three such operations: differentiation, smoothing and edge labeling [15].

First, differentiation is necessary to identify edges. We compute gradient vectors represented by $\bigtriangledown f$ using central differences. $\| \bigtriangledown f \|$ stands for its magnitude and the normalized vector is computed and denoted by $\vec{n} = \frac{\bigtriangledown f}{\|\bigtriangledown f\|}$. The frequently used second-order derivative operators are the Laplacian and the directional second-order derivative (DSD). In this paper, we compute and use the DSD along the gradient direction. The DSD operator is defined by $\frac{\partial^2 f}{\partial^2 \vec{n}} = \vec{n} \cdot \bigtriangledown \| \bigtriangledown f \|$.

Second, for smoothing purposes of data with substantial edge preservation, a bilateral filter [14] is applied to $\| \bigtriangledown f \|$ and $\frac{\partial^2 f}{\partial^2 \vec{n}}$. The domain component is $W_d(\vec{x})$ and the range component is $W_r(\vec{x})$ for each voxel, $\vec{x} = (\, i \;\; j \;\; k \,)^T$:

$$W_d(\vec{x}) = \exp\left[-\frac{(\vec{x}-\vec{y})^T(\vec{x}-\vec{y})}{\sigma_d^2}\right], \; \vec{y} \in N_{\vec{x}} \tag{1}$$

and

$$W_r(\vec{x}) = \exp\left[-\frac{(f(\vec{x})-f(\vec{y}))^2}{\sigma_s^2}\right], \; \vec{y} \in N_{\vec{x}}, \tag{2}$$

where $N_{\vec{x}}$ is a neighbor set of $\vec{x}$. $W_d(\vec{x})$ computes the weights of each voxel based on the spatial distance at $\vec{x}$, $W_r(\vec{x})$ measures the "photometric" similarity between the function values of $\vec{x}$ and the neighborhood. A bilateral filter, $\widetilde{f}(\vec{x})$, is composed of the two components and is described as follows:

$$\widetilde{f}(\vec{x}) = \frac{\sum_{\vec{y} \in N_{\vec{x}}} W_d(\vec{x}) W_r(\vec{x}) f(\vec{x})}{\sum_{\vec{y} \in N_{\vec{x}}} W_d(\vec{x}) W_r(\vec{x})} \tag{3}$$

Finally, edge labeling is used to identify authentic edges (surface boundaries in 3D) while suppressing false edges, caused primarily by noise and the non-maximum high value of $\| \bigtriangledown f \|$. Some error accumulation in the derivative computations also results in non-trivial false edges. However, in this paper, we assume that our input volume data have reasonably high signal-to-noise ratio and a pre-application of the bilateral filter solves the error accumulation in differentiation. We, therefore, only consider removal of the "phantom edges" (as defined in [3]) due to the multiple local maxima of $\| \bigtriangledown f \|$.

3 Distance to a Boundary

Direct volume rendering does not require extraction of geometric structures such as triangles to visualize interesting boundaries within volumetric data. Instead, it is enough to assign appropriate transparencies for all voxels. In other words, if a voxel is exactly on a boundary and we wish to visualize the boundary then the voxel is assigned to be totally opaque. Furthermore, for voxels away from a boundary, the opaqueness should decrease. Transparency decisions are closely related to the distance from a voxel to a boundary. Knowing the voxel distance from a boundary, one can easily set up voxel transparency. In this section, we present how to compute the distance from a voxel to an authentic boundary.

The distance is computed by tracing two rays in both the positive ($+\bigtriangledown f$) and negative ($-\bigtriangledown f$) directions from each voxel location. One of the directions is selected in which $\| \bigtriangledown f \|$ increases. The two values of $\| \bigtriangledown f \|$ and $\frac{\partial^2 f}{\partial^2 \vec{n}}$ are trilinearly interpolated at several sampling locations along the two rays. The goal is to locally and efficiently determine the zero-crossing locations of $\frac{\partial^2 f}{\partial^2 \vec{n}}$. Since the vector of $\bigtriangledown f$ is perpendicular to the boundary [15], by following one of the positive or negative directions, one is guaranteed to find an authentic boundary in 3D space. Fig. 1 shows a 2D example on computing the distance from a voxel to an authentic boundary edge along the negative gradient direction. Ideal graphs of $f(x)$, $f'(x)$, and $f''(x)$ are also shown.

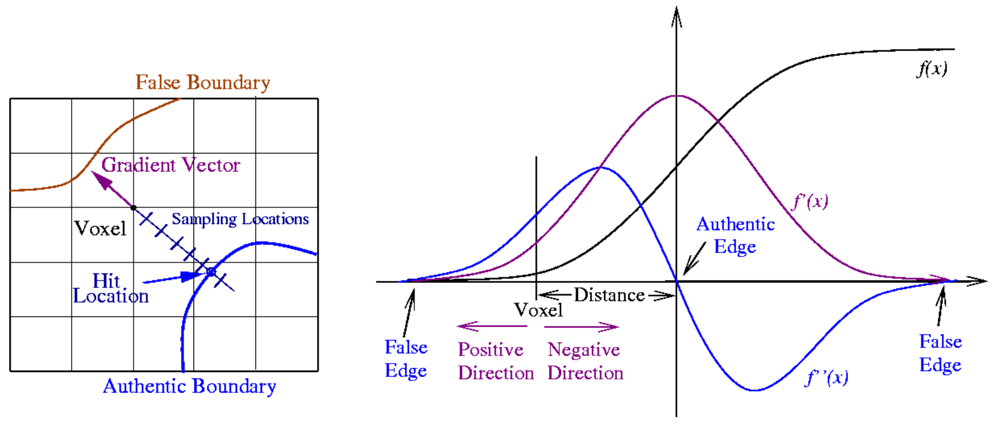


Fig. 1. 2D Example on the Distance Computation (Left Fig.) and Ideal Graphs of $f(x)$, $f'(x)$, and $f''(x)$ along the Directions of $\pm \bigtriangledown f$ (Right Fig.): The distance from a voxel to a boundary can be computed by shooting a ray along one of the directions of $\pm \bigtriangledown f$.

Based on Kindlmann's ideal boundary model [8] of Eq.(4), the three values of f, $\| \bigtriangledown f \|$ and $\frac{\partial^2 f}{\partial^2 \vec{n}}$ are changed as shown in Fig. 1(right) along the directions of $\pm \bigtriangledown f$. Sampling locations are computed by $\vec{x}_s(t) = \vec{x} + t\frac{\bigtriangledown f(\vec{x})}{\|\bigtriangledown f(\vec{x})\|}$, where $-d_{max} < t < d_{max}$. When $f''(\vec{x}_s(t_1)) \times f''(\vec{x}_s(t_2)) < 0$ is true, an authentic boundary is between the two sampling points. Once we find t_1 and t_2, we use the bisection method [1] to obtain an exact location. Experimentally, we define the sampling interval and d_{max} such as $L_{min}/5$ and $L_{min} \times 15$ respectively, where $L_{min} = Min$(width of a voxel, height of a voxel, depth of a voxel).

$$v = f(x) = v_{min} + (v_{max} - v_{min})\frac{1 + \mathrm{erf}(\frac{x}{\sigma\sqrt{2}})}{2} \qquad (4)$$

To verify the nearest boundary finding algorithm, we generated the Sphere dataset with the ideal boundary model Eq. 4 and the sphere equation ($x^2 + y^2 + z^2 = r^2$). Since Eq. 4 is a function of distance, the function value at a voxel location $P(p_x, p_y, p_z)$ is:

$$v = f(x) = v_{min} + (v_{max} - v_{min}) \frac{1 + \mathrm{erf}(\frac{\sqrt{p_x^2+p_y^2+p_z^2}-r}{\sigma\sqrt{2}})}{2} \tag{5}$$

For the Sphere dataset, the nearest boundary of a voxel is on the line that connects the voxel and the origin. We can write the equation of a line using the gradient vector ($\bigtriangledown f$) and voxel location (P) as

$$L = \bigtriangledown f \times t + P = (Tp_x, Tp_y, Tp_z) \times t + (p_x, p_y, p_z), \tag{6}$$

where

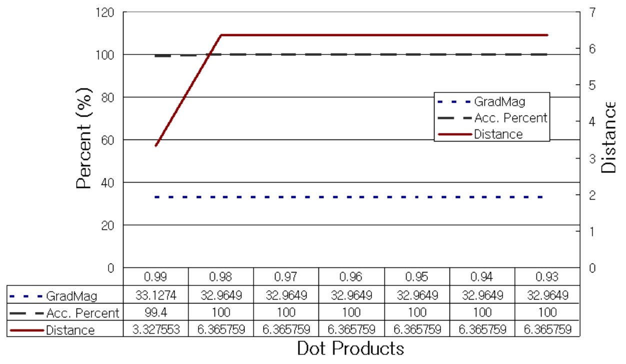
$$T = \frac{e^{-\frac{\left(\sqrt{p_x^2+p_y^2+p_z^2}-r\right)^2}{2\sigma^2}}(v_{max} - v_{min})}{\sigma\sqrt{2\pi}\sqrt{p_x^2+p_y^2+p_z^2}} \tag{7}$$

To find the nearest boundary point, there should exist t for $L = \bigtriangledown f \times t + P = (0, 0, 0)$. Therefore, we get

$$t = \frac{e^{\frac{\left(\sqrt{p_x^2+p_y^2+p_z^2}-r\right)^2}{2\sigma^2}}\sigma\sqrt{2\pi}\sqrt{p_x^2+p_y^2+p_z^2}}{v_{min} - v_{max}} \tag{8}$$

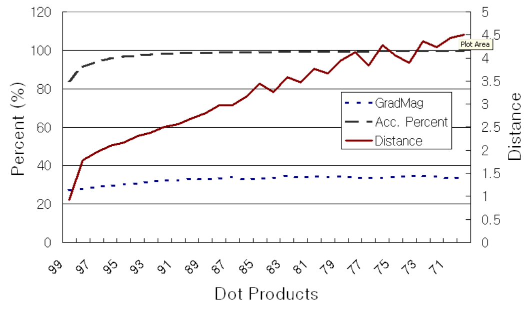
Since there exists only one t for each line equation at P, we can say that the line passes through the origin. Therefore, the nearest boundary finding algorithm converges to a nearby boundary point for the Sphere volume dataset. However, there is no simple way to prove the algorithm for general datasets such as Tooth and Torso. We suggest another experimental measurement of dot products between the two gradient vectors of a voxel and the nearest boundary point found by the algorithm. The dot product should be close to 1, if the boundary point is a nearest one, which is clear in the Sphere dataset.

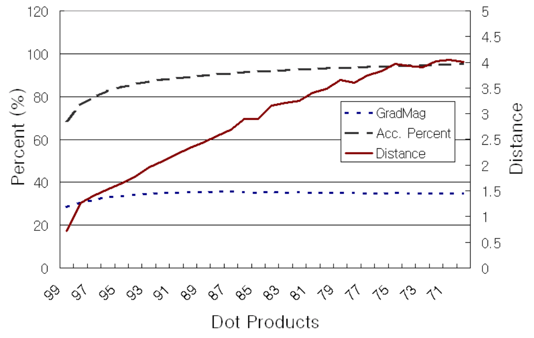
In Fig. 2, we computed the average distance and the average gradient magnitude values at each dot product (multiplied by 100 in the graphs). We also computed the ratio (%) of the number of voxels at each dot product to the total number of voxels and it is accumulated along the horizontal axis. To remove outliers, two threshold values are used. One is for the gradient magnitude at each voxel location (=5.0) and the other is for the gradient magnitude at the boundary (=20.0). The first one is to remove false boundary in homogeneous regions and the second one is for noise boundary. The average dot products of Tooth and Torso are 0.9909 and 0.9553 respectively. The both datasets are



	0.99	0.98	0.97	0.96	0.95	0.94	0.93
GradMag	33.1274	32.9649	32.9649	32.9649	32.9649	32.9649	32.9649
Acc. Percent	99.4	100	100	100	100	100	100
Distance	3.327553	6.365759	6.365759	6.365759	6.365759	6.365759	6.365759

(a) Sphere (Synthesized data by the ideal boundary model, Eq. 5)
Average dot product = 0.9994





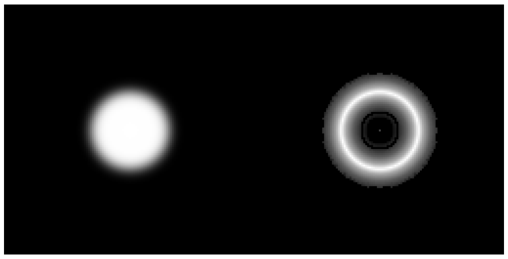
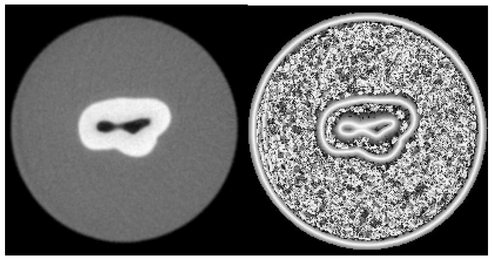
(a) Tooth
Average dot product = 0.9909

(b) Torso
Average Dot product = 0.9553

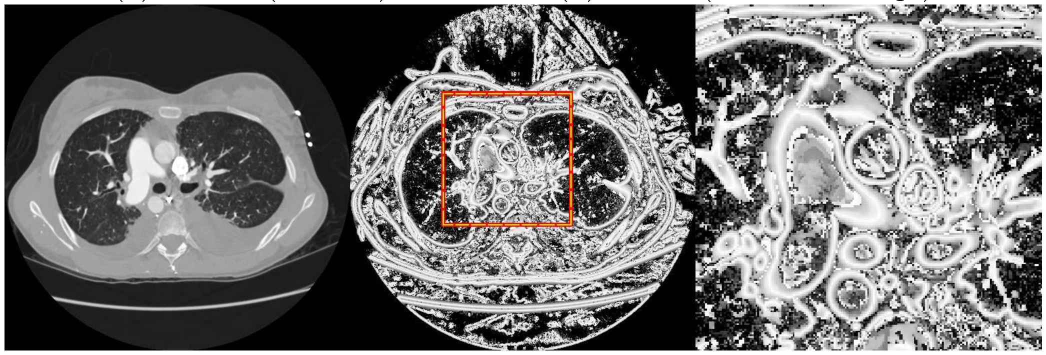
Fig. 2. Dot Products and Accumulated Percents: The sphere dataset is generated from the ideal boundary model with $\sigma = 3.0$ and $radius = 15$. The horizontal axis of each graph represents the dot product values ($\times$ 100) between the two gradient vectors of a voxel and the nearest boundary computed by the algorithm. The vertical axis represents accumulated percent (left) and distance (right). The left vertical axis is also used for the gradient magnitude. The dashed lines represent the accumulated percent along the horizontal axis. The solid and dotted lines show the average distance and gradient magnitude at each dot product value respectively.

close to the synthesized sphere dataset (0.9994). Therefore, we can say that the nearest boundary finding algorithm converges to a nearby boundary. From Fig. 2, it is also known that the error increases with the distance.

Fig. 3 shows the results of the distance computation as well as a slice through these different datasets: all distance values are flipped to provide easy boundary perception in the images. The ideal boundary model of Eq. 4 is used to generate the volume data of Fig. 3 (a). The bright grey colors of Fig. 3 (a) (right) and (b) (right) represent the boundary of the volume dataset. Unlike the synthesized data, Fig. 3 (b) (right) shows noise-like bright colors in the homogeneous regions of the tooth dataset. The main reason of the noisy boundary is that there are many zero-crossing locations of $\frac{\partial^2 f}{\partial^2 \vec{n}}$ with the small value of $\| \bigtriangledown f \|$ in the homogeneous regions. Fig. 3 (c) shows more complex boundaries of the CT-angio Torso dataset. Similar patterns are repeated as in the tooth

(a) Sphere ($\sigma = 3.0$) (b) Tooth (85th CT Image)



(c) Torso (70th CT Image)

Fig. 3. Slice Images of Volume Datasets and Distance Computation Results: The left-hand side of (a) is a slice image of the volume data generated with an ideal spherical boundary model of Eq. 4 and the distance computation results are on the right-hand side. (b) (left) is a slice from a Computed Tomography (CT) dataset of a human tooth and (b) (right) shows the distance map for this image. Similarly, the images of (c) are generated from a CT-angio Torso volume dataset, the middle image box of (c) is enlarged in the right-hand side. Each dataset resolution is following: Sphere(128^3), Tooth($256 \times 256 \times 161$), and Torso($512 \times 512 \times 181$).

dataset. The noisy boundary is easily removed from the multi-dimensional transfer functions by deselecting the low gradient magnitude regions.

4 Multi-Dimensional Transfer Functions

For our multi-dimensional transfer functions, we combine a distance based function with 2D opacity functions that is based on intensity (v) and gradient magnitude (g). The combined transfer function thus has both user control and an automatic generation of each voxel transparency.

4.1 Opacity Functions of Distance

Once we decide the distance from each voxel to a boundary, each voxel transparency can be easily computed as a suitable function of distance. We suggest three different opacity functions (linear, concave, and convex) as shown in Fig. 4. The linear opacity function maps a linearly flipped distance, while the concave and convex functions use the n-th power of the distance to control the boundary thickness. For larger value of n, the convex function generates thicker boundaries. Similarly, thinner boundaries can be generated using the concave functions with larger n's. The three distance functions are represented as following:

$$\alpha_d(d) = Max(-\frac{a \times d}{d_c} + a, \quad 0), \tag{9}$$

$$\alpha_d(d) = \begin{cases} \frac{a}{d_c^n}(\mid d - d_c \mid)^n \text{ if } d < d_c \\ 0 \qquad\qquad\qquad \text{others} \end{cases}, \tag{10}$$

and

$$\alpha_d(d) = Max(-\frac{a \times d^n}{d_c^n} + a, \quad 0), \tag{11}$$

where $0 < d_c \leq d_{max}$, $0 \leq a \leq 1$, and $n > 1$. Eq. 9, 10 and 11 are the linear (Fig. 4 (a)), concave (Fig. 4 (b)), and convex (Fig. 4 (c)) opacity functions respectively. The spatial effect of the three opacity functions are controlled by the two variables of d_c and a, while the shape of the nonlinear functions is dominated by n.

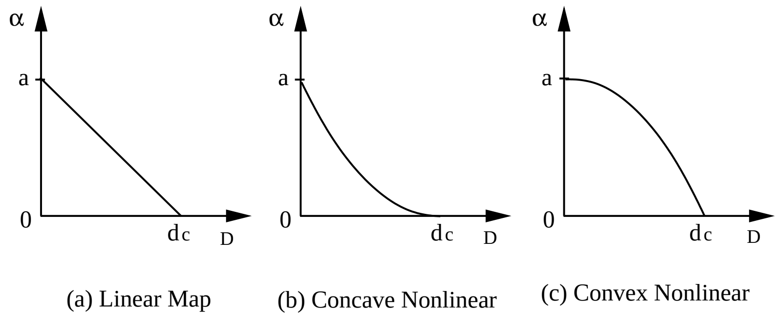
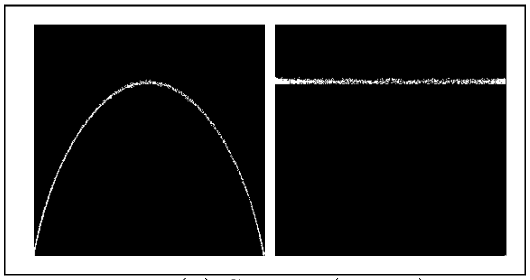


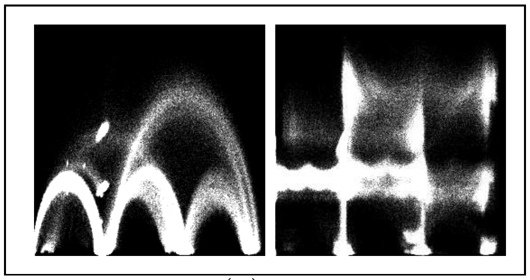
Fig. 4. Alpha Maps

The opacity function, $\alpha_d(d)$, automatically computes each voxel transparency for the entire scalar volume dataset. However, when one wishes to visualize only a small range of intensity (v) values, one cannot accomplish it with the opacity function alone, since it does not have v as a parameter. To achieve sub-range selection flexibility, we define another function of v, $\alpha_u(v)$, that is user specified. The final opacity values of each voxel are then computed by $\alpha_u(v) \times \alpha_d(d)$. Our transfer function thus has both automatic opacity generation and user control. In other words, once a user specifies some range of v with $\alpha_u(v)$, each voxel opacity is automatically generated by $\alpha_d(d)$ using one of the alpha maps of Fig. 4.

4.2 Gradient Magnitude for the Second Axis

Most multi-dimensional transfer functions have gradient magnitude (g), as the secondary axis, while intensity (v) works as the primary axis. The 2D transfer functions of v and g are more powerful than the 1D functions of v [9] [10]. However, as the transfer function domain is extended to 2D space, it makes it more difficult to search and locate features. To reduce the search time in the 2D space of v and g, we replace the values of $\parallel \bigtriangledown f \parallel$ with the interpolated values at the boundary hit location (Fig. 1). This makes all voxels have the local maximum values of $\parallel \bigtriangledown f \parallel$ along the direction of $\bigtriangledown f$ instead of their previous values.


(a) Sphere ($\sigma = 3$)


(b) Tooth

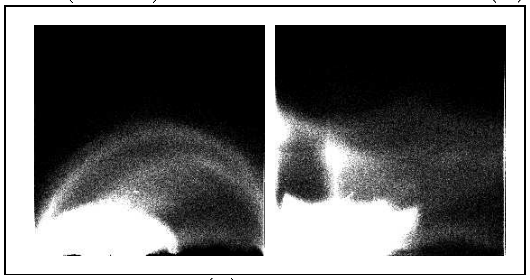

(c) Torso

Fig. 5. Normal (Left) and Stretched (Right) Histograms: The horizontal axis represents function values (v) and the vertical axis represents gradient magnitude, $\parallel \bigtriangledown f \parallel$, ($g$).

Fig. 5 shows the resulting histogram (the right-hand side of each box) of the voxel value replacement as well as the original 2D histogram (the left-hand side of each box). While the voxel value replacement generates clearer and simpler histograms for the spherical and Tooth datasets, the histogram is still murky for the Torso dataset. This is not because the Torso dataset does not have clear boundaries, but because several complex boundaries are condensed in a very small region.

There are two benefit of the above voxel replacement. First, curved features in the v and g space are mapped to straight line features (Fig. 5 (a) (right)) as well as parallel with the horizontal axis, thereby making it easier for user subrange specification. Second, when two different curves are partially overlapped in the original histogram, they become separated at each maximum value of

$\| \bigtriangledown f \|$. More details on this benefit are shown in the next section, via our example application on these volumetric datasets.

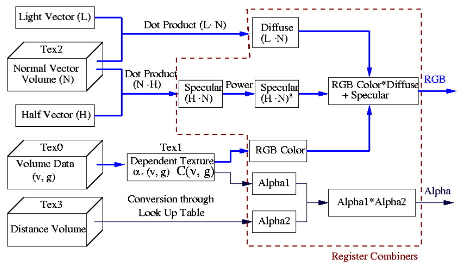
The 1D opacity function, $\alpha_d(d)$, is easily extended to a 2D opacity function of v and g, represented as $\alpha_u(v,g)$, where g represents the local maximum of $\| \bigtriangledown f \|$ along $\bigtriangledown f$. Therefore, each voxels opacity is finally decided by $\alpha_u(v,g) \times \alpha_d(d)$. The 2D opacity function, $\alpha_u(v,g)$, is controlled by a user like Kniss's function [9], but the opacity values can be 1 (meaning completely opaque) unlike Kniss's function over selected regions. Once an interesting area is selected in the 2D space of v and g with $\alpha_u(v,g)$, the opacity function of distance, $\alpha_d(d)$, automatically generates alpha values. The domain of d replaces Kniss's directional second derivative axis.

5 Hardware Implementation and Example Applications

We have implemented a 3D texture-based volume renderer in PCs equipped with nVidia graphics cards such as GeForce3, 4, and FX [12]. Multi-dimensional transfer functions can also be implemented on a PC equipped with such graphics cards that provide at least four 3D multi-textures and dependent texture reads with register combiners. Fig. 6 (a) shows the rendering pipeline for such cards and (b) is the equivalent Cg Program [4]. In this pipeline, dependent textures are used for implementing the opacity function, $\alpha_u(v,g)$ that is controlled by a user and the texture look-up table is used for a distance-based function, $\alpha_d(d)$.

The volume rendering pipeline requires six components, RGB normal (three components), intensity (v), gradient (g), and distance (d), for each voxel. If we wish to visualize a 256^3 volume dataset, (=16 Mbytes), then we need at least $256^3 \times 6$, (=96 Mbytes), texture memory. The RGB normal texture can be compressed using the s3tc format [12], one of the ARB OpenGL extensions, to reduce the texture memory requirement with an image quality trade-off.

Fig. 7 shows the result images rendered by the transfer functions of $\alpha_u(v,g) \times \alpha_d(d)$. Each area of **A** through **E** are rendered individually and any combination of the areas can be rendered at the same time like the image of **C** and **D**. The **A**, **B**, **D**, and **E** areas are rendered with the convex alpha map of the parameters, $a = 1$, $d_c = d_{max}$, and $n = 3$ (which generate thick boundaries) and the **C** area is with the concave map of $a = 0.4$, $d_c = d_{max}$, and $n = 9$ (which make thin boundaries). We experimentally defined the parameters to obtain visually smooth continuity between the two areas of **C** and **D**. Fig. 8 shows the Torso dataset images. To render the two areas of A and B at the same time, we make A area semi-transparent with $\alpha_u(v,g) = 0.15$ and the linear map of $a = 1$, $d_c = d_{max}$, and $n = 3$ are used for both areas.



(a) Register Combiners

```
float3 Expand(float3 Vector) { return (Vector - 0.5)*2.0; }
void main ( float3 TexCoor          : TEXCOORD0,   // Texture Coordinates
        out float4 oColor           : COLOR,       // Output RGBA
        uniform sampler3D RGB_NormalMap,           // Voxel Normal, Tex2
        uniform sampler3D DistanceMap,             // Distance, Tex3
        uniform sampler3D vg_Map,                  // (v, g), Tex0
        uniform sampler2D TwoDimTF,                // User Defined, Tex1
        uniform float3 Lightv,                     // Light Vector
        uniform float3 Halfv) {                    // Half Vector
        float3 Normal = Expand(tex3D(RGB_NormalMap, TexCoor).rgb);
        float   Diffuse  = abs (dot (Normal, Lightv));   // Diffuse Color
        float   Specular = abs (dot (Normal, Halfv));    // Specular Color
        float   Power8 = pow(Specular, 8);               // 8th Power of Specular
        float3 Specularf3 = float3(Power8 , Power8 , Power8 );
        float3 vg_Tex = tex3D(vg_Map, TexCoor).rgb;// (v, g) of Each Voxel
        float4 TwoDim_RGBA = tex2D(TwoDimTF, float2(vg_Tex.x, vg_Tex.y));
        float4 DistanceTex = tex3D(DistanceMap, TexCoor); // Distance-Based α
        oColor.w = DistanceTex.w*TwoDim_RGBA.w;
        oColor.xyz = (Diffuse*TwoDim_RGBA.xyz + Specularf3)*oColor.w; }
```

(b) Equivalent Cg Program

Fig. 6. Rendering Pipeline for nVidia GeForce Cards: (a) shows the rendering pipeline with register combiners and (b) is the equivalent Cg program of (a). In the pipeline, three texture volume datasets, a light vector, and a half vector are fed into the register combiners. After the user defined 2D dependent texture (Tex1) converts the (v,g) data into RGB colors and alpha values, the combiners computes the output of final RGB colors and alpha values with diffuse and specular colors. The solid lines represent data flow and processes, while the dashed line indicates register combiners.

Since, when we make volume datasets, the gradient magnitude values of each voxel are replaced with the local maxima at the hit locations (explained in 4.2), it is easy to select overlapped features. For example, the histogram of **B** is partially overlapped with the graphs of **A**, **C** and **D** in the normal case of Fig. 7 (top right). Therefore, it is not easy to select only the feature of **B**, when one uses normal graphs. However, in the stretched histogram, the feature **B** can be selected, while removing all other features, since the four features are separated in v and g space.

6 Conclusions and Extensions

In this paper, we have presented a new multi-dimensional transfer function that has both user control and automatic alpha value generation. When two or more features are overlapped in the normal histogram, it is usually difficult to visually distinguish them separately. However, this task is significantly alleviate with our stretched histogram, as at least one or two overlapped features can be selected and visualized, while removing others. Unlike traditional transfer functions, our new transfer function also provides automatic alpha value generation on a per voxel basis.

For extensions, we are currently considering semi-automatic ways of generating $\alpha_d(d)$ dependent with $\alpha_u(v, g)$. Another improvement map is to reduce the preprocessing time of computing the distance. Finally, when a dataset contains many complex features in a small area of the v-g histogram like the

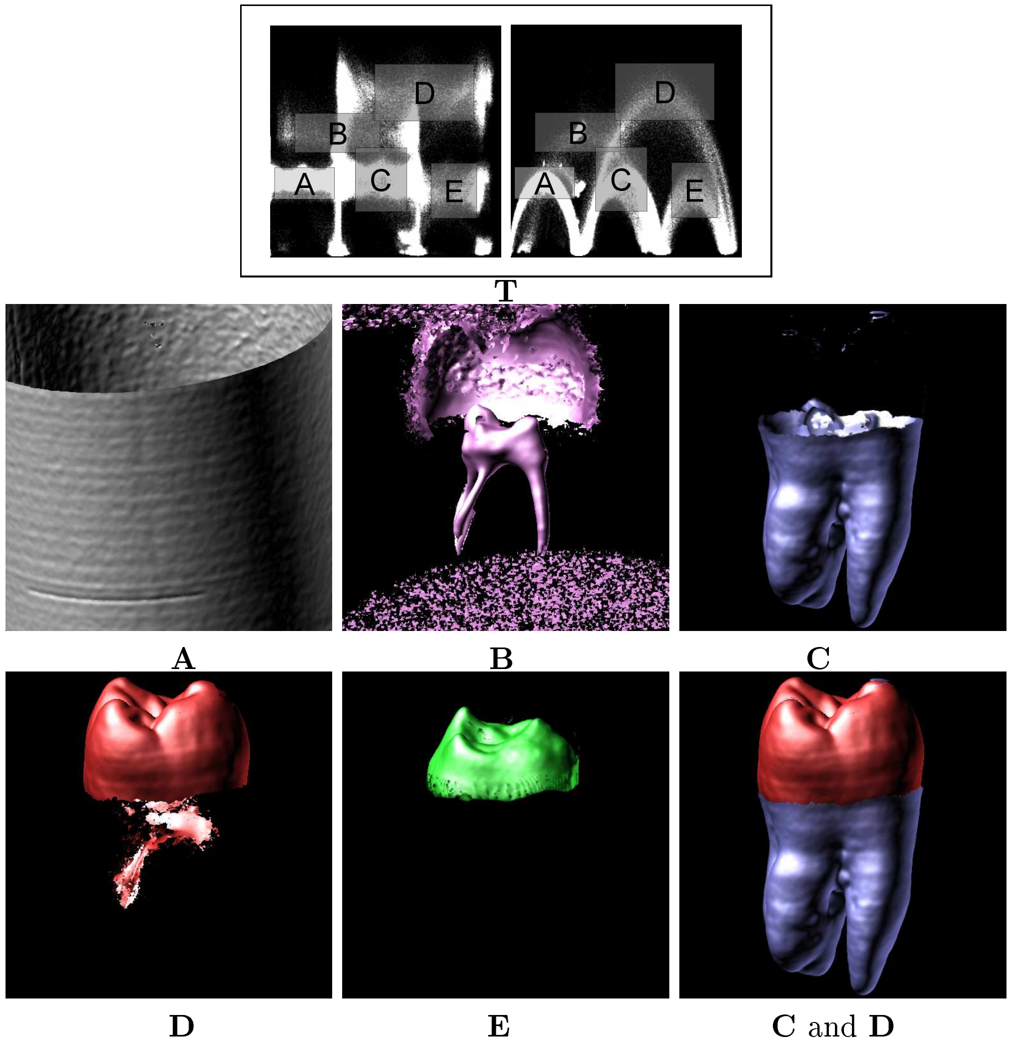


Fig. 7. Feature Selection with the Stretched Histogram and Rendering Results of the Tooth Dataset: **T** (left) shows feature selection on the stretched histogram and each box is located in the same place on the normal histogram (right). Each region is rendered in the images of **A** through **E**.

Torso dataset, it is hard to recognize the features through even the stretched histogram. We are considering alternate transfer functions to better separate the features in the transformed histogram.

7 Acknowledgments

This research was supported in part by NSF grants INT-9987409, ACI-022003, EIA-0325550, NIH grant OP20 RR020647, a grant from the Whitaker foundation, and a subcontract UCSD 1018140 as part of the NSF-NPACI, Interaction Environments Thrust.

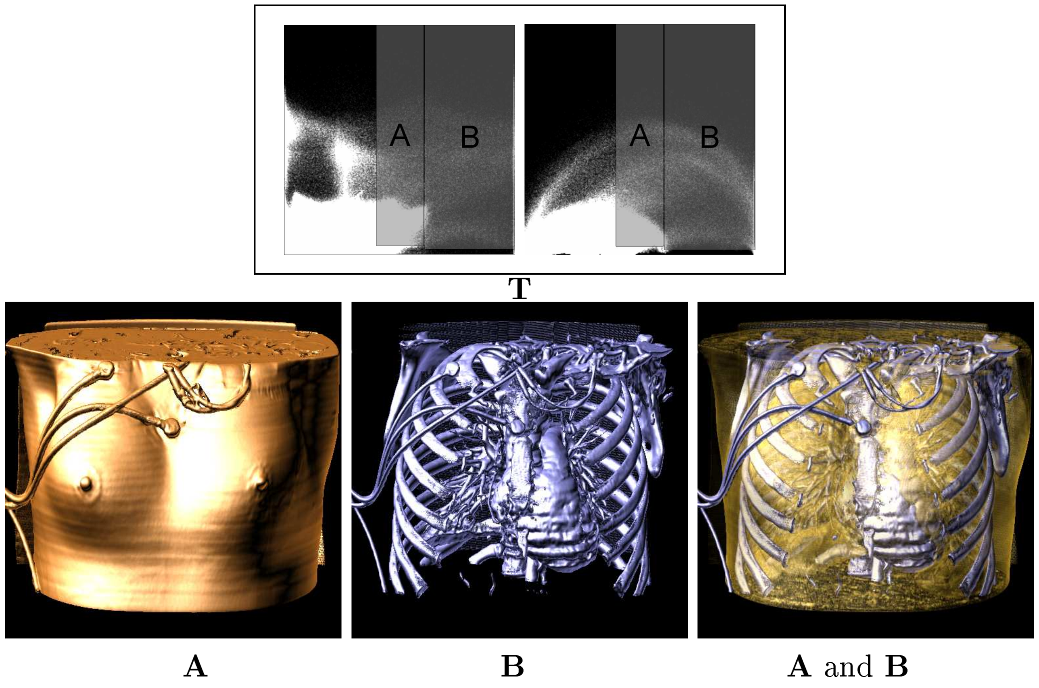


Fig. 8. Feature Selection of the Torso Dataset: The selected areas in **T** are rendered in the images of **A** and **B**.

References

[1] J. L. Buchanan and P. R. Turner. *Numerical Methods and Analysis.* McGraw-Hill, Inc., 1992.

[2] J. Canny. Finding edges and lines in images. Master's thesis, Massachusetts Institude of Technology, 1983.

[3] J. J. Clark. Authenticating edges produced by zero-crossing algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):43–57, 1989.

[4] R. Fernando and mark J. Kilgard. *The Cg Tutorial.* Addison-Wesly, 2003.

[5] S. F. F. Gibson. Constrained elastic surface nets: generating smooth surfaces from binary segmented data. In *Medical Image Computation and Computer Assisted Surgery*, pages 888–898, 1998.

[6] S. F. F. Gibson. Using distance maps for accurate surface representation in sampled volume. In *Volume Visualization Symposium*, pages 23–30. IEEE, 1998.

[7] G. Kindlmann. Multidimensional transfer functions for interactive volume rendering: Design, interface, interaction. *SIGGRAPH Course Notes*, 8(3), 2002.

[8] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of the 1998 IEEE Symposium on Volume Visualization*, pages 79–86, October 1998.

[9] J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the Conference on Visualization*, pages 255–262, October 2001.

[10] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002.

[11] M. Levoy. Display of surfaces from volume data. *Computer Graphics and Applications*, 8(5):29–37, 1988.

[12] NVIDIA. *NVIDIA OpenGL Extension Specifications.* NVIDIA Corporation, December 2004. http://developer.nvidia.com/object/nvidia-opengl-specs.html.

[13] S. Park and C. Bajaj. Multi-dimensional transfer function design for scientific visualization. In *Fourth Indian Conference on Computer Vision Graphics and Image Processing, ICVGIP*, pages 23–30, December 2004.

[14] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 839–846, January 1998.

[15] D. Ziou and S. Tabbone. Edge detection techniques - an overview. *Pattern Recognition and Image Analysis*, 8(4):537–554, 1998.