12-1-2009

# Supervised manifold distance segmentation

Guany Wang

Guanyu Wang
*Candidate*

Computer Science
*Department*

This thesis is approved, and it is acceptable in quality and form for publication:

*Approved by the Thesis Committee:*

[illegible] ,Chairperson

[illegible]

[illegible]

# SUPERVISED MANIFOLD DISTANCE SEGMENTATION

**BY**

**GUANYU WANG**

**BE., COMPUTER SCIENCE AND TECHNIQUE,**
**XIAN UNIVERSITY OF TECHNOLOGY, 2005**

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

**Master of Science**
**Computer Science**

The University of New Mexico
Albuquerque, New Mexico

**December 2009**

## ACKNOWLEDGMENTS

I deeply and sincerely acknowledge Dr. Joe Kniss, my academic advisor and defense chair, for encouraging and understanding me in my entire master student career. His professional guidance and creative instruction is the basis of this thesis.

I am also thankful to my committee members, Dr. Pradeep Sen and Dr. Lance Williams, for their valuable recommendations and constructive comments.

To Hairong Lei, my cooperator gave me immeasurable support all the time.

# SUPERVISED MANIFOLD DISTANCE SEGMENTATION

**BY**

**GUANYU WANG**

ABSTRACT OF THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

**Master of Science**

**Computer Science**

The University of New Mexico

Albuquerque, New Mexico

**December 2009**

**SUPERVISED MANIFOLD DISTANCE SEGMENTATION**

**by**

**Guanyu Wang**

**BE., Computer Science and Technique, Xian University of Technology, 2005**

**MS., Computer Science, University of New Mexico, 2009**

## ABSTRACT

In this paper, I will propose a simple and robust method for image and volume data segmentation based on manifold distance metrics. In this approach, pixels in an image are not considered as points with color values arranged in a grid. In this way, a new data set is built by a transform function from one traditional 2D image or 3D volume to a manifold in higher dimension feature space. Multiple possible feature spaces like position, gradient and probabilistic measures are studied and experimented. Graph algorithm and probabilistic classification are involved. Both time and space complexity of this algorithm is O(N). With appropriate choice of feature vector, this method could produce similar qualitative and quantitative results to other algorithms like Level Sets and Random Walks. Analysis of sensitivity to parameters is presented. Comparison between segmentation results and ground-truth images is also provided to validate of the robustness of this method.

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# 1. introduction

Image segmentation is an important task in many applications. This paper shows an algorithm that segments 2D image and 3D volume for interactive visualization applications. Users are required to define characteristics of each object to get segmentation or classification results for decision making. When recognizing these characteristics, an effective algorithm should be able to handle and analysis uncertainty in the whole segmentation procedure. In this algorithm, we preserve the uncertainty inherently in data for all measurement, classification and visualization processes. To achieve this, instead of assigning every image element with one certain label, we compute probabilistic distributions. Based on user specified seeds and a geodesic distance metric, distances are calculated along connected image elements. The distances could be converted into probabilistic distributions, which capture the uncertainty in boundary placement. Another part contributes to our algorithm is to compute probabilistic classifiers based on samples, the user specified seeds. Parzen Window is chosen as the classifier which provides our algorithm more flexibility. For small and isolated areas without any seed inside, this classifier can help the algorithm to correctly recognize them. This is meaningful for users because it saves them from putting seeds into all disconnect areas. In this sense, this algorithm is a combination of segmentation and classification. It only takes $O(N)$ in time complexity and provides high-quality visualization results for effective decision making.

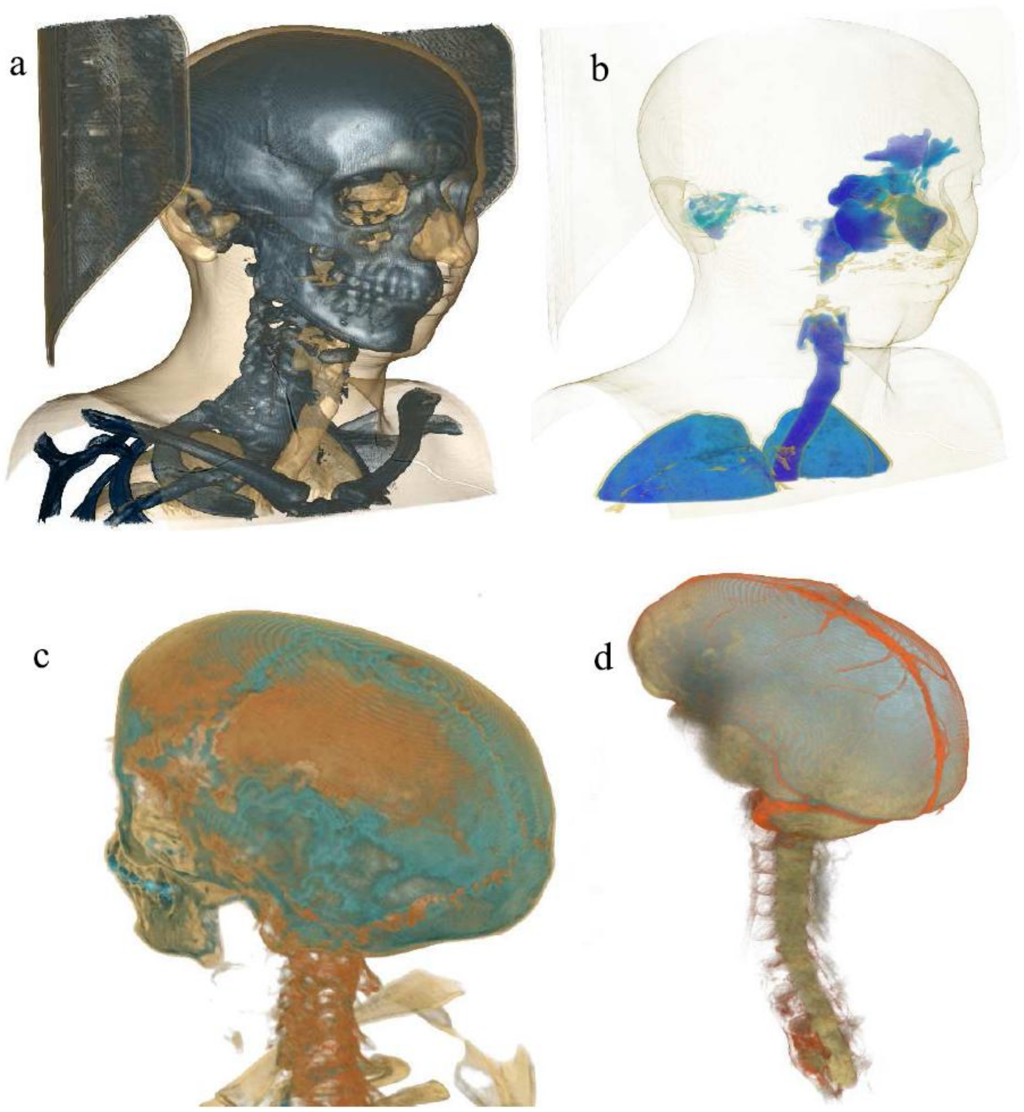


Figure 1. Comparison of Different Approaches

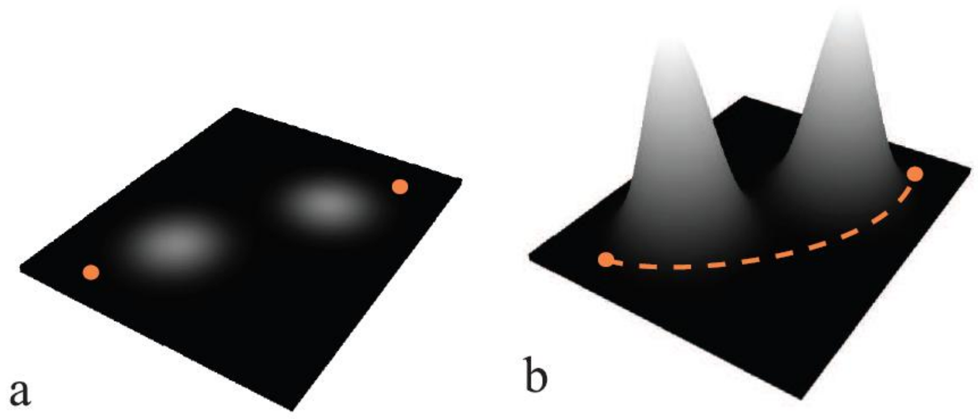
(a) is a 3D image shown directly by volume rendering. It tries to show bone object but it also shows the board. (b)(c)(d) are results generated from Manifold Distance Segmentation. This algorithm offers a fast and simple way to locate objects. It is effective and valuable especially when one object is hard to distinguish from other based on values and derived measure, like brain and soft tissue in this image. The shown $256 \times 256 \times 256$ 3D segmentation result finished in 10 minutes including user interaction.

## 2. Background

In the visual analysis of 3D data like medical image or physical simulation [1], volume rendering methods are widely used for many purposes. Commonly they utilize transform functions to map original data images into other visible images. For instance, different features could be mapped in different color channel, values could be mapped as color value and data with certain attributes could be set as highlighted or transparent. These functions usually only care about original data values or values derived locally, like derivative. However, in this way it cannot distinguish disjoint area with certain similar characteristics. A common way to implement transform functions is to build look-up tables. Although this approach is simple and effective for interactive applications, it limits the dynamic range and dimension of feature space. When the table expend in dimension, it will be inevitable to take large chunk of memory. Another disadvantage of this approach is that it always requires users with high capacity in understanding the abstract mapping relationship from aimed data to certain constrains of vector value. Therefore, in some critical situations lookup tables cannot guarantee convenient interaction and accurate results for making decision. There are some previous works on incorporating uncertainty information in volume rendering applications. Kniss *et al.* [2] introduced a high-quality probabilistic segmentation method of features as part of the visualization pipeline. In that case, the transform function utilizes class probabilities rather than simply maps feature data values into different color or opacity. The boundaries among different classes are no longer static but variable for more flexible identification. Compared to their work focusing on rendering, our work aims at handling probabilistic segmentations in interactive volume visualization application.

## 2.1 Image Manifolds

Normally we consider an image as a continuous data set with coordinates. For example, a 2D gray level image is a data set of gray value with x, y coordinates. If we combine the data value and original coordinates together, we could get higher dimension coordinates then the image itself becomes a surface in the space-feature domain. This is called space-feature manifold. In this way, a 2D gray level image could be considered as a 3D manifold with coordinates of x, y and gray level. To show this image in 3D scene, we treat the gray level coordinate as z coordinate. Figure 2 illustrate surfaces in 3D space.



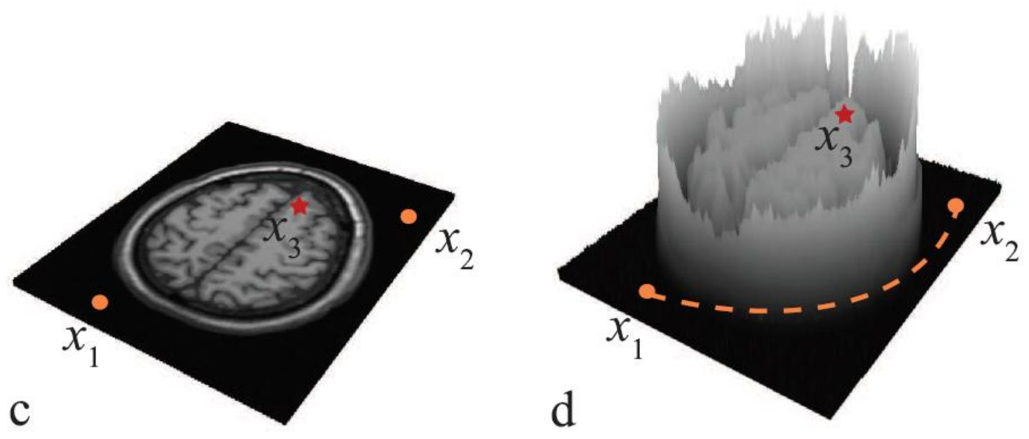


Figure 2. Simple Example of Manifold Image

(a)(c) are 2D gray level image, in which shortest distance between two points is Euclidean distance. (b)(d) are 3D manifold where height is transformed by pixel value. The geodesic distance is the length of shortest curve on the surface.

### 2.2 Image Segmentation

Image segmentation is usually defined as breaking image into connected regions relative to different objects. From user's perspective, image segmentation algorithms could be divided into two classes, supervised segmentation and unsupervised segmentation, based on whether requires user input or not, respectively. Some popular algorithms are introduced here and will be compared to our algorithm.

Level Set [3] is a supervised segmentation method which utilizes an implicit function over the image domain. Non-linear partial derivative equation is always used by the implicit function. The image is segmented with specified levels. In this method, users are required to set seeds to each label and parameters to determine the function. Another supervised segmentation algorithm is Random Walks [4]. This algorithm also needs to specify seeds for an image. These seeds for each label are considered as start points or 0 distance points, from which distances for all other non-seed points are calculated. Similar to our algorithm, one label corresponds with shortest distance for a point is assigned as the label of this point. Watershed segmentation [5] is an unsupervised algorithm which only needs pre-defined parameters rather than seeds for each image. In this algorithm, value of each pixel is considered as height of that position. Water continuously fills into the image field and these disjoint water areas will consciously merge together finally into one. With a certain height constrain, connected area is consider as one object. This method is competitive in running time but easy to get many small pieces as result. Mean Shift [6,7] is another widely used unsupervised segmentation that iteratively merges points with their most similar neighbors. The similarity is calculated by local static of the data. This algorithm also used space-feature manifold of an image.

## 3. Definitions

In this section we present mathematical definitions used in this thesis. These definitions here assume volumetric images like 3D images with a domain which is subset of $\mathbb{R}^3$. Actually they also generalize to 2D and higher dimension ones like 4D images use time as the fourth component. Section 3.1 provides a formal definition of an image as a smooth function. Section 3.2 introduces the image manifold we use, which is a surface in a space with coordinate combining original spatial position and data value. In this section we also show how to calculate distance on this surface. Section 3.3 describes the transformation among distance, probability and energy.

## 3.1 The Image Function

Our work focuses on a continuous fuzzy classification of a smooth continuous image. As for a 3D volume image, $\vec{x}$ is a point in original spatial domain $\Omega \subset \mathbb{R}^3$ and $\vec{y}$ is a point in the feature space domain $\mathbb{R}^v$. Since spatial coordinate is part of feature space coordinates and one point in spatial domain could only be transformed into one point in feature space. We can define a mapping function

$$I_v : \vec{x} \in \Omega \to \vec{y} \in \mathbb{R}^v \quad (1)$$

Pixel or point $p_i$ is discrete sample in $\mathbb{R}^v$. Given a data value set $V_i$ in $\mathbb{R}^v$, an image can be represented as a convolution function

$$I_v(\vec{x}) = \int_\Omega k(\vec{s} - \vec{x}) \sum_{i=1}^{n} \overrightarrow{V_i} \delta(\vec{s} - \overrightarrow{p_i}) d\,\vec{s} \quad (2)$$

where $\delta$ is a Dirac Delta function and k is a interpolation function. In another way, this expression could be

$$I_v(\vec{x}) = \sum_{i \in N_x} \overrightarrow{V_i}\, k(\overrightarrow{p_i} - \vec{x}) \quad (3)$$

where $N_x$ is the neighboring lattice points of x.

Equation (2) and (3) are equivalent and give the formal concept of image interpolation. With these equations we can compute value at any position inside domain $\Omega$. Different interpolation function k could bring different degrees of continuity. Equation (2) utilizes convolution function which is widely used in signal processing realm. Equation (3) is more practical from software engineering prospective.

### 3.2 The Image manifold

For a 3D volume with spatial domain $\Omega \subset \mathbb{R}^3$, we can consider it as manifold $I_v$ in higher dimension space $\mathbb{R}^3 \times \mathbb{R}^v$. This could be expressed as

$$I_v \hookrightarrow \mathbb{R}^3 \times \mathbb{R}^v = \{(\vec{x}, I_v(\vec{x})) | \forall \vec{x} \in \Omega\} \quad (4)$$

This manifold combines original spatial position and values in feature space. It is often called space-feature manifold. Two simple examples of this are given in figure 2. In this figure, we show two 2D gray images and their manifolds in 3D space. In Figure 2(a), we can see there are two points located on the plain of the image. The shortest distance of these two points is the 2D Euclidean distance. In figure 2(b), the shortest path could be a curve avoiding the bumps. In figure 2(c), there are points $x_2$ and $x_3$ which closer than points $x_1$ and $x_3$ on a 2D plane. But in figure 2(d), $x_1$ could be closer to $x_2$ than $x_3$. Because the path from $x_1$ to $x_2$ is flat while the path from $x_3$ to $x_2$ very bumpy and that brings longer distance.

Given two points $p_1$ and $p_2$, we can define the path metric $d_v(p_1, p_2)$ on an image manifold as the geodesic from $p_1$ to $p_2$ on the manifold surface. Then the shortest distance is the length of the geodesic. With this concept, a precise definition of metric tensor G is

$$g_{i,j} = \alpha_i \delta_{ij} + \sum_{k=1}^{v} {\beta_k}^2 \frac{\partial I_k}{\partial \vec{x_i}} \frac{\partial I_k}{\partial \vec{x_j}} \quad (5)$$

where $\delta_{ij}$ is the Kronecker Delta, $\alpha_i$ and $\beta_k$ are scale factors controls the effects from position and value respectively. The default rule we used for these factor are $\alpha_i = \frac{1}{d_i}$

where $d_i$ is the size of the i-th dimension of the image and $\beta_k = \frac{s}{r_k}$ where $r_k$ is the range of the k-th image value and s is another scale factor with default value of 1. With this metric tensor we can calculate distance for infinitesimal changes in $\vec{x}$. The distance, length of a curve $c(\tau)$ with beginning point $\overrightarrow{x_1} = c(a)$ and end point $\overrightarrow{x_2} = c(b)$ is measured by

$$L = \int_a^b \sqrt{\sum_{i=1,j=1}^{D} g_{ij}(c(\tau))\left(\frac{dx_i}{dt}c(\tau)\right)\left(\frac{dx_j}{dt}c(\tau)\right)}\,dt \quad (6)$$

We want to calculate distances of all unlabeled points from seeds, the pre-labeled points. Since images are sets of discrete points, we can consider them as graph and calculate the distances iteratively. At the beginning, we can compute distance for all neighboring points, then we can treat an image as a graph $G(V, E)$. Vertex set V is the set of all points in the image and edge set E is the set of all edges between every two neighboring points. Here we define neighboring point as point with distance equal or less than 1 along each spatial axis. For example, a point has 8 neighboring points in a 2D image, 26 neighboring points in a 3D image and 80 neighboring points in a 4D image. The edge weight is manifold distance between two points

$$w_{i,j} = \sqrt{\left\|\vec{\alpha}(\overrightarrow{x_i} - \overrightarrow{x_j})\right\|^2 + \left\|\vec{\beta}(I(\overrightarrow{x_i}) - I(\overrightarrow{x_j}))\right\|^2} \quad (7)$$

where $\overrightarrow{x_i}$ is the position of point i. To solve the shortest distance problem, the most common algorithm is Dijkstra's algorithm with time complexity of $O(N \log N)$. However, we can see that the graph we need to deal with is not a general graph. Under this condition, we can see the graph generated by an image is a planner graph with non-negative edges. This kind of graph allows us to use an updated algorithm significantly reduces time complexity compared to Dijkstra's algorithm. This updated algorithm will be introduced in section 4.

It should be pointed out that we assume the image is using a linear interpolation kernel. Sometimes people prefer high-order interpolation kernel, like cubic kernel such as Catmull-Rom. In this situation the Equation 7 could only be considered as an approximation. To compare our assumption with Catmull-Rom spline interpolation, we show two groups of images in Figure 3. This is a comparison between utilizing linear interpolation and cubic interpolation. We rescale our original images into 10 times larger images by Catmull-Rom interpolation and use our algorithm on the new images. The Left column shows distance map (Figure 3(a), Figure 3(c)) and labeled segmentation result (Figure 3(e)) from our original image. The right column shows the corresponding results generated from the cubic-interpolated image. Because the two experiments are using different scale images, to compare the absolute distance is meaningless. Therefore the distance map image showed here are all under normalization ranging from 0 to 1. Then we show the difference of distance results is less than 0.02 percent. In the final labeled segmentation results, we can find some isolated small areas are different, but totally these two results are qualitatively similar.

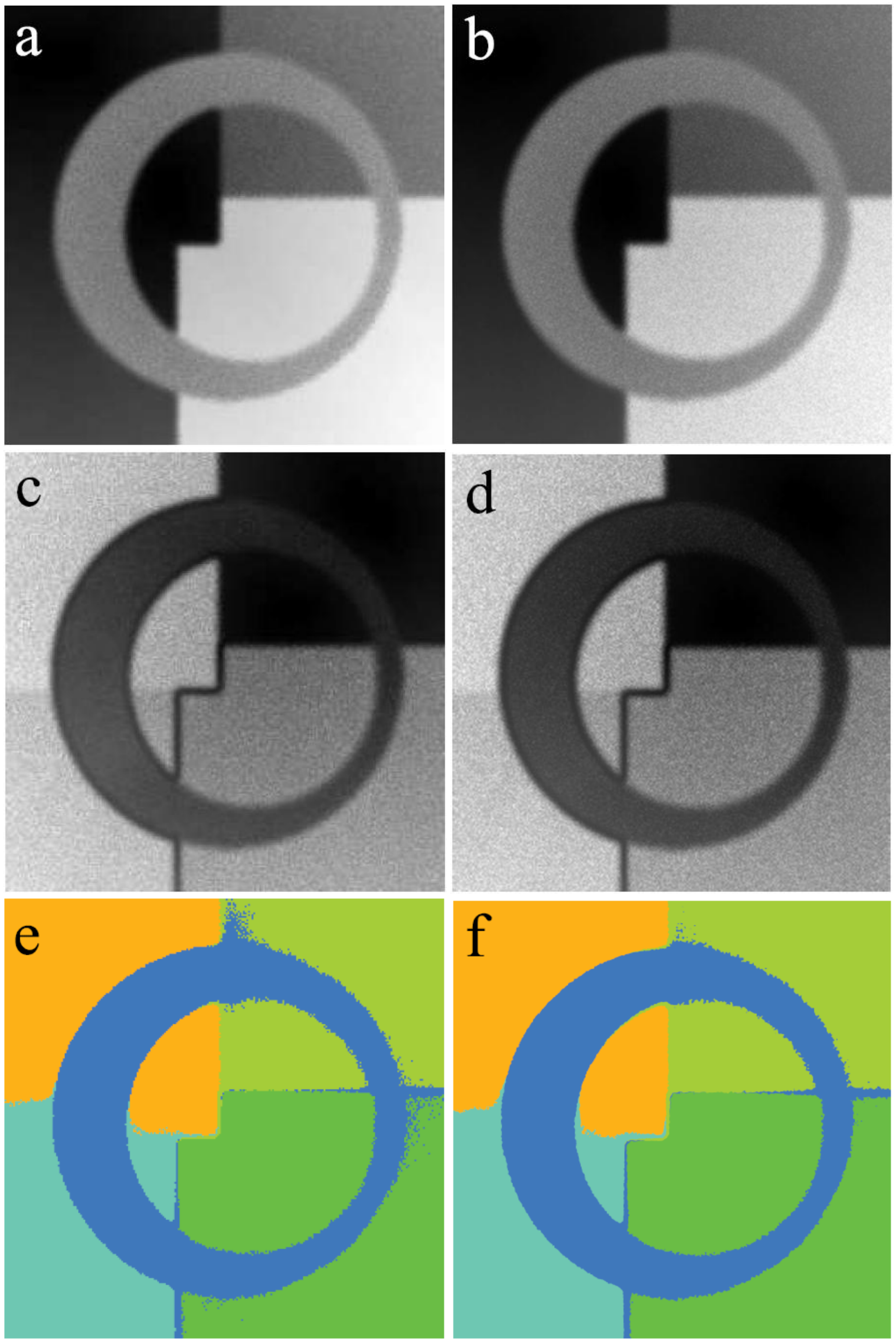


Figure 3. Different Interpolation Kernel

The left column is linear interpolation kernel on a $256 \times 256$ image and the right column is bicubic interpolation kernel simulation image. (a)(b) and (c)(d) are manifold distance for two objects. (e)(f) are labeled segmentation results.

## 3.3 Distance, Probability and Energy

Mahalanobis Distance is a way to determine the similarity of unknown data samples to known data samples which are pre-labeled seed points in our approach. Given a mean $\mu$ and covariance matrix $\Sigma$, Mahalanobis Distance is

$$d_M(x) = \sqrt{(x-\mu)^T\Sigma^{-1}(x-\mu)} \quad (8)$$

If we assume the samples follow normal distribution with mean $\mu$, the Mahalanobis Distance is equivalent to

$$d_M(x) \equiv \sqrt{-\ln(N(x) + \ln(\|\Sigma\|\sqrt{2\pi})} \quad (9)$$

where $N(x)$ is

$$N(x) = \frac{1}{\|\Sigma\|\sqrt{2\pi}} e^{-(x-\mu)^T\Sigma^{-1}(x-\mu)} \quad (10)$$

The $\frac{1}{\|\Sigma\|\sqrt{2\pi}}$ part is to ensure this probability density function could integrate to 1. With this relation between probability and distance, we can get distance function from any probability density function $p(x)$ that is

$$d_p(x) = \sqrt{-\ln(p(x) + \ln(\alpha)} \quad (11)$$

where $\alpha$ is a flexible given value which could bring different meaning to the distance function. For instance if $\alpha$ is set to be $p(\mu)$, that means the distance for $x$ equals to mean is 0. Given a distance function $d(x)$, the transformation function to a probability density function is

$$p_d(x) = \frac{1}{\int e^{-d(s)^2} ds} e^{-d(s)^2} \quad (12)$$

Here the distance $d_p(x)$ is different from the metric distance between two points. We only want to build a logically reasonable relationship between distance and probability. Then in our distance segmentation approach we can utilize properties from probabilistic segmentation or classification.

When transforming from energy to probability, we need to use Gibbs measure

$$p(x) = \frac{1}{Z(\beta)} e^{-\beta E(x)} \quad (13)$$

where $\frac{1}{Z(\beta)}$ plays a normalization role and $E(x)$ is the energy function, which could be expressed as

$$E(x) = -\ln(p(x) + \ln(\alpha) \quad (14)$$

where $\alpha$ is a factor similar to that in Equation (11). Then we can easily get the relation between energy and distance expressing as

$$d_E(x) = \sqrt{E(x)} \quad (15)$$

and

$$E(x) = d(x)^2 \quad (16)$$

From equations above we can see the minimum energy situation is the maximum likelihood situation. It is intuitive to understand the relation between distance and energy. The longer the distance is, the more energy it takes to move from one point to another point.

## 4. Approach

Our segmentation algorithm computes manifold distance from user specified seed points to establish a spatial prior. The final distance measure is a combination of manifold distance and distance transformed from a probabilistic classifier for each category. We can use the distance results to segment the image by the rule of minimum distance. Or we can generate probabilities for each category and use maximum a-posterior rule to segment the image. The algorithm could be briefly described as

Algorithm 1: Manifold Distance Segmentation Algorithm

```
foreach object 1 to k, do
    set seed points for each object
end
foreach object 1 to k, point $x \in \Omega$ do
    compute manifold distance $d_{mk}(x)$
    compute probabilistic distance $d_{pk}(x)$
    combine $d_{mk}(x)$ and $d_{pk}(x)$ to $d_{lk}(x)$
end
foreach point $x \in \Omega$, do
    choose minimum distance of $d_{lk}(x)$, set k as the label
end
```

The final distance, or likelihood distance is simply add manifold distance and probabilistic distance with a scale factor.

$$d_{lk}(x) = d_{mk}(x) + \xi d_{pk}(x) \quad (17)$$

where $\xi$ is a constant value allowing users to weight the effects of these two terms. When $\xi = 1$, by Equation (12), $d_{mk}(x) + d_{pk}(x)$ is equivalent to

$$p(x|\omega_k) = p_{dk}(x)p_k(x) \quad (18)$$

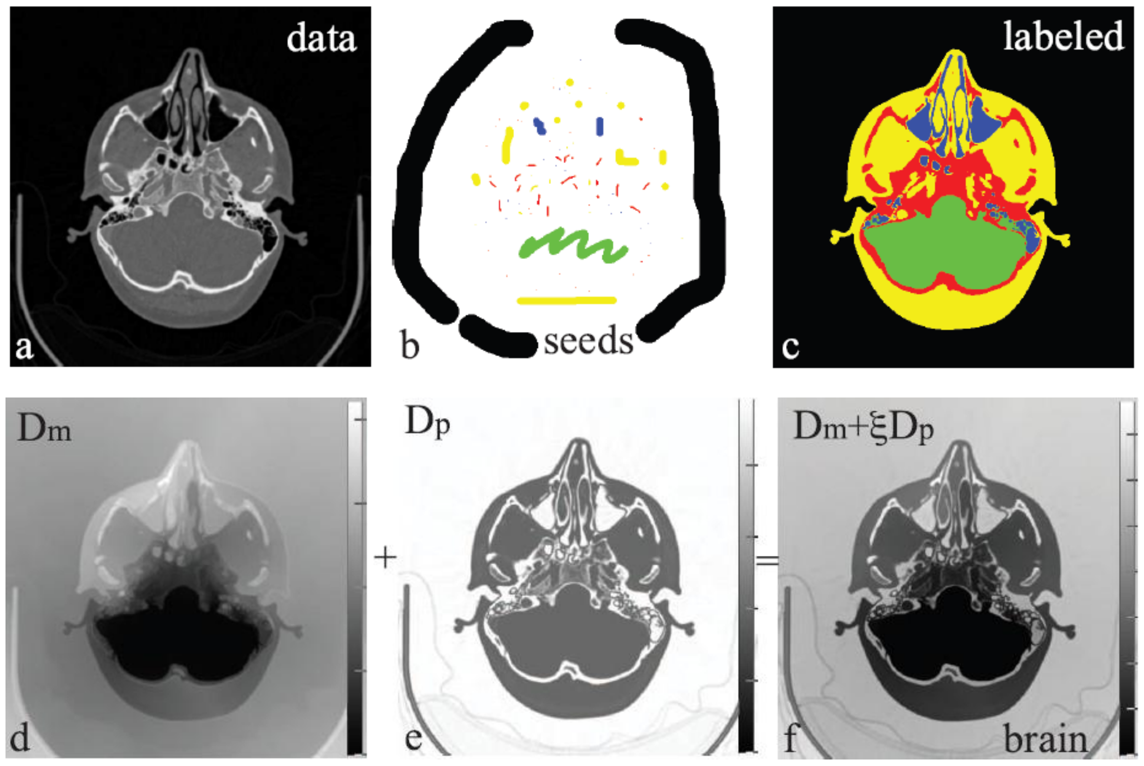
The last step, labeling image is to determine which object or category a point should belong to. If we consider from distance prospective, among k likelihood distance values, the object with the minimum distance would be considered as the label at this pixel. From probabilistic prospective, we can label images by maximum likelihood or we can get posterior probabilities for all categories from Bayes rule. The category with maximum probability is assigned as the label of that point.

There are multiple choices of $p_k(x)$, in our application users could use either normal distribution or Parzen window. For normal distribution, we calculate mean and covariance for each category from given seed points. The distance is calculated follow Mahalanobis Distance algorithm. For Parzen window classification we use normal distribution as the kernel, then the Parzen window is

$$p_k(x) = \frac{1}{Mw\sqrt{2\pi}} \sum_{i=1}^{M} e^{-\frac{\|I(x)-I(s_i)\|^2}{w^2}} \quad (19)$$

where M is the number of seed points for category k, $s_i$ is the location of seed and w is the width of the window. After translation to distance, we can get the probability distance mentioned above. The time complexity of normal distribution and Mahalanobis distance is $O(N)$. The time complexity of Parzen window is $O(NM)$. However, for low dimensional feature spaces, we can build a lookup table in $O(M)$ time and make the algorithm with time complexity of $O(N)$. For example, a gray level image has only 256 levels for its value, considering 256 as a constant number we can build this lookup table in $O(M)$ time.

Figure 4 shows results of different step in the whole process. (a) is the original image. (b) is the seed file, in which different color points are seeds for different category. (c) is the final segmentation result. (d)(e)(f) show manifold distance, probability distance and likelihood distance for brain respectively where the probability distance utilizes Parzen window and $\xi = 3.5$. (g)(h)(i) show the same terms of about for category soft. (j)(k)(l) are likelihood distance image of back, bone and sinus.



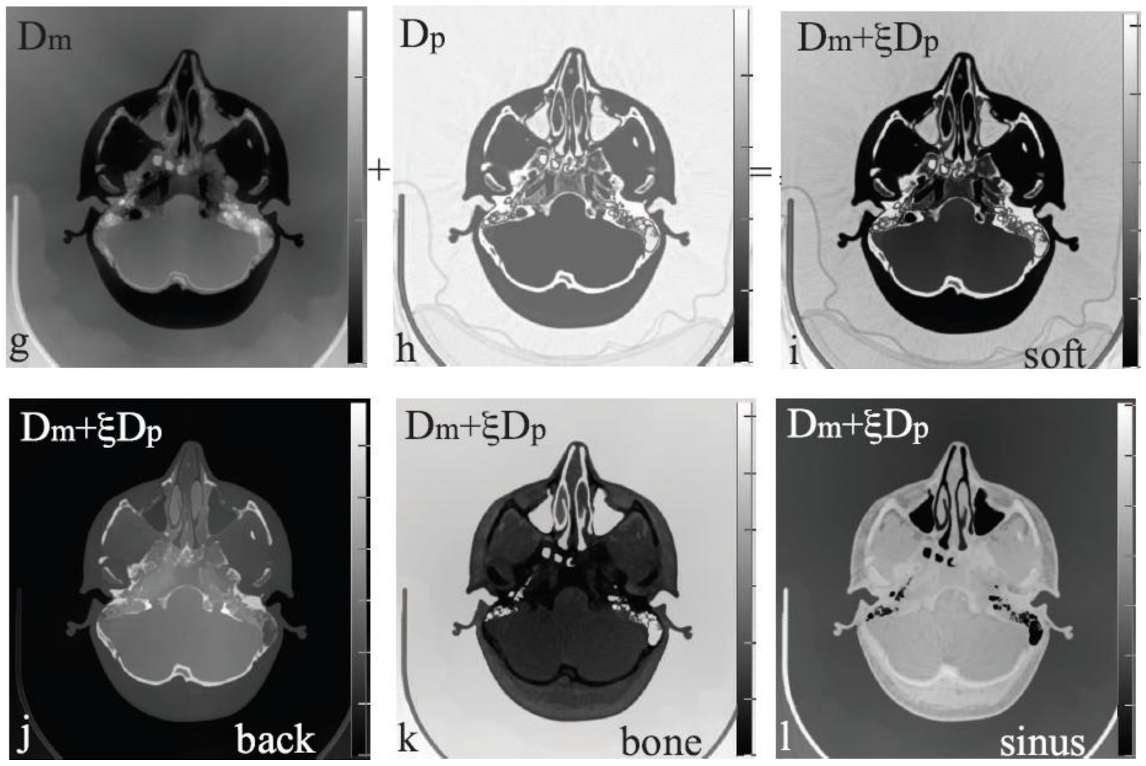


Figure 4. Elements of Manifold Distance Segmentation

(a) and (b) are input images. (a) is the original image to be segmented. (b) is the seed image. Seeds with different label are represented with different color. (c) is segmentation result based on shortest distance. (d) is manifold distance from brain. (e) is probability distance using Parzen Window from brain. (f) is combination of manifold distance and probability distance from brain. (g) is manifold distance from soft tissue. (h) is probability distance using Parzen Window from soft tissue. (i) is combination of manifold distance and probability distance from soft tissue. We can see (e) and (h) are very similar. (j) is the likelihood distance, the combination of manifold distance and probability distance, from back. (k) is likelihood distance from bone. (j) is likelihood distance from sinus.

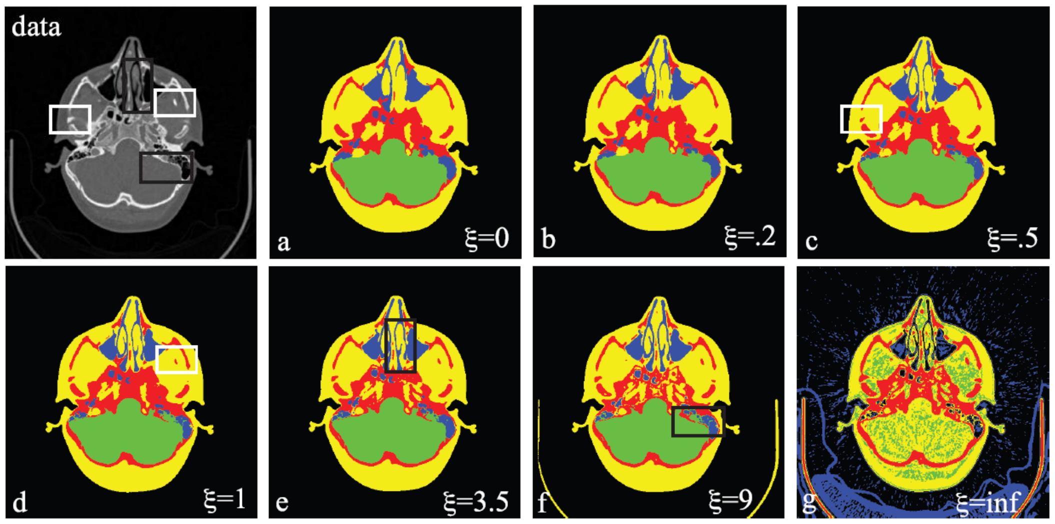


Figure 5. Combination of Segmentation and Classification

(a) is pure Voronoi tessellation of seed points based on manifold distance. (b)(c)(d)(e)(f) are combination with different $\xi$ value. (g) is pure classification result by Parzen Window classifier.

## 4.1 Implementation

Since our algorithm uses a combination of manifold distance and probability distance, we can consider this algorithm as a probabilistic segmentation or a classification with a spatial prior. We can control the effects from both sides by setting different $\xi$ value. For an extreme situation, if $\xi = 0$, that means the probabilistic term does not affect at all then the algorithm is a pure segmentation method. If $\xi$ goes to infinite, after normalization the manifold distance term will go to zero, which means in this situation the algorithm is a probabilistic classification method. Figure 5 shows labeled results with different $\xi$ value. Notice the different details in white blank for bone and black blank for sinus. There is no user specified seeds in these small pieces of area, but this algorithm can still label them out when the $\xi$ value is appropriately chosen. Users can choose different $\xi$

value to compare and get the best segmentation they want. It can also save users from the tedious work of setting seed points inside every disjoint region. Even this capacity is not what users want, it can also be eliminated in $\mathrm{O(N)}$ time by checking whether there is seed point inside.

When computing manifold distance, we use a modified Dijkstra's algorithm for shortest distance. The time complexity for Dijkstra's algorithm is $\mathrm{O(N\log N)}$ when using a priority queue to store current nodes. In practical situation, since we only need to store current nodes in the queue, the time complexity could be less than $\mathrm{O(N\log N)}$. In our experiment, the average number of current nodes is about 0.01N, which gives us time complexity of $\mathrm{O(N\log 0.01N)}$. That means for a $512 \times 512$ image, the time complexity is about $\mathrm{O(11N)}$ and for a $1024 \times 1024$ image the time complexity is about $\mathrm{O(13N)}$.

The modified Dijkstra's algorithm we use has a significant difference that is we need not to extract the minimum one from the current node list. This character allows us to use a ring buffer data structure instead of a priority queue. With this data structure, we can do insertion and extraction with $\mathrm{O(1)}$ time then the total time complexity is the times of inserting points into the list. However, within this algorithm we cannot constrain that every node only be inserted into the list only one time. So for a general graph the worst case time complexity of this modified Dijkstra's algorithm is $\mathrm{O(N^2)}$. We should think of this situation like that of Quick Sort, which also takes $\mathrm{O(N^2)}$ in worst case but normally works better than many other $O(N \log N)$ sorting algorithms. Empirically we measure this time complexity as $O(CN)$, where C is the average times of update for every pixel.

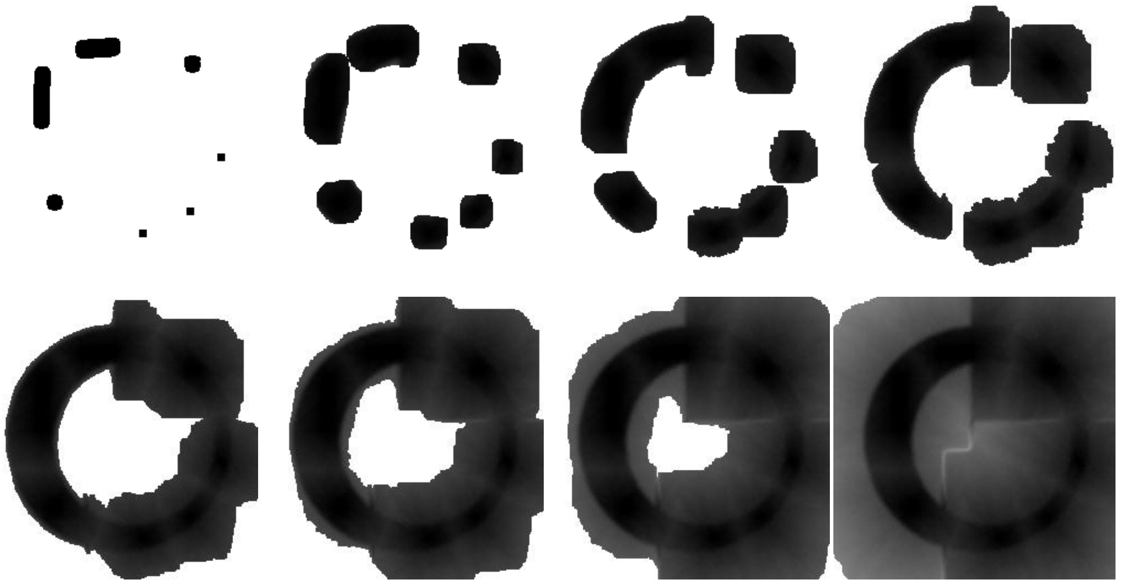


Figure 6. Shortest Distance Update

The sequence of images shows the procedure of computing shortest distance. It begins with 0 distance area of seeds then expands to the whole image gradually. The gray level indicates distance after affine transformation and quantization.

Algorithm 2: Modified Dijkstra's Algorithm

```
set all $d_{mk}(x) = \infty$
foreach $x \in$ Seeds, do
    $d_{mk}(x) = 0$
    foreach neighboring point y of x, do
        insert y
        $d_{mk}(y) = -\infty$
    end
end
foreach x in ring buffer, do
    $d_{mk}(x) = -d_{mk}(x)$
    foreach neighboring point y of x, do
        $d_y = d_{mk}(x) + \mathrm{weight}(x, y)$
        if $d_y < abs(d_{mk}(y))$, then
            if $d_{mk}(y) > 0$, then
                insert y
            end
            $d_{mk}(y) = -d_y$
        end
    end
end
```

The signed distance in the algorithm is just a practical way to indicate status of a node. There is no actual negative distance in this algorithm. We experiment images with different dimension and scale. The average value of C is 3.5 and it is relatively steady for all images. We also found the value of C does not increase with the scale of images. The reason we can get a linear time complexity is that our graph is a regular lattice with constrained edge weight range. The edge weight never range from 0 to infinite which could bring the worst case situation. In this situation, algorithm with linear time complexity has been introduced by Thorup [8]. The constant term could range for different images. We show examples with different constant term in Figure 7. (a) is a critical image for our algorithm which performs a constant term of 15 in this experiment. That means for this image with scale of $512 \times 512$, our modified algorithm even runs slower than Dijkstra's algorithm. (b) is also critical image and we expect a high constant term but this time C is only 3.9. (c) is generated from a uniform image. It shows a Voronoi graph with constant term almost equals to 1, which is the same as we expect. (d) is generated from a random image that has random value at every point. The constant term is about 4 and the segmentation result is similar to (c).

In our implementation, another work we add upon the algorithm described above is to sort the ring buffer at a certain number, s of insertion. This work can significantly reduce the constant term C because it could reduce the times of updating a point. When $s = 1$, this algorithm pops the minimum one from current node list. This is the same as

Dijkstra's algorithm and minimizes the times of total insertions. However, the sorting is also an expensive operation so we need to find an appropriate frequency of operating sorting. Figure 8 shows the relation between sorting frequency and run time for 3D images with different scale.

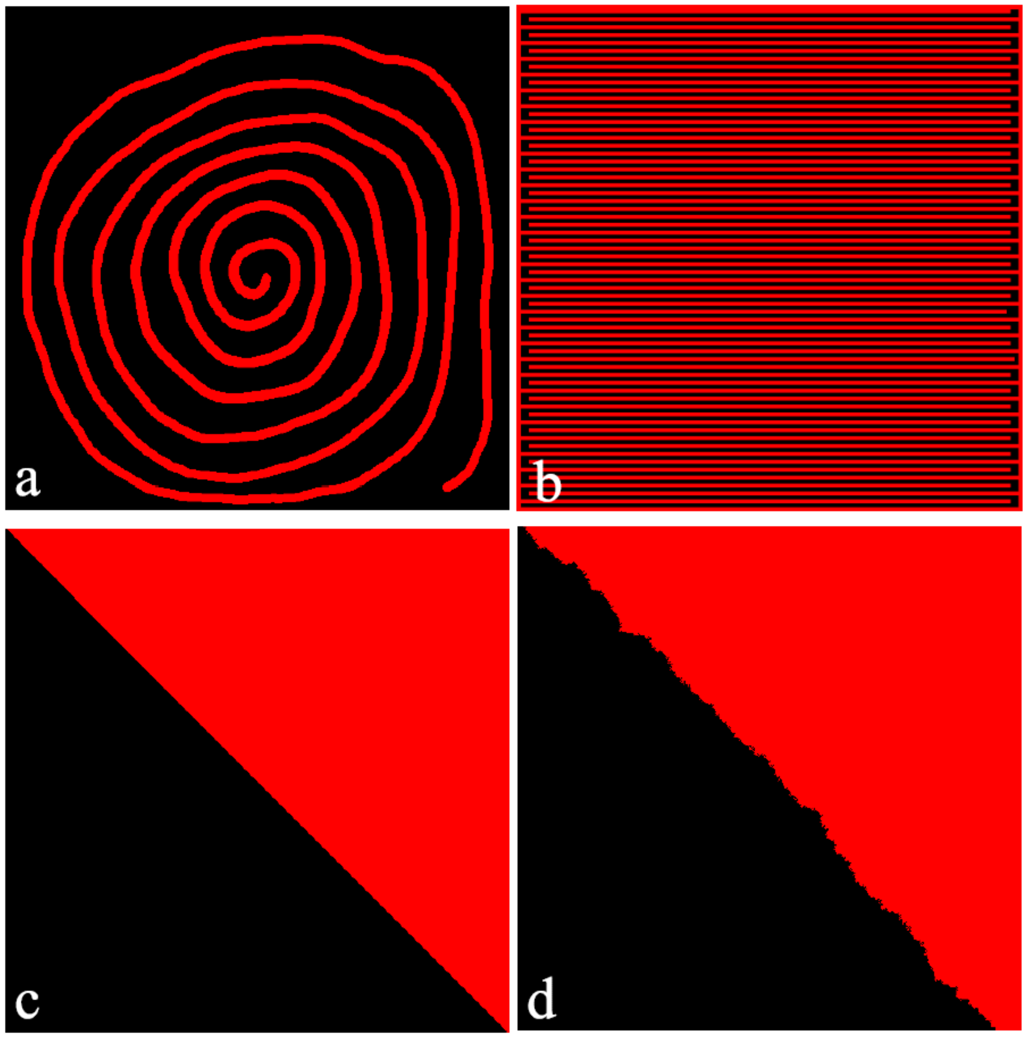


Figure 7. Critical Test for Modified Dijkstra's Algorithm

All images' sizes are $512 \times 512$. In each image, there are two segmented objects represented by black and red. (a) needs a constant factor C=15. (b) C=3.9. The results of (a) and (b) are highly accurate. (c) is generated from a pure white image and two seeds located at top-right and bottom-left corner. C=1. (d) uses the same seed file as (c) uses but the original image is a random image, in which every pixel has a random value. C=4.

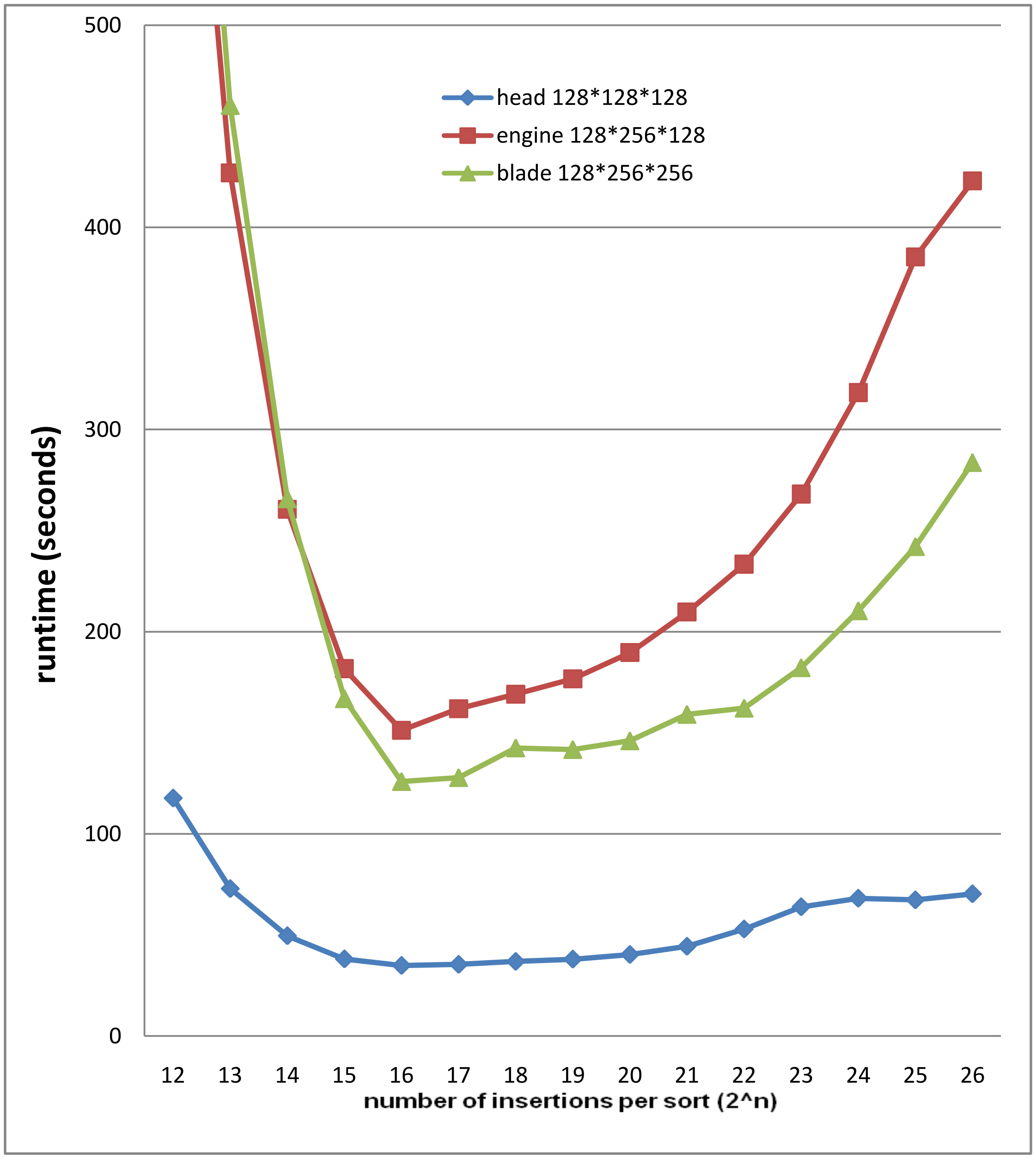


Figure 8. Runtime with Different Sorting Strategy

The horizontal axis is interval of insertion times between every two sorting. The number is exponent of 2. The vertical axis is runtime using second as unit.

# 5. Discuss

In this section a comprehensive discussion of Manifold Distance Segmentation algorithm will be presented. One key point of this algorithm, the rule of building image manifold is explored. Image noise as a common problem in image segmentation is analyzed. We also compare our algorithm with other popular image segmentation algorithms in time complexity and quality of results. In the end of this section we indicate current limitation and possible improvement in the future.

From a software engineering point of view, we evaluate this method by validation and verification process. For validation process, the aim is to determine whether the segmentation results reflect the location and boundary of objects. Some of the original images are geometry shapes with blur or noise generated by ourselves. For these images we only need to compare the segmentation results with the ground-truth images label by label. For those images without ground-truth standard, like physical simulation images, the test is based on human experiences. Both kinds of experiments indicate that this approach satisfies the specified requirement. In the verification process, we make experiment and analyze results to make sure that every step in the procedure satisfy the requirements and design specification. In the selection of feature space, we evaluate different choices with signal to noise ratio. When computing shortest distance, various critical situations are experimented and show expected results. In the combination with classification method, we study the sensitivity of regularization parameter and provide relative expressions. These step by step unit tests give this algorithm a successful dynamic verification.

## 5.1 Feature Space

We studied various locally derived terms to fill the feature space. The basic and original feature is only the data value of an image. For instance, a gray level image has 1D feature space, gray value. A color image has 3D feature space of red, green and blue color value. We can expand feature by various measures like partial derivatives, gradient magnitude and mean value of neighbors. If we add these three terms on, a gray level 2D image would have a 5D feature space. It is meaningless to expand the feature space as large as possible. Figure 9 shows segmentation results using different feature spaces. In all these experiments we set $\xi = 0$, which means only manifold distance is taken into account because the feature space is not related with probability distance and we do not want that affect now. All feature spaces take data value as a necessary term and different other terms as option. To give a quantitive measure of segmentation result instead of just human eyes, we build an expected image $I'$ to compare with original image I. The expected image is generated by

$$I'(x) = \sum_{k=1}^{K} p(\omega_k|x)\mu_k \quad (20)$$

where $p(\omega_k|x)$ is the posterior probability. The expected value for one category k,

$$\mu_k = \sum_{i=1}^{N} I(x_i)p(x|\omega_k) \quad (21)$$

where $p(x|\omega_k)$ is the likelihood. Finally we check the difference between original image and expect image by Signal to Noise Ratio

$$\mathrm{SNR} = 20\log_{10}\left(\frac{\|\mu_I\|}{\sqrt{\sum_{i=1}^{N}\|I(x_i)-I'(x_i)\|^2}}\right) \quad (22)$$

where $\mu_I$ is the average value of original image, the signal. The sum of difference between two images is noise. Figure 10(a-e) shows the posterior probability for each category of a brain image. (f) is we called label-expected image, for which we use the mean value of each object to fill all area labeled as that object. We can consider it as the signal part. (h) is the expected image. Different from (f), (h) is calculated from Equation (20), not replaces with mean value. (i) shows the difference of them, which could be considered as noise part. The ideal scenario of this image should be pure dark. We see in this image the bone area has relative high value. This is because bone has a wide value range then the variance of this object is high. The signal to noise ratio is 8.7 dB with $\xi = 3.5$.

Compared with figure 9, we find that under this standard the natural feature space, which contains data values and positions is the best solution in all feature spaces we experiment. This group of results is compatible with human experience. In figure 9, we show labeled segmentation result utilizing different feature space. From the signal to noise ratio shown we can see all these features do not help to improve results. The gradient magnitude is calculated from the two neighboring points for each axis. The mean, median and variance are calculated from a 2D local $5 \times 5$ lattice with center at the point. Although we show that many common feature spaces are not successful and there are also some unsuccessful one we do not post on this thesis, we cannot deny that there are possible feature spaces would improve our current algorithm. In the future we can try other features to optimize our algorithm.

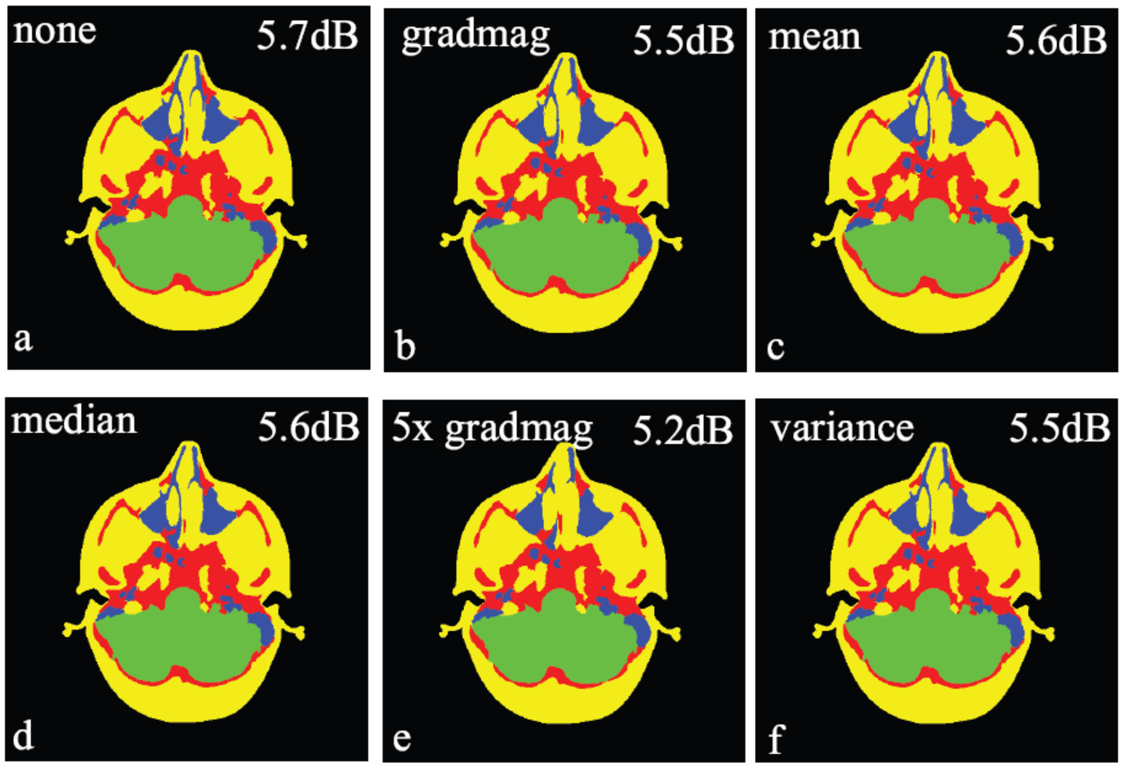


Figure 9. Exploration of Feature Spaces

(a) uses only original image data value. (b) uses original data value and gradient magnitude. (e) uses original data value and 5 times of gradient magnitude. (c)(d)(f) use mean, median and variance computed from $5 \times 5$ neighboring points around the point. From signal to noise ratio shown in top-right corner of each image, we can see none of the additional term improves the performance.

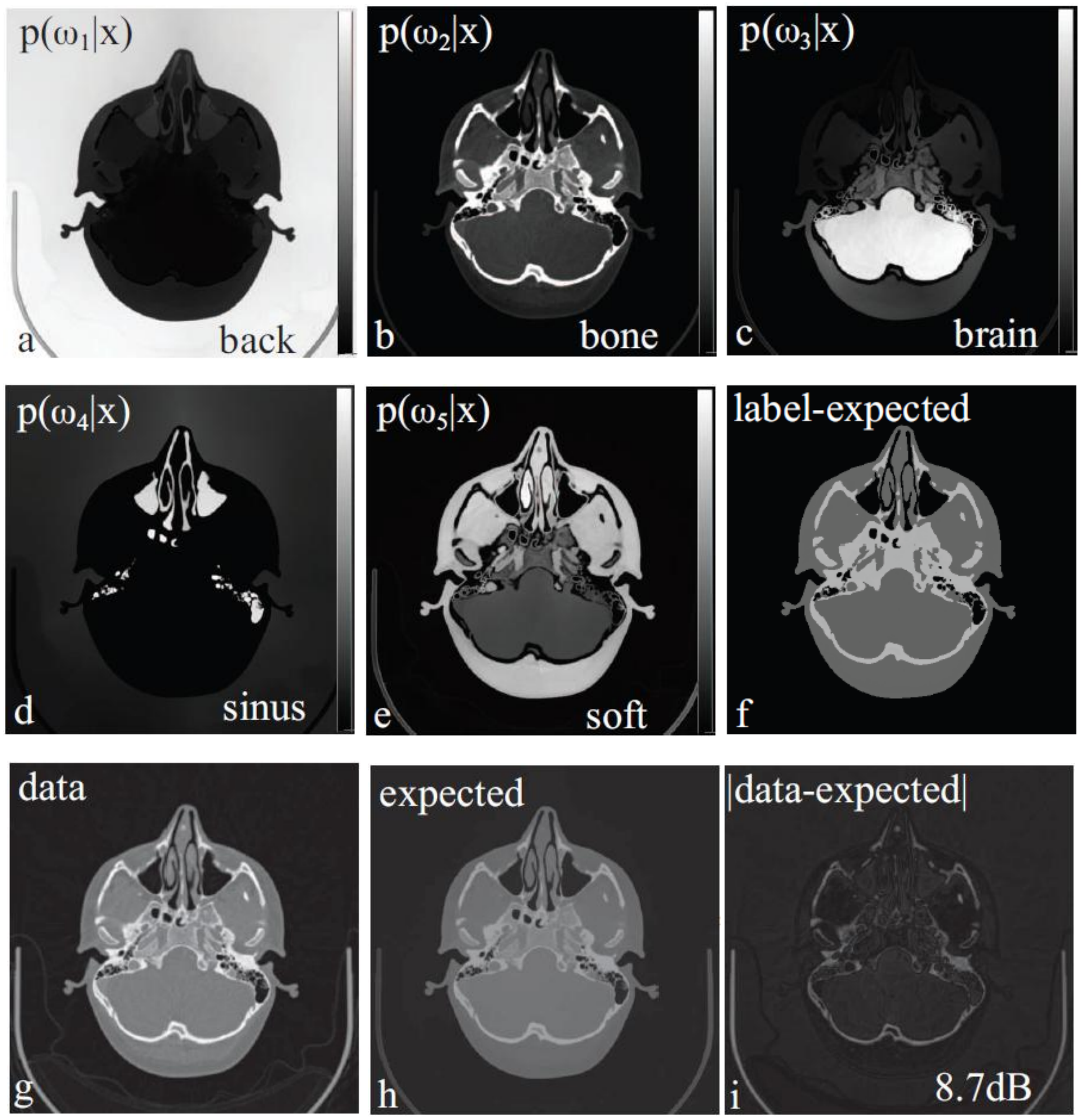


Figure 10. Posterior Probabilities and Expected Images

(a)(b)(c)(d)(e) are posterior probabilities for each label shown in bottom-right corner. (f) is called label-expected image. (g) is original image. (h) is expected image. (i)=|(g)-(h)|.

## 5.2 Noise Sensitivity

Most segmentation algorithms prefer data sets with distinct value of each object and sharp boundary between objects. However images in practical situation like medical images and physical simulations could be mixed with undesired noise. These irrelevant and meaningless data could affect the final segmentation results in different extent. A robust segmentation algorithm should be able to perform stably with noise.

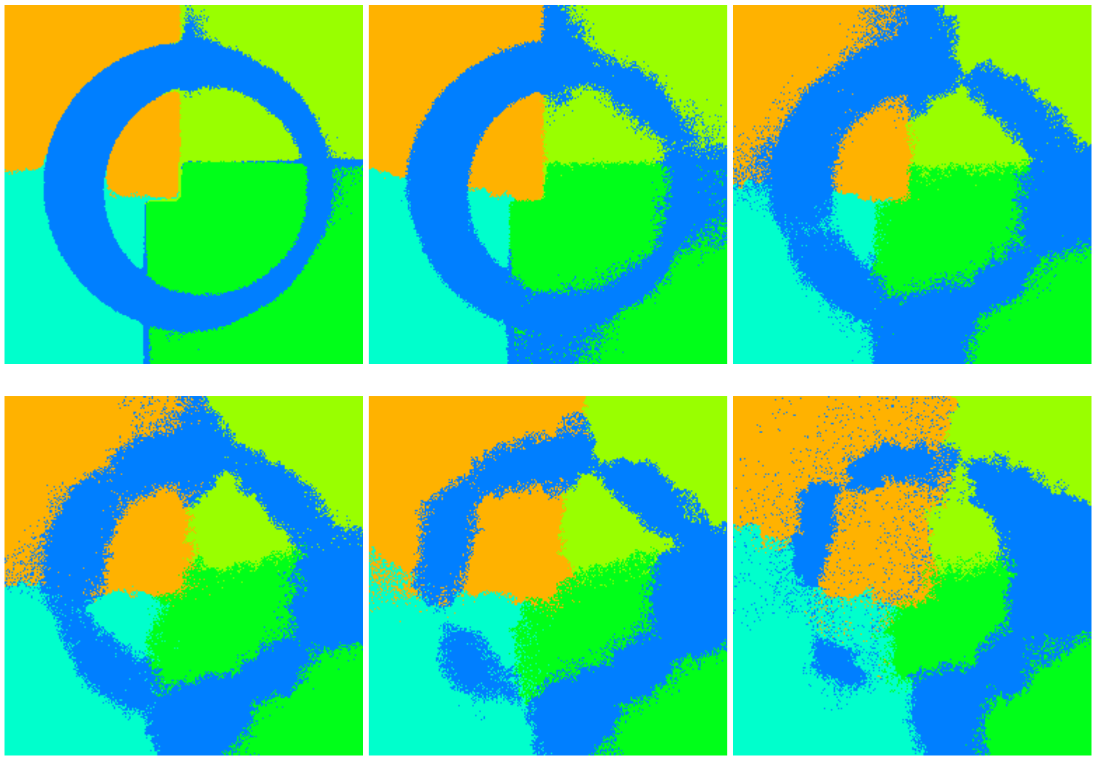


Figure 11. Segmentation from Images with Noise

In the first row are results from images with noise ratio of 0, 0.08 and 0.2, from left to right. The second row are results from iamges with noise ratio of 0.3, 0.5, 1.0.

We can see this approach can segment a rounded circle when the noise ratio is 0.3. Even when the noise ratio is 1.0, it can still capture the rough shape of the four rectangle objects. Experiments with slight noise images for various segmentation algorithms are shown in next section.

## 5.3 Comparisons

In this section we compare our algorithm with other common and widely used segmentation algorithms we briefly mentioned before. They are supervised algorithms Level Sets, Random Walks and unsupervised algorithms Watershed, Mean Shift. Supervised algorithms need users input seeds to define number of objects and characteristics of each object. Usually they are more robust in segmenting disjoint areas if seeds are appropriated set. Unsupervised algorithm has no given information. Therefore they will possibly segment one category into multiple pieces. However, unsupervised algorithms may behave better in running time than supervised algorithms because they do not need to iteratively compute for each object like supervised ones.

### 5.3.1 Complexity

As discussed above, our algorithm could perform a $O(N)$ time complexity. For Level Sets algorithm, the time complexity ranges largely depending on the selection of implicit function. Fully evolved implicit functions could make the method perform $O(N^2)$ time complexity. If the segmented region is small and only the region near the level-set of the implicit function is updated, the algorithm could run in $O(N)$ time. Random Walks algorithm requires a $O(N^2)$ time complexity in a naively computing way. However, if this algorithm uses iterative conjugate gradient to compute, it only requires $O(MN)$ time complexity where M is the number of unique eigenvectors of the system. M has an upper boundary of N in which case the time complexity is still $O(N^2)$ but normally it is less than N. Grady et al. also introduced an approximate solution to solve Random Walks. In this method a reduced set of pre-computed eigenvectors replaces the set of all unique eigenvectors [9]. For example, it needs 40 eigenvectors to segment a $512 \times 512$ image. If we ignore the procedure of calculating the eigenvectors, this algorithm runs in $O(N)$ time, although the constant factor is large, say, 40 in this example. Mean Shift segmentation algorithm's time complexity is affected by multiple factors. Say $P$ is the number of pixels in spatial kernel, $V$ is the size of value kernel, $I$ is the number of iteration the algorithm needs to coverage. The time complexity of this algorithm is $O(PVIN)$, which is hard to compare with others. Watershed segmentation algorithm requires $O(LN)$ time complexity, where $L$ is the size of water-table recording altitude levels used. There is optimization on this algorithm which could reduce time complexity to $O(N)$ [10] with a large constant term. Table 1 shows runtime of three algorithms for three 3D volume images with different scales. From the table we can see our manifold distance segmentation algorithm runs 5-10 times faster than other two.

| | Head | Blade | Mhc |
|---|---|---|---|
| Level Sets | 370s | 680s | 4218s |
| Watershed | 595s | 1790s | 7200s+ |
| Manifold Distance | 34s | 127s | 454s |

Table 1: Runtime comparison

The first line shows three image files. Head is a $128 \times 128 \times 128$ image with 5 object labels. Blade is a $128 \times 256 \times 256$ image with 3 object labels. Mhc is a $256 \times 256 \times 256$ image with 5 object labels. All images are gray level images. All programs are run under same hardware and software environment with single thread. Figure 12 shows results of this group of experiments.

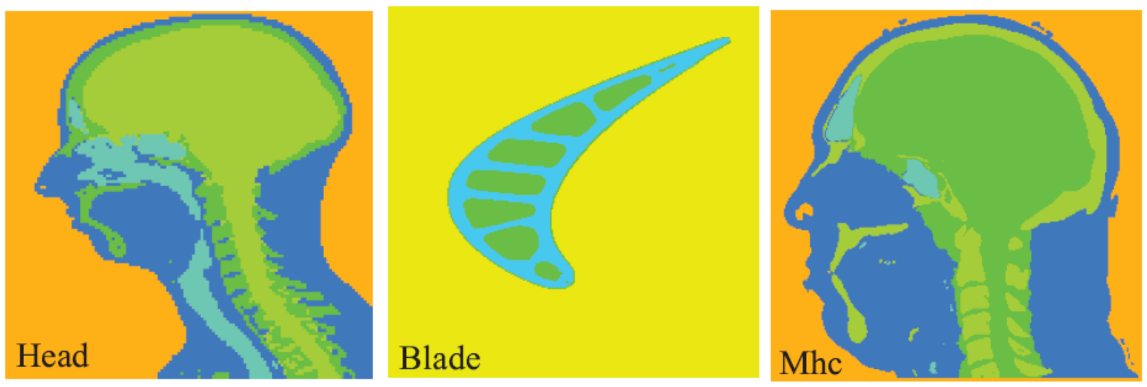


Figure 12. 3D Segmentation Results

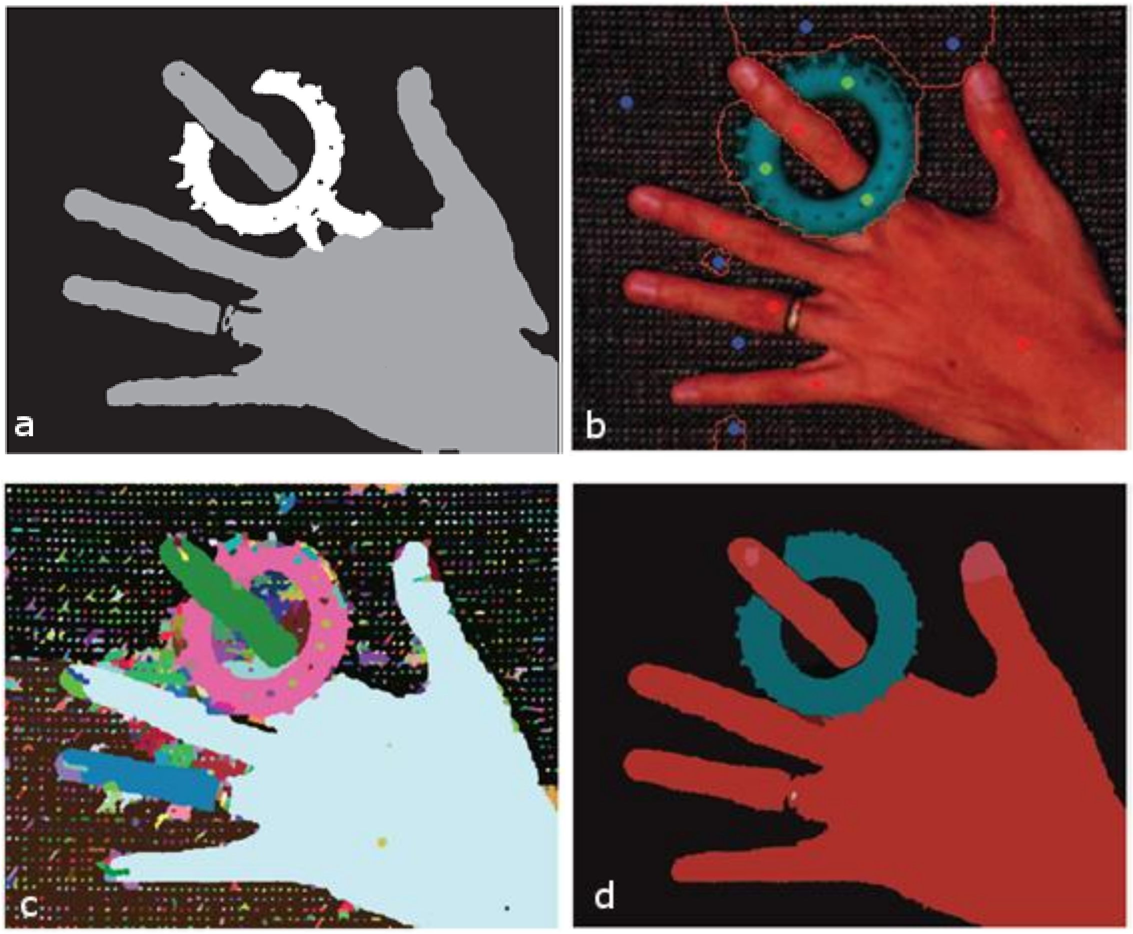
Images shown here are slices of 3D result images from Manifold Distance Segmentation.

### 5.3.2 Results

All the algorithms we experimented to compare with our algorithm are implemented posted from reliable sources. Level Sets and Watershed are applications from Insight Toolkit [11]. Random Walks application is gained from Matlab code posted on Leo Grady's website [12]. We use "Edge Detection and Image Segmentation System" (EDISON) [13] as Mean Shift application. For all supervised algorithms, Level Sets, Random Walks and ours, we used exactly same seed information. Although the seed image files could be different because these applications require different input files, the positions and labels for all seeds are the same. All experiments are done under the same hardware and software environment. We use a desktop with a 3.0G Intel core2 quad CPU and 3G RAM. IDEs are Microsoft Visual Studio 2005 or Matlab R2007a under Window

XP OS. Random Walks application uses default Matlab sparse system solver. We do not put Random Walks and Mean Shift results in Table 1 is because in the applications we get, they cannot operate on 3D volume images.

In Figure 13, the finger-ring image is a test image for Mean Shift in the work of Comaniciu *et al.* [6] This image is a natural picture with texture and intricate shapes so it is difficult for image segmentation. Among all other results listed, the Manifold Distance Segmentation is the best one and the only one could compare with the result of Mean Shift. But the runtime of Manifold Distance Segmentation is much less than that of Mean Shift. For Mean Shift algorithm, parameters $h_s$ and $h_r$ could significantly affect the runtime. We choose these parameters exactly same as Comaniciu's choice. For Mean Shift, if parameters are appropriately chosen, runtime could be largely reduced with only little quality loss. In this experiment, Random Walks could hardly segment any category. Figure 14 shows image used as Random Walks' test image in Grady's work [12]. This time Random Walks precisely segments the area containing the seed points, while Manifold Distance Segment could not get an even close result. However, if we replace the old seed information with our new 4 label seeds, Manifold Distance Segmentation could clearly label all four categories. Figure 15 shows a bunch of results based on a group of ring images provided by Kniss *et al.* [2]. Each row of results is from the same algorithm and each column is from the same source image. The four source images have different degree of blur that could represent a wide range of images. We can see all algorithms show artifacts at various ways. The bottom line is expected images of Manifold Distance Segmentation with means based on the ground-truth colors. We can see the blurring of expected images is consistent with their corresponding source images. The Signal to Noise Ratio of them is at least 17 dB.



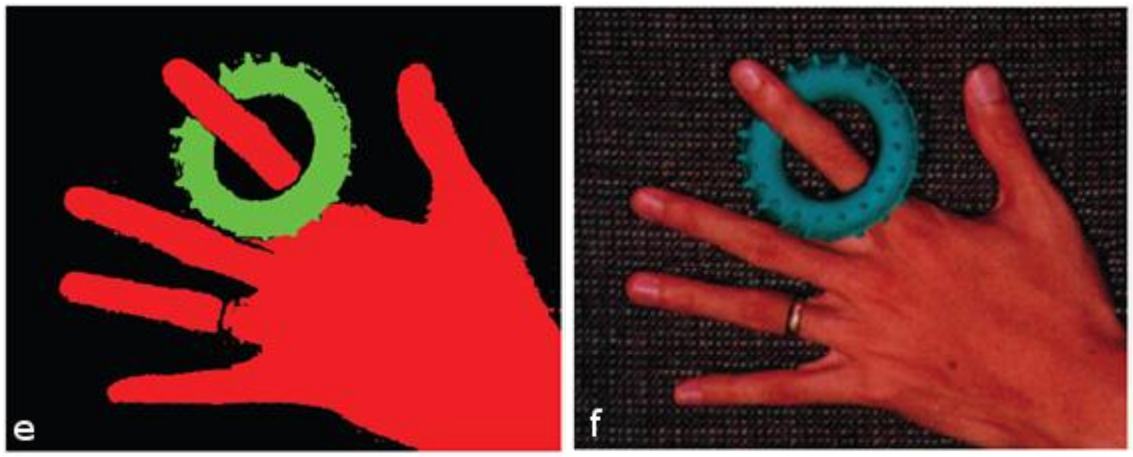
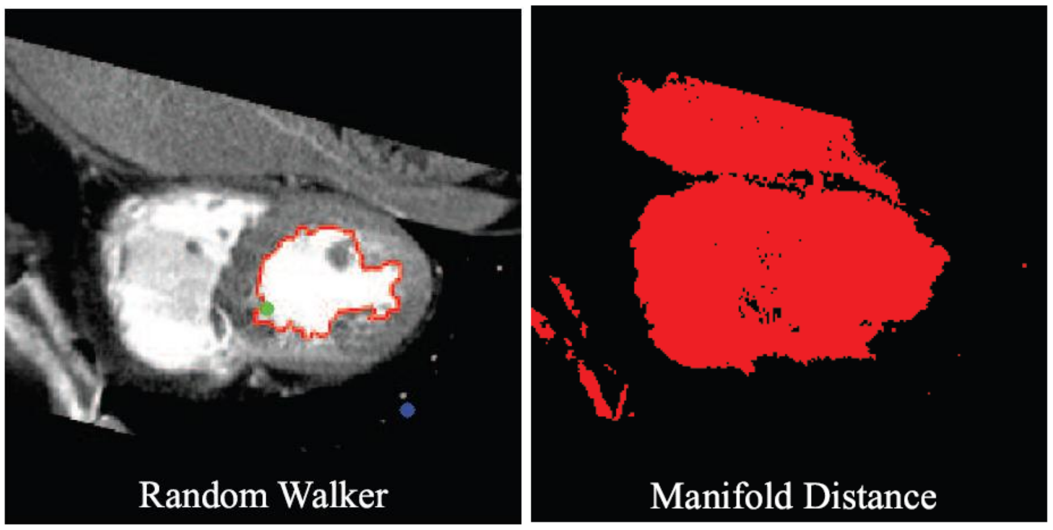


Figure 13. Segmentation Results of Finger-ring from Various Algorithms

(f) is the original image. The Threshhold Level Set (a) uses upper-lower bounds [54,89] for the hand and [60,150] for the ring. Distance is set as 5 and curvature is set as 10. Runtime is 25 seconds. Random Walks (b) algorithm uses $\beta = 25$. Runtime is 5 seconds. Watershed (c) uses conductance=3, iteration=10, threshold=0.0001, level=0.06 and principal component=1. Runtime is 40 seconds. Meanshift (d) uses $h_s$=16, $h_r$=19 and M=40. Runtime is 40 seconds. Manifold Distance uses $\xi = 3.5$. Runtime is 23 seconds. In this experiment, Mean Shift perfectly defeats all other algorithms. Although Random Walks runs fast, its result is far from correct. Manifold Distance (e) is the only one could compare with Mean Shift.



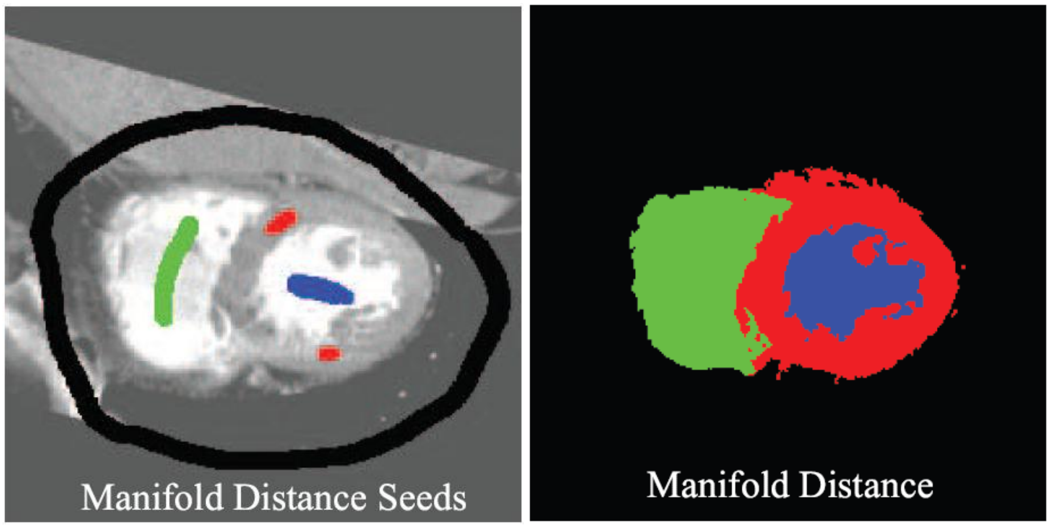
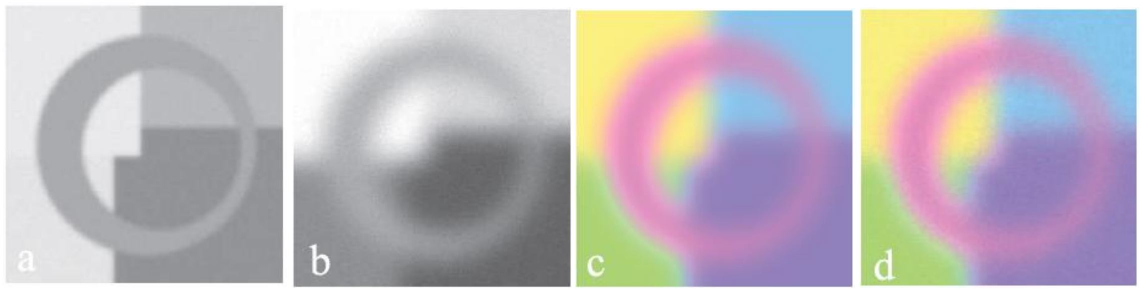


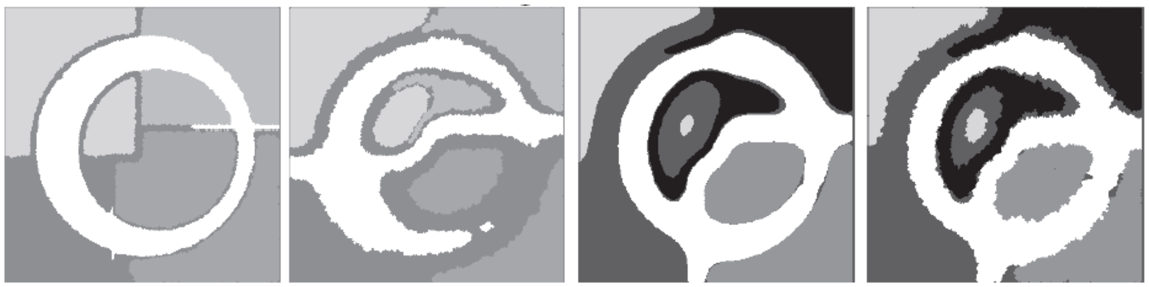
Figure 14. Comparison between Random Walks and Manifold Distance

The first row shows results of Random Walks and Manifold Distance with seed points provided by the demo experiment of Random Walks. Random Walks works well while Manifold Distance does very poor. However, when we define the image as more objects and corresponding characteristics, Manifold could also works very well (bottom-right image). In this experiment, Random Walks uses $\beta = 25$ and Manifold Distance uses $\xi = 1.0$.

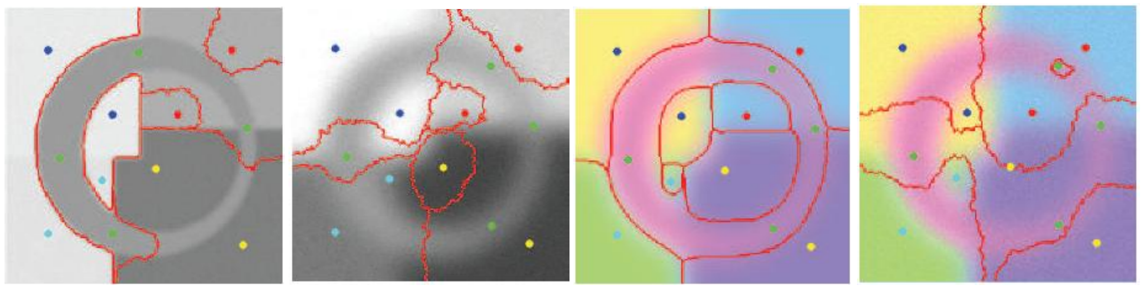
Original Image:



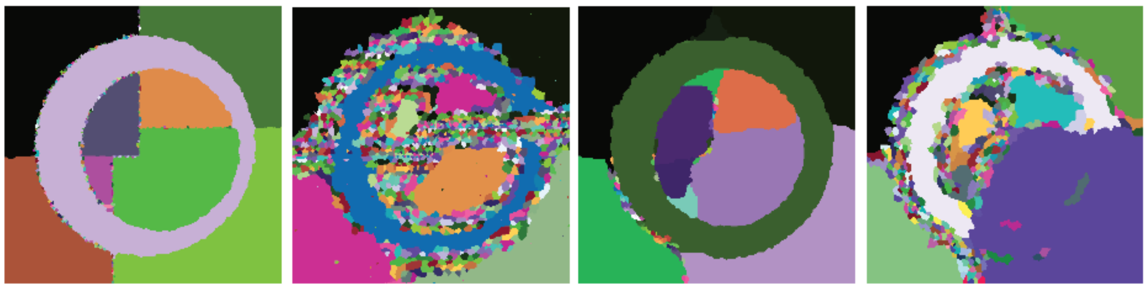
Threshold Level Set:



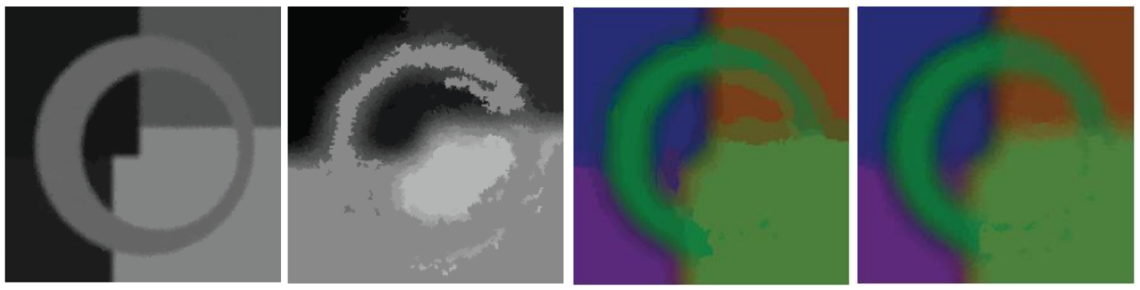
Random Walks:



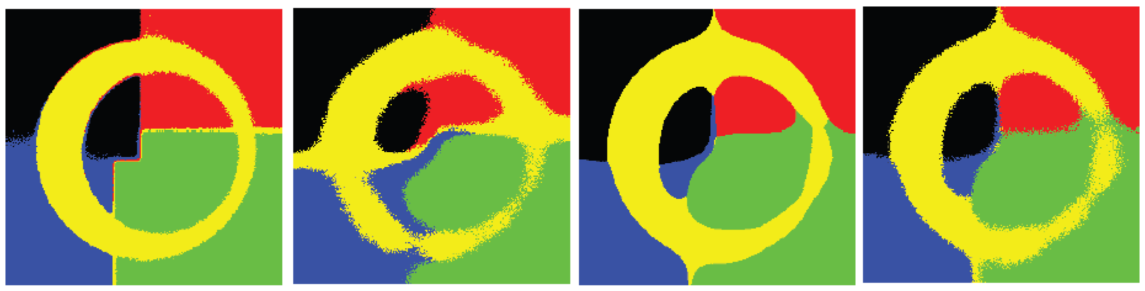
Watershed:



Mean Shift:



Manifold Distance:



Manifold Distance Expected Image:

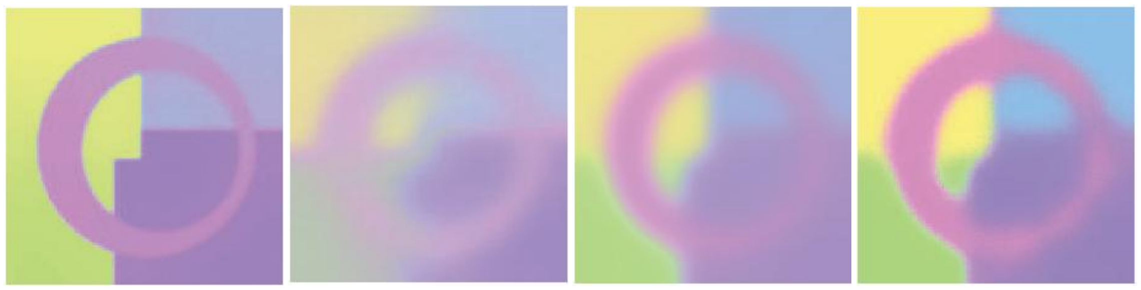


Figure 15. Comparison of Different Algorithms with a Serial of Ring Images

The first row shows original images. (a) is low-blur gray level image without noise. (b) is high-blur gray level image with noise. (c) is high-blur color image without noise. (d) is high-blur color image with noise. The second to the sixth row are segmention result from Level Set, Random Walks, Watershed, Mean Shift, and Manifold Distance respectively. The seventh row lists expected images. The result is generated from the original image in the same column. All supervised segmentation algorithms share the same seeds which are shown as color dots in Random Walks results. In this experiment, Manifold Distance Segmentation performs the best especially for the situations with noise.

## 5.4 Limitations

We find Manifold Distance Segmentation performs better on images with stable color and clear boundary for each category. On the other hand, it meets difficulty when handling images with large quantity of noise or texture. This is true for almost all image segmentation algorithms. Texture segmentation is possibly a solution to solve this problem. In that case we do not have a constant feature space but an approximate one in which the difference of pixel value is texture similarity, not based on raw data values.

### 5.4.1 Boundary Leak

When we check result images carefully, we can see in the boundary area between two categories, it is not rare to see that a third party category occupies a slim line. We think the reason is when we calculating distance, in the boundary area, where distances change quickly, it is possible that the distance is between these two neighboring categories but similar to a third party category. In this case, it is possible the in the final result the boundary is labeled as that third-party category. We study and try to fix this problem by adding the weight with another term that is sensitive to the direction of gradient. We hoped this will stop a third-party category invading into the boundary area because from that direction these will be a term added on the weight then defend the invasion. This term is called resistance. In order to conveniently control the effect of this term, a parameter $\zeta$ is introduced. The new weight of edge between two nodes i and j could be expressed as:

$$w_{i,j}{}' = w_{i,j} + \zeta \cdot r_{i,j} \quad (23)$$

where $w_{i,j}$ is the original weight, $\zeta$ is a numerical parameter and $r_{i,j}$ is the resistance term.

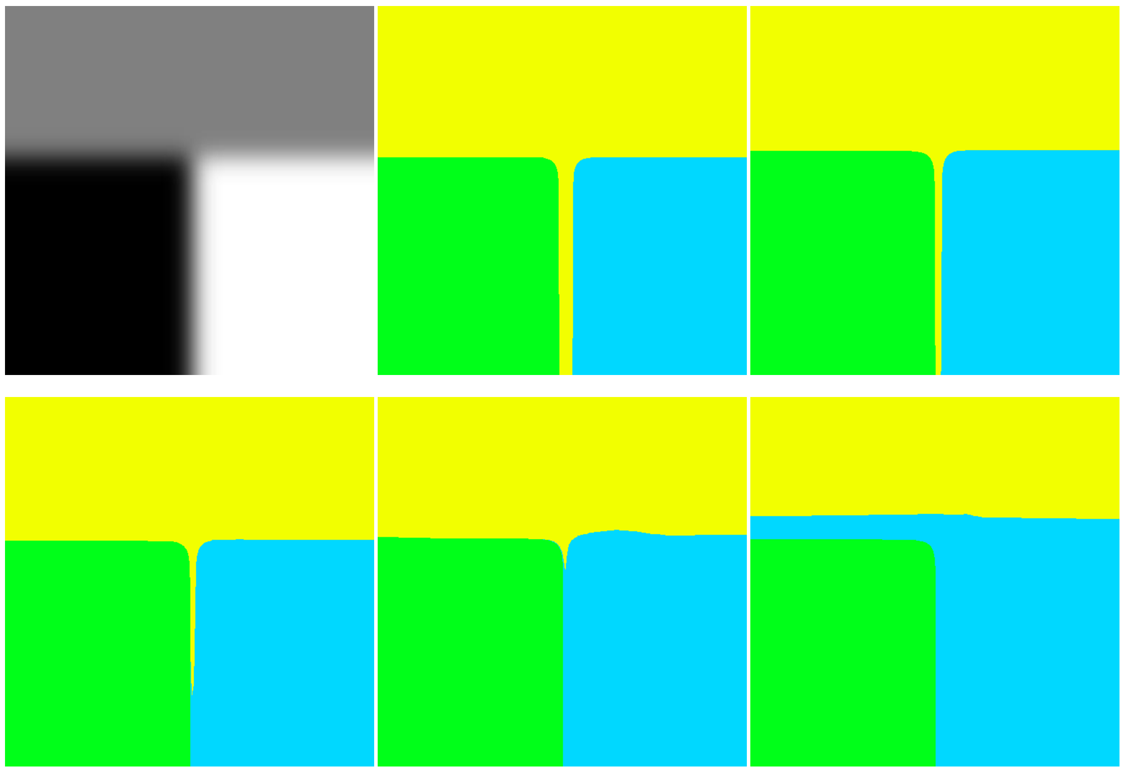


Figure 16. Boundary Leak Resistance

The top left iamge is the source image with blured boundaries. The rest images are segmentation results with different $\zeta$ value. It varies from 0, 0.3, 0.6, 0.7 and 0.9 from top to down, left to right.

The results show although we indeed stop invasion as we expected, the new term also brings new problems because it make distances cross boundary relatively shorter.

### 5.4.2 Reliance on User Input

As a supervised algorithm, users are required to provide specified seed information for every input image. Even with a well designed GUI tool, this work is still tedious and easy to fault. Like the Mhc image in Figure 12, the isolated blue points outside head is caused by inaccurate seeds. Another situation is the segmentation results from same image could be different in details because of different user input. Therefore, in order to reduce works for user and keep stable performance, this algorithm should be improved in reliance on user input. Users will be required to provide quite limited number of seeds or just number of labels. An ideal plan is to transform this algorithm into an unsupervised segmentation algorithm.

Without user input information, the approach needs to define characteristics of labels by itself. At first we can build a histogram of data value from the original image. Since pixels belonging to the same object share similar data value, a small continuous range with large number of pixels can be considered as one object. These ranges with small number of pixels are considered as boundary areas. In these narrow areas data values vary in a large scope. One object range should be clamped be two boundary ranges. In this meaning, seeds should be picked from object ranges instead of boundary ranges. Assume pixels distributed normally in each object range, seeds should be chosen near the mean area. For a range in histogram, the peak area, or the area with minimum gradient magnitude should be selected as the mean area of normal distribution.

In this way it seems like this algorithm goes back to a classification method since it does nothing with distinguishing disjoint objects with similar data value. To solve this problem, a combination of special information and data value could substitute for only data value. But this also could bring new problems, say mapping points in different object to a same label after combination. Besides, one object can also consist of multiple disjoint areas. Further progress solving this problem is mandatory to changing this current supervised segmentation algorithm into an unsupervised one.

## 6. Conclusion

Manifold Distance Segmentation is a high efficiency and quality image segmentation algorithm. It preserves uncertainty in the whole process. Its time and space complexity are both $O(N)$ and with a relatively small constant term. This algorithm could be affected in runtime and result by parameter setting. It is a combination of distance segmentation and probabilistic classification method. It is suitable in interactive visualization applications. In the feature we could study more different feature spaces, less user input and resistance to boundary leaks.

## References

[1] M. Levoy (1989). Display of Surfaces from Volume Data. PhD thesis, University of North Carolina at Chapel Hill.

[2] Joe M. Kniss, Robert van Uitert, Abraham Stephens, Guo-Shi Li, Tolga Tasdizen, and Charles Hansen (2005). Statistically Quantitative Volume Visualization. Visualization Conference, IEEE, 0:37.

[3] Stanley Osher and James A. Sethian (1988). Fronts propagating with curvature dependent speed: algorithms based on Hamilton-Jacobi formulations. Journal of Computational Physics, 79:12–49.

[4] Leo Grady (2005). Multilabel random walker image segmentation using prior models. CVPR 2005.

[5] S. Beucher and C. Lantuejoul (1979). Use of watersheds in contour detection. International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation, Rennes, France., September 1979.

[6] D. Comaniciu and P. Meer (2002). Mean shift: a robust approach toward feature space analysis. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 24(5):603–619.

[7] K. Fukunaga and L.D. Hostetler (1975). The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Trans. Information Theory, 21:32–40.

[8] Mikkel Thorup (1999). Undirected single-source shortest paths with positive integer weights in linear time. J. ACM, 46(3):362–394.

[9] Leo Grady and Ali Kemal Sinop (2008). Fast approximate Random Walker segmentation using eigenvector precomputation. CVPR, 2008.

[10] A. Bieniek and A. Moga (2000). An efficient watershed algorithm based on connected components. Pattern Recognition, 33(6):907–916.

[11] L. Ibanez, W. Schroeder, L. Ng, and J. Cates (2003). The ITK Software Guide: The Insight Segmentation and Registration Toolkit. Kitware, Inc.

[12] Leo Grady (March 2009) . Walker algorithm and script. http://cns-web.bu.edu/ lgrady/.

[13] Bogdan Georgescu and Chris M. Christoudias (March 2009). Edge detection and image segmentation system. http://www.caip.rutgers.edu/riul/research/code.html.