# Multi-Dimensional Transfer Function Design for Scientific Visualization

Sangmin Park
Univ. of Texas at Austin
smpark@cs.utexas.edu

Chandrajit Bajaj
Univ. of Texas at Austin
bajaj@cs.utexas.edu

## Abstract

*Direct volume rendering of field data can be accomplished through a correct choice of transfer functions that map data values to visual properties such as transparency and color. Such one-dimensional transfer functions, especially when specified through trial and error selection, often prove inadequate for producing correct and informative visualizations. In this paper, we consider multi-dimensional transfer functions for scalar fields which utilize field gradient magnitude and second derivative information, in addition to data values, to discriminate features with overlapping ranges. Our automatic method is further distinguished by our setting of voxels opacities, based on their spatial distance to boundaries in the specified volumetric scalar field data.*

## 1. Introduction

One of the main advantages of direct volume rendering is to visualize volume data without generating geometric structures. It simply converts each volume element (or voxel) to visual properties such as opacity and color and composes the properties into an image. The converting work is usually done by transfer functions. Good transfer functions produce good images that visualize interesting structures in volume data while removing uninteresting area. However, it is known that finding good transfer functions is one of the hardest problems in the direct volume rendering area.

The multi-dimensional transfer functions are the mathematical functions of several parameters such as intensity, gradient magnitude and the second derivative. Since the functions are hard to control by hand, many people still use intensity-based one-dimensional (1D) functions. Kindlmann's survey [6] reviews many types of the functions and describes the reasons why it is not a trivial work. It also explains that the functions can be generalized by increasing the function's domain. Kniss and et al. [8] shows the advantage of multi-dimensional functions against 1D functions. Even though multi-dimensional functions are hard to control, the functions are still useful for separately visualizing several features that intensity ranges are overlapped, but other data (i.e. gradient) are distinguishable.

When multi-dimensional functions are designed, intensity has been used for the first axis. The second axis of the functions has been gradient in [11], [7], [8], [9], and [10] without any doubt for more than a decade. The main reason is that some region of volume data with high gradient is likely to have a boundary (or a feature). For the third axis, the directional second derivative along gradient direction is used in [8], [9], and [10].

Kindlmann and Durkin [7] suggests a semi-automatic method to find boundaries in the 2D space of intensity and gradient through histogram volume inspection. Kniss and et al. [8] developed a convenient interface with manipulation widgets to search boundaries in 3D space of intensity, gradient and the directional second derivative. However, the method suggests a way to search boundaries using gradient and the second derivative that have no direct boundary information.

Once scalar volume datasets (i.e. medical data) are generated, the role of each voxel is fixed in terms of boundary. In other words, some of the voxels are exactly on a boundary, some are a part of the boundary with some thickness and others are not related to any boundary. If we know the role of each voxel, then transfer function generation will be more intuitive and easier. In this paper, we present a method to decide a voxel role in terms of boundary and suggest another data values for the second axis and the third axis for multi-dimensional transfer functions.

The remainder of the paper is organized as follows. In the next section, we review related work. In section 3, we present a method to decide a voxel role in volume data in terms of boundary. Based on the voxel roles, we suggest new opacity functions in section 4. In section 5, The functions are implemented using modern graphics hardware and the rendering results are explained. Finally, we make conclusions and suggest a couple of future work in section 6.

## 2. Related Work

Levoy [11] suggested two-dimensional transfer functions of intensity and gradient magnitude and suggested semi-transparent multiple surface visualization. After the research, multi-dimensional transfer functions have been the function of intensity and gradient. Kindlmann and Durkin [7] suggested semi-automatic generation of transfer functions. They still used two-dimensional transfer functions of intensity and gradient. The semi-automatic generation algorithm uses second derivatives to automatically compute boundary thickness at some intensity and uses a linear function to assign alpha values to the thickness. Kniss and et al. [8] suggested three-dimensional transfer functions and interactive interface widgets. The Kindlmann's survey [6] explains many kinds of transfer functions, but all multi-dimensional transfer functions are the functions of intensity, gradient, and the second derivative.

Distance map have been used for volume rendering in [4] and [5] for the visualization of binary segmented volume data. The research gave us a hint on the multi-dimensional transfer function design, since the distance map has an interesting character like that: the zero-value iso-surface of the distance map yields the object surface.

## 3. Boundary Detection and Voxel Roles

Many boundary detection (or edge detection) algorithms have been developed in the image processing and pattern recognition areas. One of the traditional edge detection techniques is Canny's method that finds the local maxima along the gradient direction [2]. The most common edge detection schemes include three operations: differentiation, smoothing and edge labeling [14].

First, differentiation is the computation of the derivatives to identify edges. We compute gradient vectors represented by $\bigtriangledown f$ using the central difference operator and the magnitude of the gradient vector is $\parallel \bigtriangledown f \parallel$. The normalized gradient vector is computed as following:

$$\vec{n} = \frac{\bigtriangledown f}{\parallel \bigtriangledown f \parallel} \tag{1}$$

The frequently used second-order derivative operators are the Laplacian operator and the directional second-order derivative. In this paper, we compute and use the directional second-order derivative along gradient direction. The operator is defined by:

$$\frac{\partial^2 f}{\partial^2 \vec{n}} = \vec{n} \cdot \bigtriangledown \parallel \bigtriangledown f \parallel \tag{2}$$

Second, for the smoothing purpose, a bilateral filter that smoothes data values while preserving edges [13], is applied to gradient and the directional second derivative.

Finally, edge labeling is to identify authentic edges while suppressing false edges produced by the reasons of noise and non-maximum high gradient. Since we assume that volume data have reasonably high signal-to-noise ratio and some noise that can be accumulated in the first and second derivative computations is reduced by a bilateral filter, we consider only removing the "phantom edges" (defined in [3]) or non-maximum gradient. Ziou and Tabbone's survey [14] shows several ways to remove the phantom edges.

Phantom edges can be distinguished easily by climbing a gradient mountain along gradient direction, while a voxel role is decided. In the next section, we present the phantom edge removing in detail.

## 4. Distance as a Voxel Role

Since the direct volume rendering does not need any geometric structure, it is enough to find proper transparencies for all voxels. If we know the roles of each voxel in terms of boundary, then we can assign correct transparencies to the voxels easily based on the voxel roles. In other words, if a voxel is exactly on a boundary, then the voxel should be totally opaque, while a voxel that is a part of the boundary with some thickness should have proper transparency.

In this section, we define voxel roles with distance from an authentic edge. The distance is the Euclidean distance along gradient direction in 3D space. The distance is computed by shooting two rays to both positive and negative gradient directions at every voxel location. The two values of gradient and the directional second derivative are interpolated by tri-linear interpolation at every sampling location. If gradient decrease at one of the directions, then the other direction is taken and keep sampling until it hits a zero-crossing location of the second derivative. Since gradient direction is perpendicular to the edge orientation [14], if we follow one of the gradient directions, we will find a zero-crossing locations or a boundary in 3D space.

Fig. 1 shows a 2D example on computing a distance from a voxel to an authentic edge along the negative gradient direction. Ideally, the two values of gradient and the directional second derivative are changed like Fig. 2 along both positive and negative gradient directions.

A sampling location is represented by $\vec{x}_s(t) = \vec{x} + t\frac{\bigtriangledown f(\vec{x})}{\parallel \bigtriangledown f(\vec{x}) \parallel}$, where $-d_{max} < t < d_{max}$. If the two signs are changed at the two consecutive sampling locations of $t_1$ and $t_2$ as following, $f''(\vec{x}_s(t_1)) \times f''(\vec{x}_s(t_2)) < 0$, then we compute the exact zero-crossing location with the bisection method, [1]. Experimentally, we decide 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).

Fig. 3 shows the results of the distance and the directional second derivative computation as well as each dataset

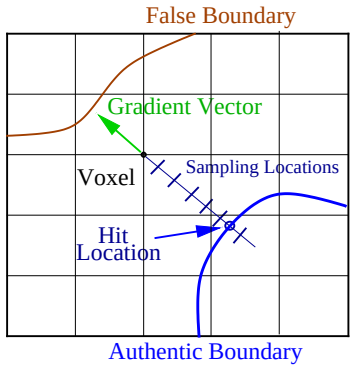



**Figure 1. The voxel role or distance computation by sampling from a voxel to an authentic edge along the gradient direction of the voxel**

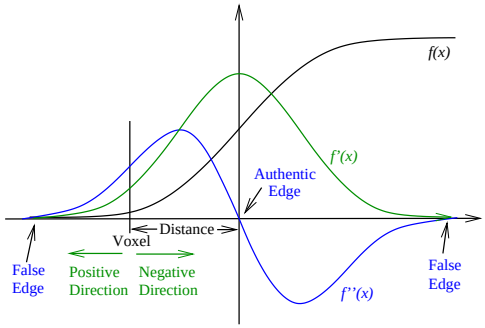


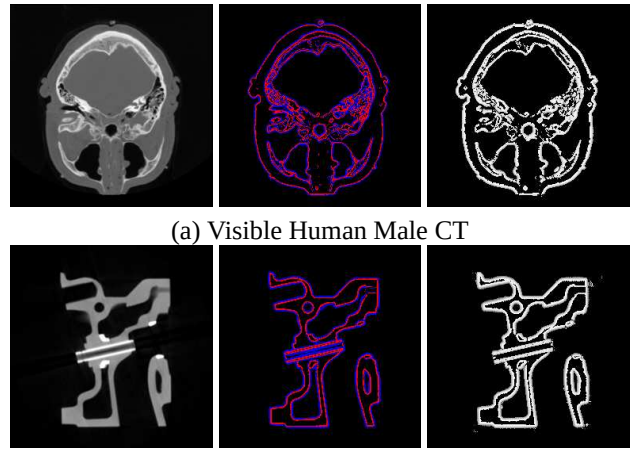

**Figure 2. The authentic and false edges and the relations of $f$, $f'$, and $f''$**

slice. The positive and negative second derivative values are colored with blue and red respectively. Therefore, the zero-crossing locations of the second derivative are between the two colors. To denote the distance values through images, we linearly flip the distance values. For example, if the distance values at $d(\vec{x})$ ranges from 0 to $d_{max}$, then we simply compute the following formula for each pixel value of Fig. 3 (right) and Fig. 4.

$$D(\vec{x}) = \frac{d_{max} - d(\vec{x})}{d_{max}} \times 255 \qquad (3)$$

## 5. Multi-Dimensional Transfer Functions

In this section, we suggest the opacity functions based on the distance from a voxel to an authentic edge. We also suggest the 2D opacity functions of intensity and gradient magnitude at hit locations. The two kinds of opacity functions are multiplied to generate the final opacities of each voxel.



(a) Visible Human Male CT

(b) Engine

**Figure 3. Volume Data Slice(left), the Directional Second Derivative(middle) and Distance(right): The red and blue colors of the middle column represent the negative and positive values of the directional second derivative respectively.**

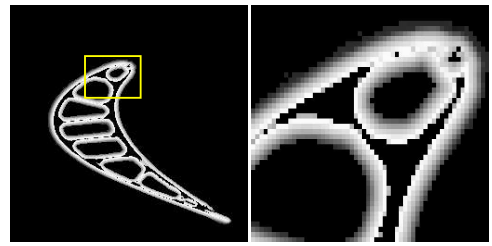


**Figure 4. Turbine Blade Distance Image: The right-hand side image is the enlarged picture of the left image of the yellow box.**

### 5.1 Opacity Functions of Intensity

Once we decide each voxel role in terms of boundary, the transparency of each voxel can be generated easily based on the role or the distance that is computed in the previous section. We suggest three different opacity functions, linear, nonlinear concave, and nonlinear convex functions as Fig. 5. The linear opacity function is to map the flipped distance and the concave and convex nonlinear functions use the $n$-th power of the distance as following:

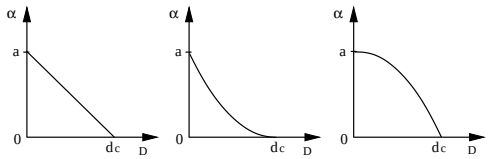
$$\alpha_d(d) = Max(-\frac{a \times d}{d_c} + a, \quad 0), \qquad (4)$$

$$\alpha_d(d) = \begin{cases} \frac{a}{d_c^n}(\mid d - d_c \mid)^n & \text{if } d < d_c \\ 0 & \text{others} \end{cases}, \qquad (5)$$

and

$$\alpha_d(d) = Max(-\frac{a \times d^n}{d_c^n} + a, \quad 0), \tag{6}$$

where $0 < d_c \leq d_{max}$, $0 \leq a \leq 1$, and $n > 1$. Eq. 4, 5 and 6 are linear (Fig. 5 (a)), nonlinear concave (Fig. 5 (b)), and nonlinear convex (Fig. 5 (c)) opacity functions respectively. The location of the three opacity functions are controlled with $d_c$ and $a$ and $n$ dominates the shape of the nonlinear functions.




(a) Linear Map (b) Concave Nonlinear (c) Convex Nonlinear

**Figure 5. Alpha Maps**

Since the opacity function, $\alpha_d(d)$, is computed based on only distance, we define another opacity function of intensity like $\alpha_u(v)$ that is controlled by a user. The final opacity value is computed by the multiplication of the two opacity functions as following, $\alpha_u(v) \times \alpha_d(d)$. The opacity function has both of user control and automatic opacity generation. While a user turn on some range of the intensity values with $\alpha_u(v)$ by assigning a totally opaque value such as 1, $\alpha_d(d)$ automatically generate the opacities of each voxel with the alpha map of Fig. 5. $\alpha_u(v)$ also provides the ramps of traditional 1D transfer functions.

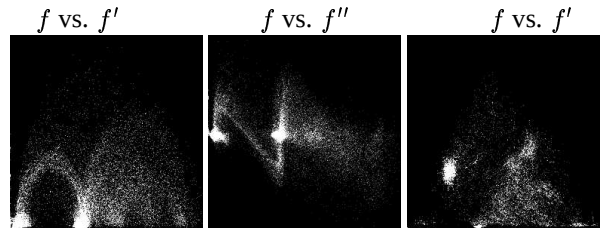
### 5.2 Gradient Magnitude for the Second Axis

Most multi-dimensional transfer functions have gradient magnitude for the second axis, while intensity works as the first axis. The 2D functions of intensity and gradient are more powerful than 1D functions. However, as the function domain is extended to 2D space, it is much harder to control by hand, since we need to search the combined ranges of intensity and gradient.

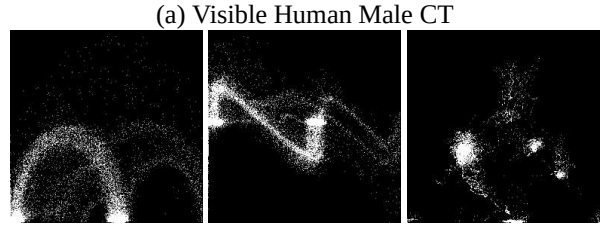
We assume that most boundaries of volume data have some thickness and all voxels that are in a thick boundary (or boundary voxels) should be visualized with proper transparencies. Therefore, the gradient range of a feature can vary from a relatively small value to a big value in a thick boundary. Even though all voxels opacities of a volume dataset are decided by some pre-processing, it is not a trivial work to collect boundary voxels through searching a gradient range.

To reduce the searching time of the gradient ranges, we replace each voxels gradient value with the interpolated value at the hit location. When we compute the distance as in Fig. 1, the gradient at the hit locations is interpolated by tri-linear interpolation. If we use the interpolated value at hit locations as the second axis, then it will make the searching work easier, since we only have to consider the gradient ranges at a boundary.

The 2D opacity function of intensity and gradient is represented as $\alpha_u(v, g)$, where $g$ represents gradient of each voxel at the hit location. 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 [8]'s function, but the opacity values of each voxel can be 1 always, since the opacity function of distance, $\alpha_d(d)$, generates alpha values. A user only have to select some regions to be visualized in the 2D space of intensity and gradient.




(a) Visible Human Male CT



(b) Engine

**Figure 6. The Graphs of $f$ vs. $f'$ (1st Column), $f$ vs. $f''$ (2nd Column), and $f$ vs. $f'$ (3rd Column): The $f$ and $f'$ values of the 3rd column graphs are interpolated at the hit locations of the zero-crossing second derivative. Each slice of (a), (b), and (c) of Fig. 3 is used to generate these graphs.**

The 3rd column of Fig. 6 shows the graphs of intensity and gradient at the hit locations that are described in Fig. 1. The each slice of Fig. 3 is used for the graphs. We can easily recognize that each blob of the 3rd column graphs of Fig. 6 represents a boundary and the blobs are usually located in the local maxima of gradient (the 1st column graphs of Fig. 6) or in the zero-crossing locations of the second derivative (the 2nd column graphs of Fig. 6).

## 6 Implementation and Results

We have implemented a 3D texture-based volume renderer using nVidia graphics cards such as GeForce3, 4, and FX. Since the cards provide at least four 3D multi-textures and dependent texture reads with register combiners, the mult-dimensional transfer functions can be implemented on

a PC equipped with those graphics cards. Fig. 7 shows the rendering pipeline in nVidia GeForce cards. In the rendering pipeline, dependent texture is used for implementing the opacity function, $\alpha_u(v, g)$ that is controlled by a user and assigning colors to each voxel with the color function of $v$ and $g$.

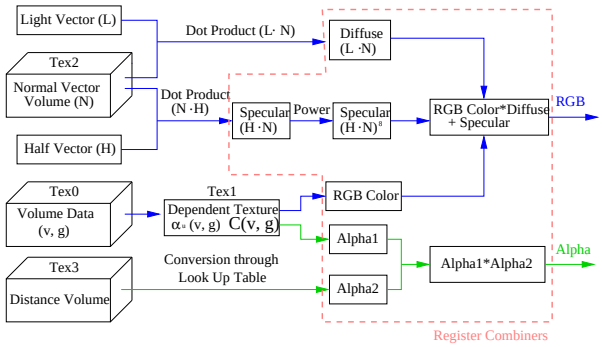
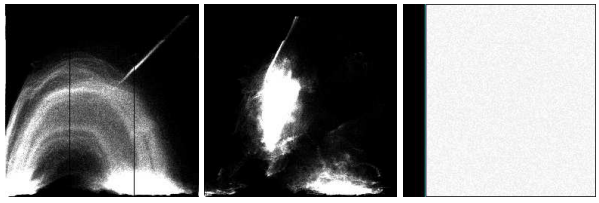



**Figure 7. Rendering Pipeline in a nVidia GeForce Card: Three texture volume datasets feed into the register combiners and a RGB color and an alpha value are computed with the register combiners. The solid lines represent data flow and processes and the dashed line indicate the register combiners.**

The volume rendering pipeline requires the six components, RGB normal, intensity ($v$), gradient ($g$), and distance ($d$), for each voxel. If we want to visualize a $256^3$ volume dataset, (=16 Mbytes), then we need at least $256^3 \times 6$, texture memory. To reduce the texture memory requirement, our implementation relies on hardware assisting texture compression that is one of the ARB OpenGL extensions. Especially, we use the s3tc texture compression format [12] that is provided by the nVidia graphics cards.



(a) (b) (c)

**Figure 8. The Turbine Blade Graphs and an Alpha Map: (a) is the general graph of $f$ vs. $f'$, (b) is the graph of $f$ vs. $f'$ at Hit Locations and (c) is the alpha map, $\alpha_u(v, g)$, defined by a user**

We generated images to test the distance-based multi-dimensional transfer functions with alpha maps. First, all voxels are turned on by assigning 1 to all voxels that are in the white region of Fig. 8 (c) and we adjust the alpha map parameters of $a$ and $n$ and fixed $d_c$ like Fig. 9. When we want to visualize thick and opaque boundaries of a volume dataset, then the convex alpha map of Fig. 9 (b) will be useful, while the concave alpha maps of Fig. 9 (c) and (d) are useful to make boundaries thinner. Since the gradient of a voxel is replaced with the value at the hit location of the zero-crossing second derivative, it is easy to decide a gradient range.

One of the main reasons that make transfer function generation hard is a huge number of degrees of freedom. Even though we consider only 1D transfer functions, each control point has the two degrees of freedom in intensity and transparency axes. However, in the proposed multi-dimensional functions, the number of degrees of freedom is reduced very much, since the alpha values of each voxel are decided by the alpha function, $\alpha_d(d)$, automatically. Plus, since the shape and location of $\alpha_d(d)$ can be controlled by the three parameters, $a$, $d_c$, and $n$, a user can adjust the transparency effect.

## 7. Conclusions and Future Work

In this paper, we presented a new multi-dimensional transfer functions. The proposed functions can reduce the number of degrees of freedom radically, since the alpha values of all voxels are decided automatically with the three parameters, $a$, $d_c$, and $n$ based on distance to the zero-crossing location of the second derivative. Plus, the searching time of the gradient magnitude range can decreases, since each voxels gradient is replaced with the value of the hit location.

For the future work, we need to make some dependency of the two different alpha functions of $\alpha_u(v, g)$ and $\alpha_d(d)$. It will provide differenct $\alpha_d(d)$ for some different $v$ and $g$. Another improvement of this paper is to show how much bilateral filters can improve Canny's edge detector. Originally, Canny's edge detector is combined with Gaussian filters.

## 8. Acknowledgments

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

## 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. Technical report, 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] S. F. F. Gibson. Constrained elastic surface nets: generating smooth surfaces from binary segmented data. In *Medical Image Computation and Computer Assisted Surgery*, 1998.

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

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

[7] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions dor direct volume rendering. In *Proceedings of the 1998 IEEE Symposium on Volume Visualization*, October 1998.

[8] 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*, October 2001.

[9] 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.

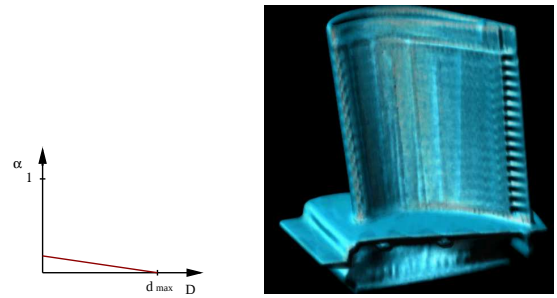
[10] J. Kniss, S. Premože, M. Ikits, A. Lefohn, C. Hansen, and E. Praun. Gaussian transfer functions for multi-field volume visualization. In *IEEE Visualization 2003*, October 2003.

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

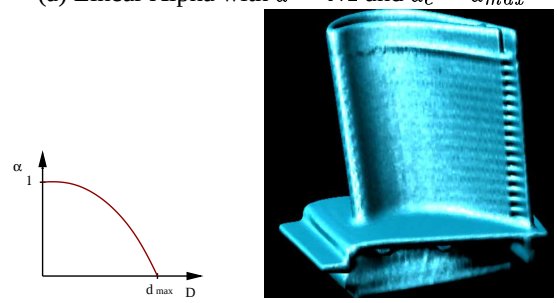
[12] NVIDIA. *OpenGL Extension Specifications*. NVIDIA Corporation, March 2004. http://developer.nvidia.com/page/home.

[13] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the IEEE International Conference on Computer Vision*, January 1998.

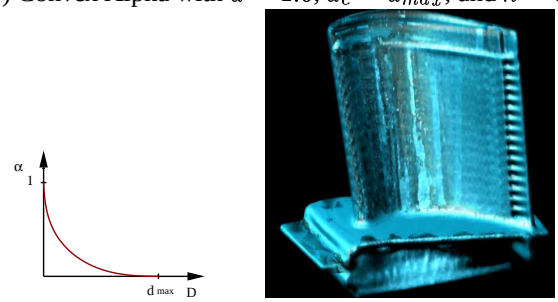
[14] D. Ziou and S. Tabbone. Edge detection techniques - an overview. *Pattern Recognition and Image Analysis*, 8(4):537–554, 1998.



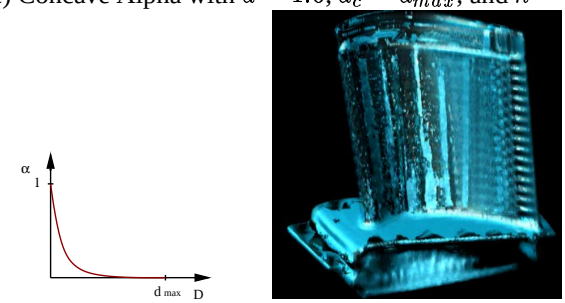

(a) Linear Alpha with $a = 0.1$ and $d_c = d_{max}$




(b) Convex Alpha with $a = 1.0$, $d_c = d_{max}$, and $n = 3.0$




(c) Concave Alpha with $a = 1.0$, $d_c = d_{max}$, and $n = 5.0$




(d) Concave Alpha with $a = 1.0$, $d_c = d_{max}$, and $n = 7.0$

**Figure 9. Turbine Blade Rendering with Several Alpha Maps**