AN ABSTRACT OF THE THESIS OF

Avneet Sandhu for the degree of Master of Science in Computer Science

presented on September 21, 2006.

Title: Volume-Volume Matching

Abstract approved: _____

Dr. Mike Bailey

Oftentimes in visualization, the goal of using volume datasets is not just to visualize them but also to analyze and compare them. In order to compare the two volumes, we cannot take all the voxels into consideration. The size of a typical volume data set is quite large (maybe a billion or more voxels), thus it is not feasible to compare all these voxels to all in the other volume set. This project uses optimization methods to find the best orientation for aligning a second volume data set with a first. Using these methods, we will be able to determine how well one medical data volume matches against a "normally-developed" volume of the same type.

Volume-Volume Matching

by

Avneet Sandhu

A Thesis

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented September 21, 2006
Commencement June 2007

Master of Science thesis of Avneet Sandhu presented on September 21, 2006

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Avneet Sandhu, Author

Master of Science thesis of Avneet Sandhu presented on September 21, 2006

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Avneet Sandhu, Author

ACKNOWLEDGMENT

These lines of text are an expression of gratitude to all those who helped to make this thesis possible and for their never faltering encouragement and support.

I would like to express the highest gratitude to my parents who actually brought me to computer science many years ago and supported me throughout the duration of my studies. I would also want to specially thank my uncle who motivated me to join MS program at OSU and provided me with every possible comfort over the span of these 2 years.

I would like to thank my Major Advisor, Dr. Mike Bailey for offering me an opportunity to work with him and for his patient supervision, guidance, constant encouragement, valuable criticism and scientific discussions throughout the MS program. His visionary work in the field of Scientific Visualization gave me an opportunity to work on a very interesting project, volume-volume matching. Additionally, I want to thank the members of my committee, Dr. Eric Mortensen, Dr. Ron Metoyer and Dr. Harry Yeh for their guidance.

I would also like to thank the Graduate School and Dept. of EECS, who gave me an opportunity to pursue my MS degree in Computer Science at OSU. I also am indebted to all my professors under whose supervision I gained a lot of knowledge.

I also owe a debt of gratitude to my colleagues of Scientific Visualization Group for their valuable suggestions and friendly welcome that helped on the development of this thesis. I owe a special thanks to my friends, Robin Hess and Vasumathi Lakshmanan who have been very supportive and helpful throughout my thesis.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

Table Page

Volume-Volume Matching

1. INTRODUCTION AND OVERVIEW

A ***volume data set*** is a $3D$ entity that is represented as a $3D$ cloud of points. It can be uniform or non-uniform, rectilinear or non-rectilinear and can have any number of scalar data values attached to each voxel. It is typically a set S of samples (x, y, z, v), representing the value v of some property of the data, at a $3D$ location (x, y, z). If the value is simply a 0 or a 1, with a value of 0 indicating background and a value of 1 indicating the object, then the data is referred to as binary data. The data may instead be *multivalued*, with the value representing some measurable property of the data, including, for example, color, density, intensity, heat or pressure. ***Voxel*** is a smallest distinguishable volume element, representing a value in a $3D$ space. Voxels are frequently used in the visualisation and analysis of medical and scientific data. Some of the sources of volumetric data are:

- *Medicine* - imaging methods such as CT, MRI, ultrasound etc.
- *Science and Engineering* - simulations (flow, stress, heat), observations (wind, weather, pressure, satellite) and design (CAD/CAM).
- *Geology* - explorations (seismology, oil, precious metals), map-masking (satellite terrain mapping).
- *Entertainment* - games, special effects and weather forecasts.

This volume data could be used for *rendering, interaction, analysis, matching* and *data mining.*

This project, called "***volume-volume matching***", uses *optimization methods* to find the best orientation for aligning a second volume data set with the first one. Using these methods, we have been able to determine how well one volume matches against a "normally developed" volume of the same type.

It is not a straightforward comparison. In order to compare the two volumes, we cannot take all the voxels into direct consideration. The size of a typical volume data set is quite large, thus it is not feasible to compare all voxels in one set to all voxels in the other volume set. Instead, we need to determine an optimal way to reduce the number of voxels under consideration to just the ones that are important. The basic aim here is the content-based retrieval of representative subsets of volume data. Though we do not know the correspondence between the voxels in each volume data sets, reducing the number of voxels to a few makes it is feasible to compare voxels in either volume.

The steps taken for volume-volume matching is shown in Figure 1.1 and can be given as:

1. *Filter* the volume data to remove or reduce the noise in it.

2. Extract distinctive invariant voxels (*hotspots*) from volumes that can be used to perform reliable matching between different volumes. This can be achieved by retrieving the voxels where sharp change in data values is taking place.

3. Reduce the problem size further by *clustering* and considering the mean of clusters, as it is not feasible to perform all permutations of hotspot-hotspot transformations.

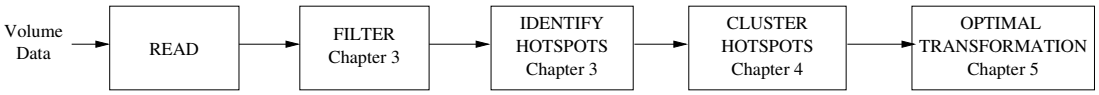


FIGURE 1.1. Pipeline for "volume-volume matching"

4. Application of *Least Square method* to determine the optimal transformation matrix that transforms one set of points to another.

For this project, we used *Terarecon's* vox file format for storing and retrieving the volume data. Vox1999a is a simple, flexible format for voxel data storage, exchange and input and output filter targeting and is operating system and processor independent. The voxel file format (Appendix A) consists of three logical sections - file header, volume description and volume data.

2. PREVIOUS WORK

Our project, called "***volume-volume matching***", aims at matching or comparing two volume data sets. Doctors, for example can extensively use this in medical fields for various purposes, to match the person's brain (MRI data) to a "normally developed" brain data. A major influence for this project has been the work done by David Lowe in 2D image domain. Image matching is one of the fundamental algorithms used in computer vision. The *distinctive image features method* [27] introduced by David Lowe is a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. These features are invariant to image scaling and rotation. The major stages of computation used to achieve the end result are scale space extrema detection, keypoint localization, orientation alignment and keypoint descriptor.

Volume-volume matching includes content-based retrieval of representative subsets of volume using eigenvalues of the Hessian matrix, clustering and least squared method to determine the optimal transformation matrix. Though, this is completely a novice approach in 3D, work has been done to identify important subsets of a volume data set. Jiri Hladuvka [20] introduced *exploiting the eigenvalues of Hessian matrix for volume dissemination* in which they threshold the maximum and the minimum eigenvalues. Further subset reduction was achieved by using filter methods for edge and line detection considering the maxima of first derivative and zero crossing of the second derivative.

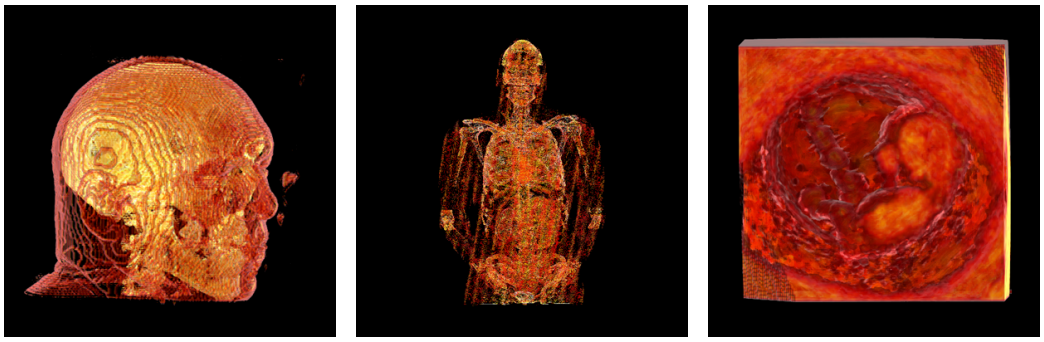
Isosurface algorithms belong to most often used techniques in indirect volume rendering. The choice on the isovalue(s) can be done *interactively* [26] or, after a previous analysis, *semi-automatically* [24], or *automatically* [17]. *Saito* [30]

employs a *non-uniform stochastic Poisson sampling*. Mroz et at. [28] reduced the size of data with respect to applied visualization technique, i.e. *maximum intensity projection* (MIP). *Gradient magnitude* of scalar field can be also considered as a priority function because it emphasizes boundaries, which most commonly attract human attention.

Other popular volume visualization methods are *direct volume rendering* (DVR) and *surface-fitting algorithms* (SF). DVR methods map elements directly into screen space without using geometric primitives as an intermediate representation. SF methods are also called feature-extraction or isosurfacing and fit planar polygons or surface patches to constant-value contour surfaces. *Ray casting* [14] is a well-known method used to render high-quality images of solid objects. *Splatting* [14] is a technique which trades quality for speed. Here, every volume element is splatted onto the viewing surface in back-to-front order. These splats are rendered as disks whose properties (color and transparency) vary diametrically in normal (Gaussian) manner. A new approach to volume rendering was developed by Philippe Lacroute and Marc Levoy [14] in which the viewing transformation is transformed such that the nearest face of the volume becomes axis aligned with an off-screen image buffer with a fixed scale of voxels to pixels. The volume is then renderered into this buffer using the far more favourable memory alignment and fixed scaling and blending factors.

3. IDENTIFYING VOLUME HOTSPOTS

Volumetric data is typically a set S of samples (x, y, z), representing the value v of some property of the data, at a 3-D location (x, y, z) called a volume element, or voxel. If the value is simply a 0 or a 1, with a value of 0 indicating background and a value of 1 indicating the object, then the data is referred to as binary data. The data may instead be multivalued, with the value representing some measurable property of the data, including, for example, color, density, intensity, heat or pressure. The value v may even be a vector, representing, for example, velocity at each location. In our case, "v" contains the "intensity" value at location (x, y, z) and we get this value from the "volume data file". Some of the volume images are shown in Figure 3.1.



(a) smallhead.vox (b) chestir.vox (c) benoit.vox

FIGURE 3.1. Example of Volume Images

Volume visualization is about understanding complex 3-D data and a method of extracting meaningful information from volumetric data. The visualization of volumetric data has aided many scientific disciplines ranging from geophysics to the biomedical sciences.

The algorithm to study and analyze volumes is given as:

1. Data acquisition either via empirical measurement or computer simulation.

2. Put the data into a format that can be easily manipulated. This may entail scaling the data for a better value distribution, enhancing contrast, filtering out noise, and removing out-of-range data. The same set of operations must be applied to all the data slices.

3. Application of transfer function to manipulate and represent data.

4. Display the data as a 3-D point cloud.

This chapter describes the first three steps of the above volume visualization algorithm. Data is acquired from a "volume data file" and stored in an appropriate data structure. Filtering is then performed to reduce the noise in the volume data, followed by retrieving the significant voxels based on eigenvalues of the Hessian matrix at each voxel.

3.1. Volume Data File

For this project, we used *Terarecon's* vox file format [33]. *Vox1999a* (more detail in Appendix I) is a simple, flexible format for voxel data storage, exchange and input and output filter targeting. The file format is operating system and processor independent. It is capable of storing 1, 8, 16, 32 and 64 bit volumes, each of which can contain multiple fields. This format can also store multiple volumes in a single file. Volumes stored in this format consist of slices of voxels with equal distance between the slices. The voxel file format consists of three logical sections - file header, volume description and volume data as shown in Table 3.1.

Section	**Description**
File Header	signature, number of volumes, copyrights and description
Volume Description	volume dimension, voxel size, voxel fields
Volume Data	voxel data streams
Volume Description	...
Volume Data	...
⋮	

TABLE 3.1. Volume data file format

3.2. Voxel Hotspots

Oftentimes in visualization, the goal of using volume datasets is not just to visualize them, but also to analyze and compare them. In order to compare the two volumes, we cannot take all the voxels into consideration. The size of a typical volume data set is quite large (maybe a billion or more voxels), thus it is not feasible to compare all these voxels to all in the other volume set. Instead, we need to determine an optimal way to reduce the number of voxels under consideration to just the ones that are important. The basic aim here is the *content-based* retrieval of *representative subsets* of volume data.

There are many techniques aiming at identifying important subsets of a volume data set. *Isosurface extraction algorithms* [26, 17] belong to the most commonly used techniques for indirect volume rendering. *Gradient magnitude of intensity* can also be considered as a priority function. The voxels that mostly attract human attention usually belong to the strong boundaries that exhibit high values of the gradient magnitude.

In this project, our aim is to extract distinctive invariant voxels from volumes that can be used to perform reliable matching between different volumes. Volume data sets provide a sampled representation of measured or computed quantities that vary across a region of space. There are certain voxels, where data changes more rapidly than other voxels. We are interested in these voxels, where sharp change in data values is taking place. These locations are termed "***hotspots***". We are assuming that the critical points in two volumes of the same relative type, two brains for example, exist in about the same locations in each volume. The number of final voxels or hotspots should be *optimal* - not "*too few*" or "*too many*". If they are too few, certain important details in volume set may be missed and if they are too many, they may over represent the volume and might create problems while clustering.

3.3. Method to Determine Hotspots

A common approach to analyzing the behavior of intensity I in the neighborhood of point x_0 in a 2-D image or a 3-D volume is to consider the initial terms of a Taylor series expansion [27, 20, 21]:

$$I(x0 + \Delta x) \approx I(x_0) + \Delta x^T \nabla I(x_0) + (\frac{\Delta x^T}{2!}) H(x_0) \Delta x + \ldots \tag{3.1}$$

where ∇I is the gradient vector and H denotes the Hessian matrix.

The components of equation 3.1 - gradient vector and the Hessian matrix can also be used separately. The gradient vector is widely used as a normal to an implicitly defined isosurface. Its magnitude provides a tool for edge/boundary detection and in volume visualization can be used as the opacity modulation factor. The Hessian matrix is used whenever a second derivative at a grid point

x_0 and a direction Δx is requested. The last term in equation 3.1, $(\frac{\Delta x^T}{2!})H(x_0)\Delta x$, represents a quadratic form and yields the desired second derivative.

3.3.1. Hessian matrix

The *Hessian matrix* [1] is the square matrix of second partial derivatives of a scalar-valued function. Given the real-valued function $f(x_1, x_2, ..., x_n)$, if all partial second derivatives of f exist, then the Hessian matrix of f is the matrix $H(f)_{ij}(x) = D_i D_j f(x)$ where, $x = (x_1, x_2, ..., x_n)$. That is,

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

The *mixed derivatives* of f are the entries off the main diagonal in the Hessian. Assuming that they are continuous, the order of differentiation does not matter. Hence, $H(f)$ is symmetric because of the equality of mixed partials,

$$\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\right) = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial x}\right),$$

which can also be written as $f_{xy} = f_{yx}$.

In our case, we use the Hessian matrix for determining the hotspots because it determines the ***rate of change of gradient*** in each direction. For a 3-D volume, it is a 3×3 matrix [21] comprised of second partial derivatives of intensity I.

$$H = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix},$$

where $I_{ab} = \frac{\partial^2 I}{\partial a \partial b}$.

3.3.2. Eigenvalues of a Hessian matrix

An $N \times N$ matrix A is said to have an *eigenvector* x and corresponding *eigenvalue* λ if

$$Ax = \lambda x.$$

Jacobi method is employed to be compute the *eigenvalues and eigenvectors* for real-symmetric matrices. The main idea behind the eigenvalue routine is to push the matrix A towards diagonal form by a sequence of transformations:

$$A \ \rightarrow \ P_1^{-1}AP_1 \ \rightarrow \ P_2^{-1}P_1^{-1}AP_1P_2 \ \rightarrow \ P_3^{-1}P_2^{-1}P_1^{-1}AP_1P_2P_3 \ \rightarrow \ \cdots$$

If we compute the diagonal form, then the eigenvalues are given by the columns of transformation $P_1P_2P_3\cdots$. But if we are interested in computing only the eigenvalues and not the eigenvectors, we just have to transform the matrix A to triangular form and can avoid getting all the way to the diagonal form. The Jacobi method consists of a sequence of orthogonal similarity transformations. Each Jacobi rotation eliminates one of the off-diagonal matrix elements. The off-diagonal elements get smaller and smaller until the matrix is reduced to diagonal form by successive rotations. Accumulating the product of the rotations as we go gives the matrix of eigenvectors while the elements of the final diagonal matrix are the eigenvalues.

We incorporate the eigenvalues in our application for two reasons:

- The maximum of the three eigenvalues is used as a marking point in the data because it determines where the *intensity values* are *changing most rapidly.*

- The eigenvalues are *invariant to scaling and rotation*. Eigenvalues are computed from the Hessian matrix and the Hessian matrix in turn is dependent on the gradient of the voxels (gradient of the gradient of an intensity function). Eigenvalues find regions of maximal change and the direction of maximal change. If a volume is rotated, its direction of maximal change changes, but the amount of change does not.

3.4. Noise

Volumes are often degraded by noise. Noise is inherent in data-gathering process and in digital sampling errors. *Noise* [18] is the data without a meaning and can take the form of extreme intensity values and therefore affects how well new intensities are calculated. It is independent of spatial coordinates and it is uncorrelated with respect to volume itself. Some examples of noise are:

- *Gaussian or White noise* [6] - each voxel in the volume is changed from its original value by a small amount.
- *Rayleigh noise* - model noise in range imaging.
- *Shot or Impulse noise* - found in quick transients (e.g faulty switches).
- *Periodic noise* - it is same as the white noise, but present cyclically and manifests itself in volume as a set of sharply defined spikes.
- *Uniform noise* - noise which is uniformly distributed.
- *Salt and pepper noise* [6] - the color of noisy voxel bears no relation to the color of the surrounding voxels i.e. they are vastly diferent in color from their neighboring voxels.

3.4.1. Cause of noise in our application

The raw volumes often contain considerable noise. Figure 3.3-(a) shows the "*smallhead.vox*" volume containing noisy data. Noise has a great impact on our application. The use of neighboring voxels on either side of voxel v to calculate the gradient at v, produces a gradient that is properly centered, but very sensitive to noise. This gradient is used to compute the Hessian matrix that makes the Hessian matrix sensitive to noise as well. The hotspots achieved in this stage are clustered in the next stage. Clustering is a useful exploratory tool, but the cluster results are very sensitive to noise.

3.4.2. Removal of noise by filtering

One of the most common problems in volume analysis is estimation and removal of noise or other artifacts by using spatial filters. In order to achieve good volume-volume matching results, it is very important to get rid of or to reduce noise in volume data. This can be achieved by filtering. Common techniques [18] include mean filtering, median filtering, Gaussian filtering and anisotropic filtering.

3.4.2.1. Mean Filter

The mean filter [9] is a simple, intuitive, sliding window spatial filter for smoothing volumes i.e. reducing the amount of intensity variation between one voxel and the other. This is a *low pass filter*, which is also good at reducing *Gaussian noise*. The idea behind mean filtering is to replace each voxel value in the volume with the "mean" or "average" value of its neighbors, including itself.

This has an effect of de-emphasizing voxel values that are unrepresentative of their surroundings. The results of mean filtering on original volume "smallhead.vox" is shown in Figure 3.3-(b). Mean filtering is often thought of as a convolution filter that is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean. Often a $3 \times 3 \times 3$ cube kernel is used for volumes as shown in Figure 3.2.

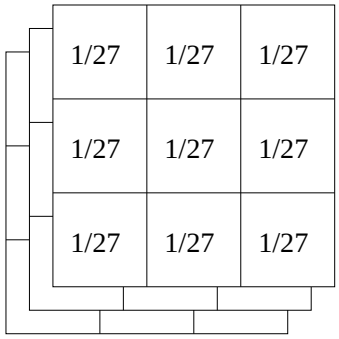
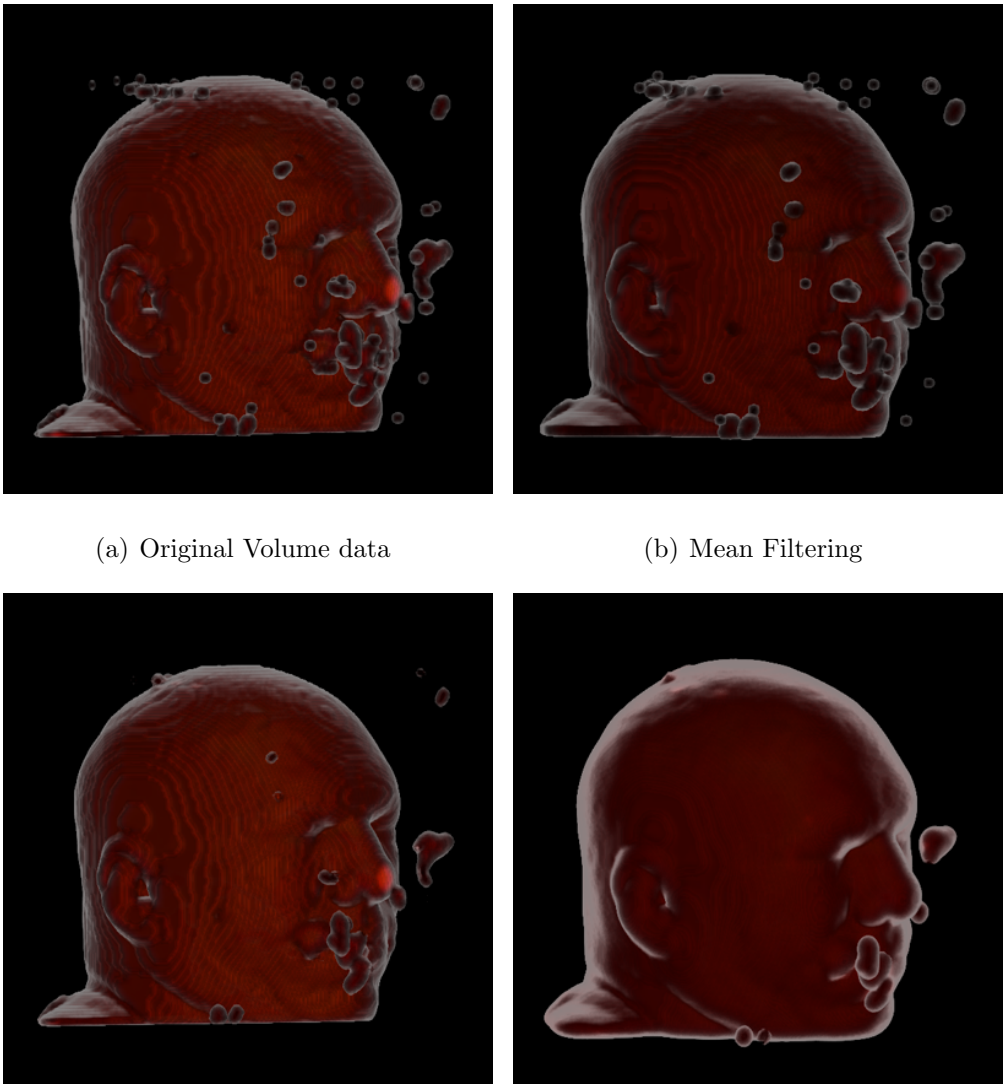


FIGURE 3.2. $3 \times 3 \times 3$ mean kernel

3.4.2.2. Median Filter

Median filtering [10] is a simple and very effective noise removal filtering process and is shown in Figure 3.3-(c). Its performance is particularly good for removing *shot noise*. Shot noise consists of strong spikelike isolated values. This filter considers each voxel in the volume in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of replacing the pixel value with the mean of neighboring pixel values, it replaces it with the "median" of those values. The median is calculated by first sorting all the voxel values from the surrounding neighborhood into numerical order and then replacing the voxel under consideration by the middle ($13^{th}/27$) voxel value.



(a) Original Volume data

(b) Mean Filtering

(c) Median Filtering

(d) Gaussian Filtering

FIGURE 3.3. Example of different filtering strategies on “smallhead.vox”

The median filter has two main advantages over the mean filter:

- A single very unrepresentative pixel in a neighborhood will not affect the median value significantly because median ignores severe outliers.
- It preserves the useful detail like the sharp edges that the mean filter cannot retain.

One of the *major problems* with the median filter is that it is relatively expensive to compute. To find the median it is necessary to sort all the values in the neighborhood into numerical order which is relatively slow, even with fast sorting algorithms such as quicksort.

Our aim has been the quality of output rather than the speed of the program. Once the desired quality of output is achieved, doing computation in parallel i.e. by involving multiple processors can make the whole program faster, but we keep that for future enhancements.

3.4.2.3. Gaussian Filter

The Gaussian smoothing [7] operator is a convolution operator used to blur volumes and remove detail and noise. The result of Gaussian filtering applied to "smallhead.vox" volume data containing noise is depicted in Figure 3.3-(d). It is similar to mean filtering, but uses a different kernel that represents the shape of a Gaussian hump. In order to perform Gaussian filtering in three dimensions, the 3-D convolution can be performed by first convolving with a 1-D Gaussian in the x direction, then convolving with another 1-D Gaussian in the y direction, and then convolving with yet another 1-D Gaussian in the z direction. The Gaussian distribution in 1-D is given in equation 3.2:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}. \tag{3.2}$$

For our application, we take $\sigma = 1.4$. In general terms, to preserve finer details, we pick a smaller sigma and to leave only coarser details, we pick a larger sigma.

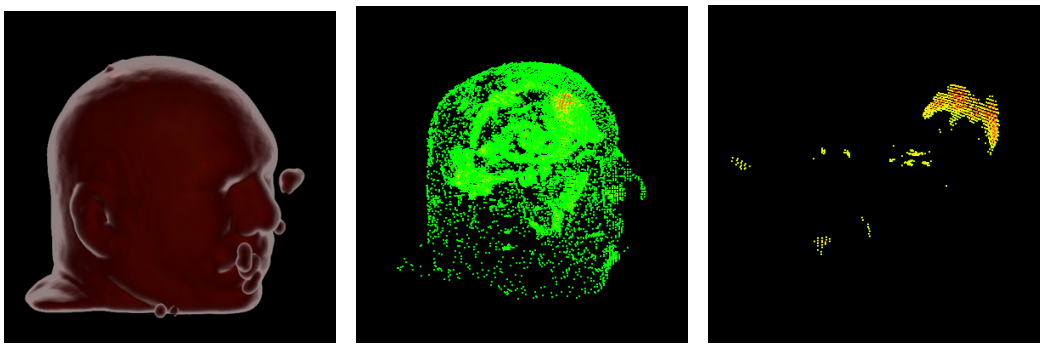
3.5. Results

3.5.1. Threshold - maximum eigenvalue

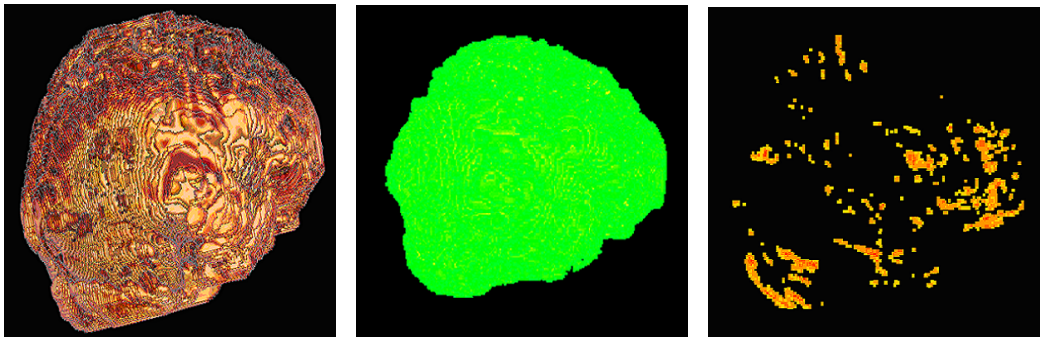
The maximum of the three eigenvalues obtained from the Hessian matrix is used as a threshold value. The threshold value should be adjusted in such a manner that the final number of voxels (hotspots) are neither "*too few*" nor "*too many*". Figure 3.4-(a), (d), (g) shows the original volume sets - "smoothhead.vox", a head with ($128 \times 128 \times 128$ voxels), "toga.vox", a brain with ($200 \times 200 \times 200$ voxels) and "twins.vox", a 3D ultrasound with ($128 \times 128 \times 128$ voxels) and (b), (c), (e), (f), (h) and (i) shows the reduced set of voxels that represent the entire volume.

3.5.2. Results of the "Test Case"

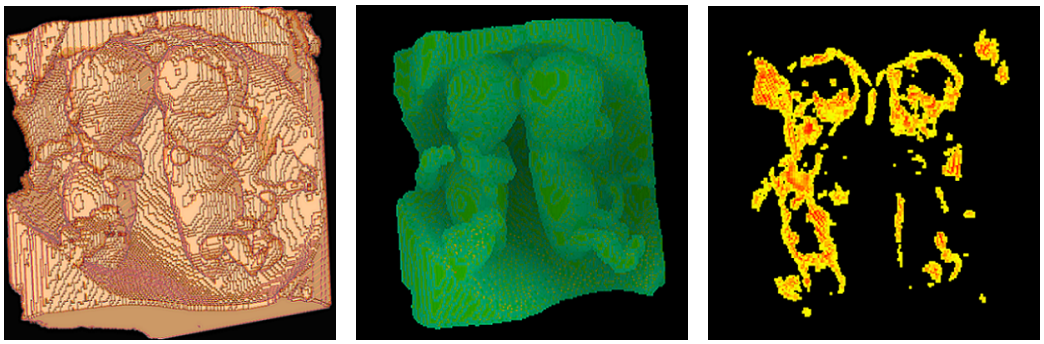
Test Case - For the test case, we took an original volume data "*smoothhead.vox*" and a transformed volume data "*smoothrotate.vox*", which is the original volume data rotated by an angle of 45 degrees about Y-axis. Since we know that the eigenvalues are rotationally invariant, the two volumes should come up with almost the same eigenvalues. We selected this as the test case because if this volume matching doesn't work successfully, we cannot expect real volume-volume matching to work at all.



(a) $128 \times 128 \times 128$ (b) 852585 (c) 977



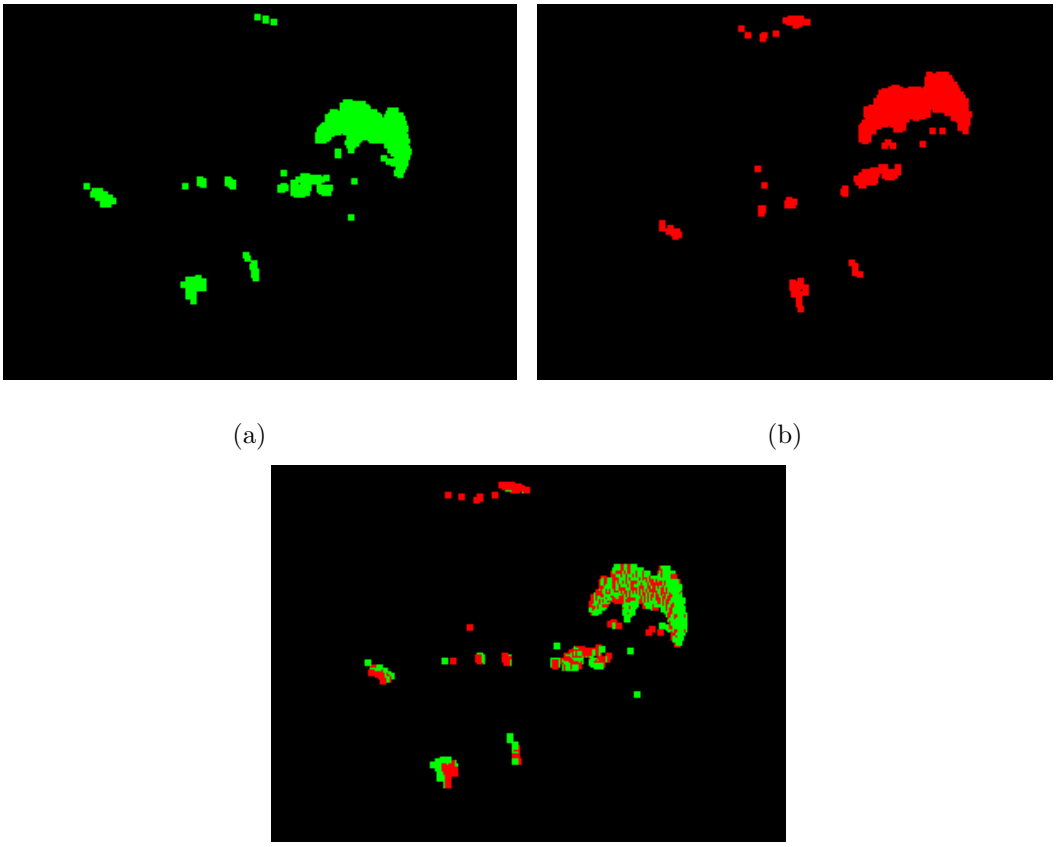
(d) $200 \times 200 \times 200$ (e) 852585 (f) 1890



(g) $128 \times 128 \times 128$ (h) 167276 (i) 3644

FIGURE 3.4. Thresholding hotspots : (a), (d) and (g) original volume files and (b), (c), (e), (f), (h) and (i) important subsets of volume data. The numbers below the figures are the number of voxels in the volume.

We observed the same hotspots in both Figure 3.5-(a) and (b), despite transformation of the second. We rotated the second set by 45 degrees about



(a) (b)

(c)

FIGURE 3.5. Results for "Test Case": (a) Hotspots for smoothhead.vox (b) Hotspots for smoothrotate.vox (c) Combined Hotspots

Y-axis and observed that the hotspots are almost in the same relative locations as shown in Figure 3.5-(c). This proved rotational invariance and marked the successful achievement of the first stage in volume-volume matching.

Clearly, we cannot perform all permutations of hotspot-hotspot transformations, so we needed to reduce the problem size by ***clustering*** the hotspots.

4. CLUSTERING

After the previous stage, in which we determine the hotspots, we were left with almost 1000 voxels in each data set. Clearly, in order to compare two volume data sets we cannot perform 1000! combinations as we do not know the correspondence between the hotspots. We have to further reduce these voxels to a few, so that it is feasible to perform volume-volume matching. This is accomplished by *clustering* the data and then taking the *mean* of the clusters.

Clustering [4] is the classification of similar objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset share some common trait - often proximity according to some defined distance measure. There are many clustering methods present and each one of them is capable of giving a different grouping of data sets. They can be ***hierarchical*** or ***partitional***. Hierarchical algorithms find successive clusters using previously established clusters, whereas partitional algorithms determine all clusters at once.

The outline of this chapter is as follows: We tried several clustering algorithms starting with *median cut*. Not satisfied with the results achieved by the original median cut, we performed some *modifications* to it by not limiting it to X, Y and Z axis subdivision and determining the cutting plane based on exhaustive search using the Golden Ratio method. It definitely improved the results, but still it wasn't exactly what we wanted. It was observed that this clustering wasn't rotationally invariant and the sum of squared error metric used was very sensitive to outliers, and it degraded the performance of clustering. After trying the *k-means* algorithm, we observed the performance of all the three clustering algorithms tried so far. We concluded that in our case, *hierarchical clustering* would be better than

partitional clustering because it follows a bottom-up approach rather than top-down. We finally tried the hierarchical clustering algorithm which was robust to outliers and gave good clustering results.

4.1. Median Cut

The concept behind the median cut algorithm introduced by *Heckbert* [19, 31] is to use each of the colors in the synthesized colormap to represent a representative number of pixels in the original image. This algorithm repeatedly subdivides the color space into smaller and smaller rectangular boxes. It starts with one box, which tightly encloses the colors of all pixels from the original image.

Iteration step: Split a box. The box is "shrunk" to fit tightly around the points (colors) it encloses, by finding the minimum and maximum values of each of the color coordinates. "Adaptive partitioning" is then used to decide which way to split the box. The enclosed points are sorted along the *longest dimension* of the box, and segregated into two halves at the median point. Approximately equal numbers of points will fall on each side of the cutting plane.

The above step is recursively applied until K boxes are generated. After K boxes are generated, the representative for each box is computed by averaging the colors contained in each.

Other criteria could be used to decide which coordinate to bisect. Instead of choosing the coordinate with the largest range, one might use the one with the *largest variance* or choose the split plane so that the *sum of variances for the two new boxes is minimized*. This would tend to minimize the mean squared error better than the median criterion.

We applied this clustering process to our $3D$ points. It was observed that the clusters achieved were not optimal as the performance of clustering was being highly affected by outliers and axis aligned cutting planes. The ***result*** for 5 clusters using Median Cut is shown in Figure 4.1.

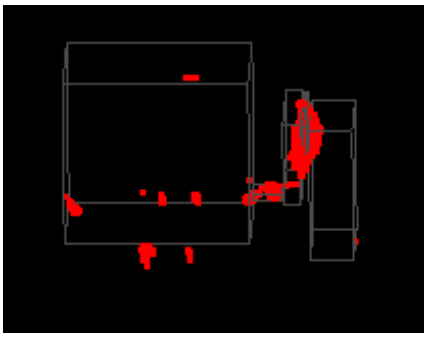


FIGURE 4.1. Results of Median Cut clustering on "smoothhead.vox" (We can clearly see that *axis aligned* feature and *cutting at median* feature results in poor clustering.)

We observed the ***drawbacks of the Median Cut clustering algorithm*** when applied to our data set as:

- It forces the clustering alignment to be parallel to the principal axis (X, Y and Z).
- When a single cluster contains disproportionately more points than the rest, the median of all points may lie in the middle of the large cluster, forcing the median cut algorithm to erroneously divide that cluster into two. This scenario is clearly depicted in the Figure 4.1.
- This clustering method is very sensitive to outliers.

4.2. Improved Median Cut

Some changes were incorporated in the Median Cut algorithm so as to improve the clustering results.

We start with one box that tightly encloses the points. The box is "shrunk" to fit tightly around the points it encloses, by finding the maximum and minimum values of each of the point coordinates.

Next, we used "adaptive partitioning" to decide which way to split the box. According to the original Median Cut, the enclosed points are sorted along the longest dimension of the box, and segregated into two halves at the median point. We achieved better results by instead considering squared *Manhattan distance* (which is rotationally invariant) as a metric to determine error. The two main modifications made were:

1. Use a *Least Square Plane* [8] approach to determine the $3D$ orientation for the optimal bounding box. This was accomplished by first computing the centroid of the points present in the *bounding box*, as the least square plane is guaranteed to pass through this point. A new list was then created in which the points were centered around the centroid and a least squares 3×3 matrix was formed. The 3 eigenvectors of this matrix determined the three principle directions. Computing the minimum and maximum "d" (in plane equation: $ax + by + cz + d = 0$) values for each principle direction defined the search space to look for the optimal cutting plane location.

2. Determine the best location to split that box using the *Golden Ratio optimal search* [29] technique. After getting the bounding planes in the three principle directions using a *Least Square Plane*, we find two additional planes in

either directions based on the Golden Ratio. If P and Q are the bounding planes, d value of two additional planes R and S can be given by:

$R_d = P_d + G \times (Q_d + P_d)$

$S_d = P_d + (1 - G) \times (Q_d + P_d)$,

where G (Golden Ratio) = 0.38197.

Now, we determine the Manhattan Distance (M_d) [22] for each of the planes using:

$$M_d = max(\sum_{i=0}^{n}(P_i - \hat{P})^2, \sum_{j=0}^{m}(P_j - \hat{P'})^2)$$

where P_i are the points on left side of the plane,

$\hat{P}$ is the mean of points on the left,

P_j are the points on the right side of the plane,

$\hat{P'}$ is the mean of points on the right.

Based on this distance, we eliminate one of the bounding planes and repeat the process until the distances of the two planes in the middle are the same. This is done for each principle direction and the plane with the least error is selected as the *cutting plane*.

The steps performed durng Improved Median Cut clustering algorithm can be shown in Figure 4.2.

The Improved Median Cut algorithm was applied to the points achieved after the first stage. The ***results*** are shown in Figure 4.3.

After the implementation of Improved Median Cut, the results of the clustering stage improved a lot and the error metric was considerably smaller($\sim 60\%$) as compared to Median Cut. It still did not satisfy the results that we expected. Some of the ***drawbacks*** of this algorithm were:

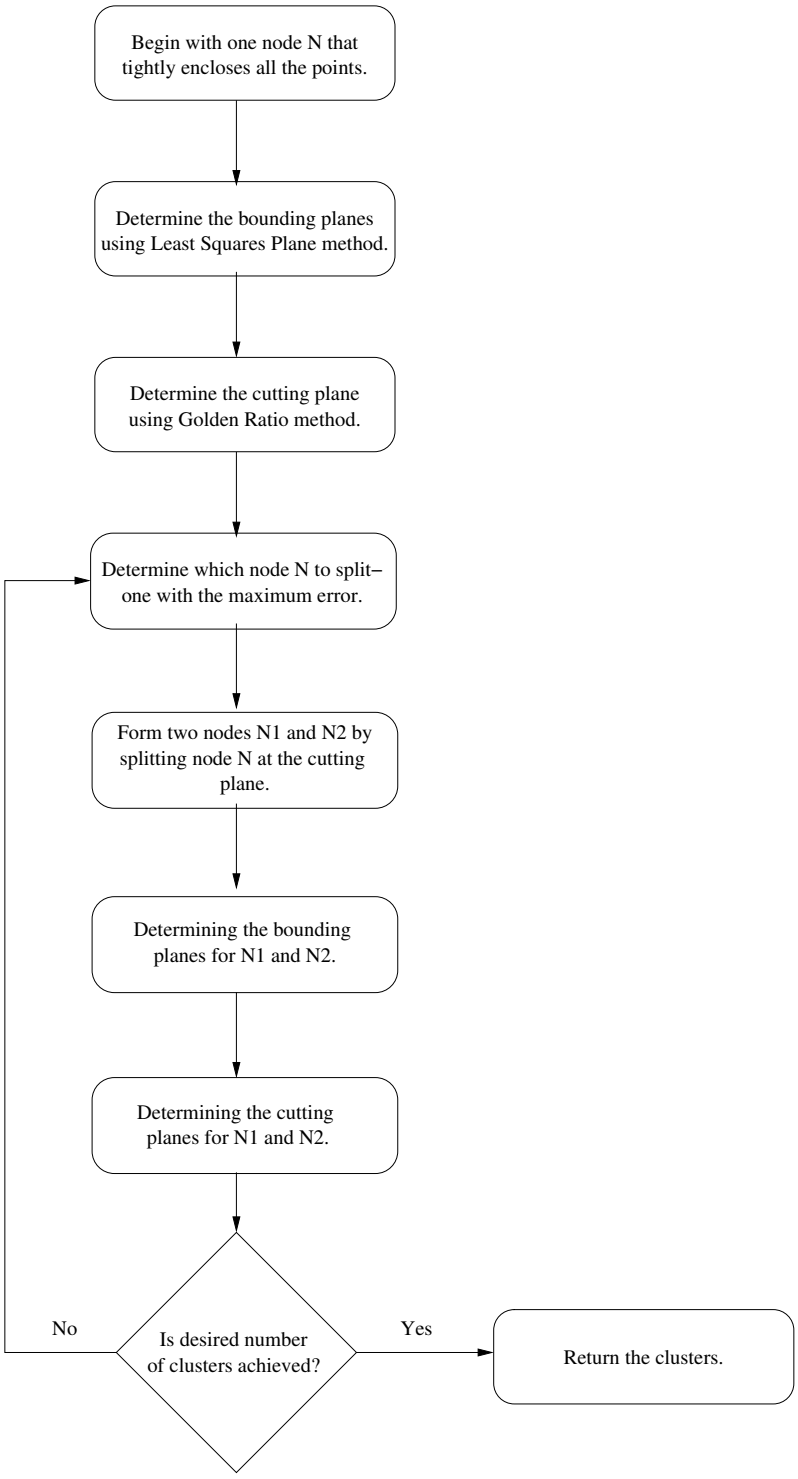
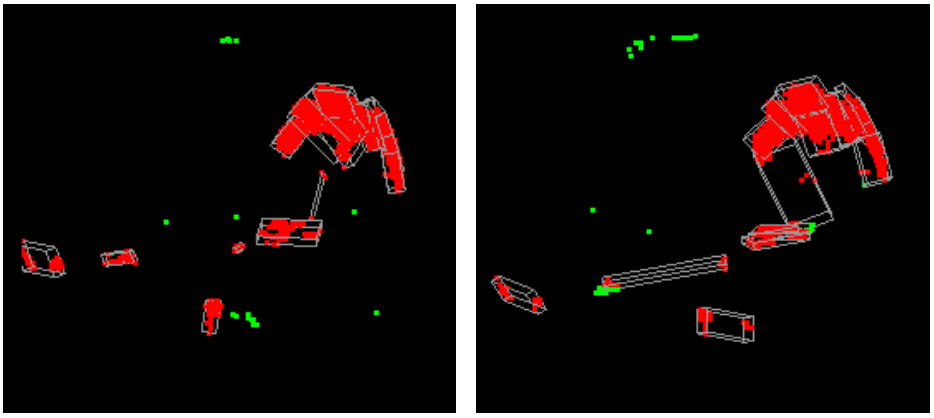


FIGURE 4.2. Algorithmic flow of Improved Median Cut algorithm



(a) smoothhead.vox (b) smoothrotate.vox

FIGURE 4.3. Results of Improved Median Cut on "Test Case" (11 clusters are represented by 11 bounding boxes)

- It partitioned a cluster containing many points closer to one another rather than partitioning another cluster that had many small sub-clustered points separated from other sub-clusters in it. This scenario is clearly depicted in Figure 4.3 where the big cluster is partitioned into 6 sub-clusters when there are some small clusters which should have been 2 clusters rather than one.
- The results were not as rotationally invariant as we expected because of the round-off error when rotated.
- The results were sensitive to outliers.

4.3. k-means Clustering

K-means is one of the simplest unsupervised learning algorithms [4, 32] and hence an easy way to quantize a given data set into a given number of clusters. The main idea is to define k centroids, one for each cluster. These centroids

should be placed as far from each other as possible because different locations cause different results. The next step is to take each point belonging to the data set and associate it to a nearest centroid. After associating all the points, the first groupage is over. Now we recalculate the new k-centroids and new binding has to be done for these same data points and new nearest centroids.

This algorithm aims at minimizing the objective function [32] which in this case is ***squared error*** function. The ***objective function*** is:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} ||x_i^{(j)} - c_j||^2,$$

where $||x_i^j - c_j||^2$ is a chosen data measure between a data point, $x_i^{(j)}$ and cluster center c_j, is an indicator of distance of n points from their respective cluster centers.

The ***algorithm for k-means clustering*** can be given as and the flow of execution can be shown in Figure 4.4:

1. Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.

2. Assign each object to the group that has the closest centroid.

3. When all objects have been assigned, recalculate the positions of the K centroids.

4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

For successful k-means clustering, the initial centroids should be carefully selected and should be as far from each other as possible. For this project, out of

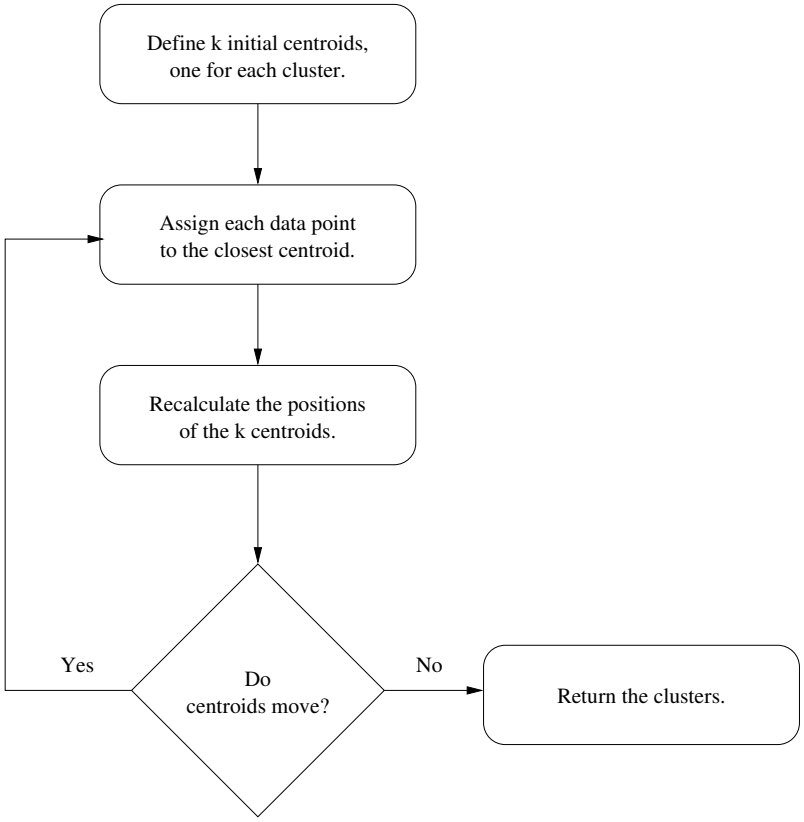
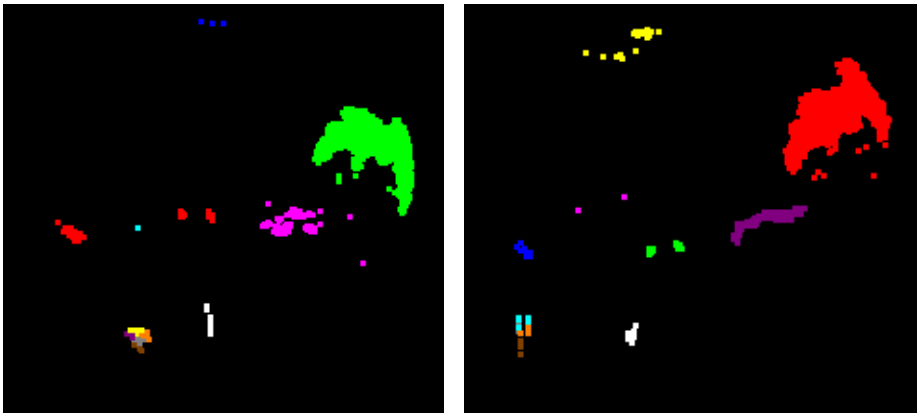


FIGURE 4.4. Algorithmic flow of k-means clustering algorithm

all the points, we selected two points which were farthest from each other; then we selected the third point which was farthest from those two points and so on. This way we got 11 initial seeds for the algorithm. The ***results*** achieved through k-means are shown in Figure 4.5.

Some of the observed ***drawbacks of k-means*** [5] are:

- Since all the attributes have the same weight, we can never determine which attribute contributes more to the grouping process.
- It is sensitive to the initial condition. Different initial conditions can produce different clusterings.



(a) smoothhead.vox (b) smoothrotate.vox

FIGURE 4.5. Results of k-means clustering on "Test Case" (11 clusters are represented by 11 different colors)

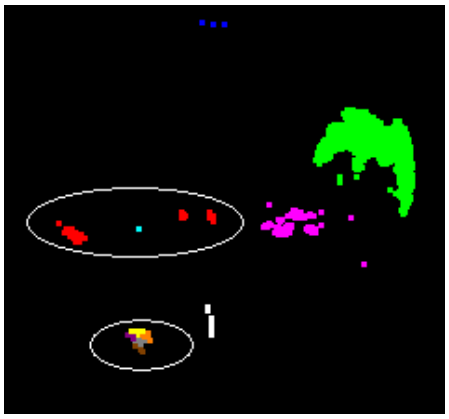


FIGURE 4.6. Drawbacks of k-means (The highlighted region above should have been 3 clusters whereas the one below should have been a single cluster.)

- We never know which attribute contributes more to the grouping process since we assume that each attribute has the same weight.
- The arithmetic mean is not robust to outliers.

The combined effect of all these drawbacks is depicted in Figure 4.6.

4.4. Hierarchical Clustering

Hierarchical cluster analysis [4, 32] is a statistical method for finding relatively homogeneous clusters of points based on measured characteristics. It can be either *agglomerative* (iteratively joining small clusters into larger) or *divisive* (starting with one cluster and separating these into several smaller). Hierarchical clustering starts with each point in a separate cluster and then combines the clusters sequentially, reducing the number of clusters at each step until only one cluster is left. When there are N cases, this involves $N - 1$ clustering steps, or fusions.

In hierarchical clustering each sample is initially thought off as a cluster. A similarity measure is computed between all pairs of clusters and the two most similar are grouped together as a new cluster. This procedure is repeated until there is only one cluster left or some cut-off point of minimum similarity is reached.

1. Initialize every point as a cluster.
2. While more than one cluster remains
   (a) Calculate similarity between all the pair of clusters.
   (b) Find the most similar pair of clusters.
   (c) Merge the two clusters.
3. End.

An important component of hierarchical clustering is the ***distance measure*** [25] between the points. If the components of the data instance vectors are all in the same physical units then **Euclidean distance** metric is sufficient to successfully group similar data instances. In a univariate example the Euclidean

distance between two values is the arithmetic difference, i.e. $value_1 - value_2$. In the bivariate case, the minimum distance is the hypotenuse of a triangle formed from the points. For three variables the hypotenuse will be extended through 3-dimensional space. This method has certain advantages (e.g., the distance between any two objects is not affected by the addition of new objects to the analysis, which may be outliers).

$EuclideanDistance(x, y) = \{\sum_i (x_i - y_i)^2\}^{1/2}$.

City-block (Manhattan) distance is another distance metric. This distance is simply the sum of difference across dimensions. The effect of a single large difference in one variable is dampened and Manhattan distance is more robust to outliers.

$ManhattanDistance(x, y) = \sum_i |x_i - y_i|$.

In order to place progressively greater weight on objects that are further apart, we may want to **square the standard Euclidean distance**.

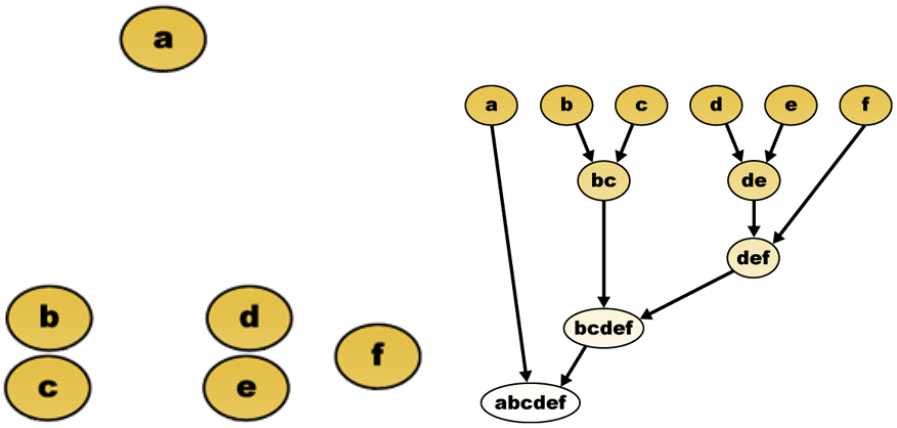
$SquaredManhattanDistance(x, y) = \sum_i (x_i - y_i)^2$.

4.4.1. Representation

This hierarchical clustering process can be represented as a *tree*, or *dendrogram* [4] (binary tree with a distinguished root) as shown in Figure 4.7, where a join of the tree illustrates each step in the clustering process.

4.4.2. Agglomerative Methods

An agglomerative hierarchical clustering [2] procedure produces a series of partitions of the data, $P_n, P_{n-1}, \ldots, P_1$. The first P_n consists of n single object "clusters", the last P_1, consists of a single group containing all n points. At



(a) set of 6 points (b) dendogram

FIGURE 4.7. Hierarchical Clustering based on Euclidean Distance (image taken from Wikipedia [19])

each particular stage the method joins together the two clusters that are closest together (most similar). There are different ways of defining the similarity between clusters.

4.4.2.1. Single Linkage Clustering

This is one of the simplest agglomerative hierarchical clustering methods also known as the *nearest neighbor* technique. The defining feature of the method is that distance between clusters is defined as the distance between the *closest* pair of points as shown in Figure 4.8-(a), where only pairs consisting of one point from each cluster are considered.

$D(r, s) = min\{d(i, j)\}$, where point i is in cluster r and point j is cluster s.

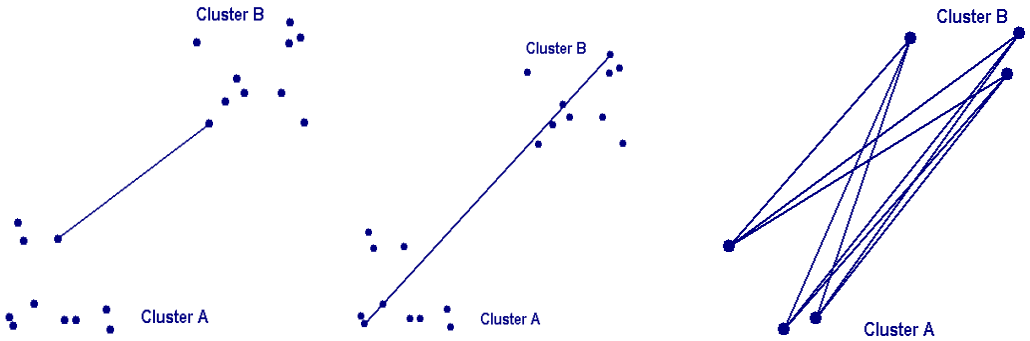
The distance between two clusters is given by the value of the shortest link between the clusters.

4.4.2.2. Complete Linkage Clustering

Complete Linkage Clustering is also known as *farthest neighbor* technique. Distance between clusters is defined as the distance between the most distant pair of points, one from each cluster.

$D(r, s) = max\{d(i, j)\}$, where object i is in cluster r and object j is in cluster s.

The distance between two clusters is given by the value of the longest link between the clusters as shown in Figure 4.8-(b).



(a) Single Linkage (b) Complete Linkage (c) Average Linkage

FIGURE 4.8. Agglomerative Hierarchical Methods

4.4.2.3. Average Linkage Clustering

The distance between two clusters is defined as the average of distances between all pairs of points as shown in Figure 4.8-(c), where each pair is made up of one point from each cluster.

$$D(r, s) = \frac{T_{rs}}{(N_r \times N_s)}$$

where, T_{rs} is the sum of all pair wise distances between cluster $\boldsymbol{r}$ and cluster $\boldsymbol{s}$. N_r and N_s are the sizes of the clusters r and s respectively.

4.4.3. Single Linkage Clustering Algorithm

In our application, we make use of *Single Linkage Clustering* [13] because we want to cluster the points which are closer to one another based on Euclidean distance. The algorithmic flow is shown in Figure 4.9 and the algorithm is as follows:

1. Begin with disjoint clustering having level $L(0) = 0$ and sequence number $m = 0$.

2. Find the least dissimilar pair of clusters in the current clustering, say pair (r), (s), according to $d[(r), (s)] = mind[(i), (j)]$, where the minimum is over all pairs of clusters in the current clustering.

3. Increment the sequence number: $m = m+1$. Merge clusters (r) and (s) into a single cluster to form the next clustering m. Set the level of this clustering to $L(m) = d[(r), (s)]$.

4. Update the proximity matrix, D, by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and column corresponding to the newly formed cluster. The proximity between the new cluster, denoted (r, s) and old cluster (k) is defined in this way: $d[(k), (r, s)] = mind[(k), (r)], d[(k), (s)]$.

5. If all objects are in one cluster, stop. Else, go to step 2.

4.4.4. Results of Hierarchical clustering

Finally after removing the outliers, 8 clusters are considered (shown in 8 different colors) in Figure 4.10.

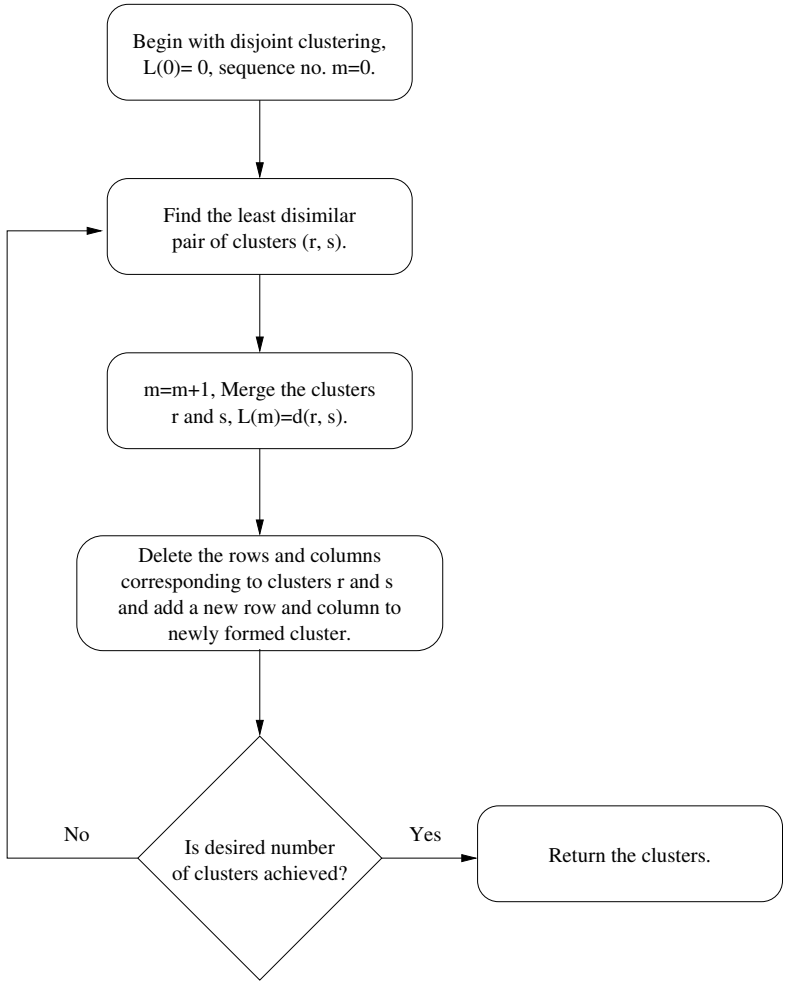


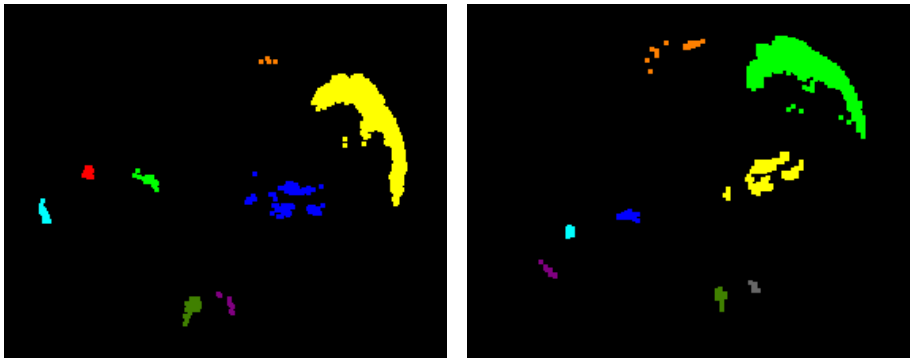
FIGURE 4.9. Algorithmic flow of Hierarchical clustering algorithm

4.5. Outliers

An outlier [16] is an observation or subset of observations, which appear to be inconsistent with remainder of that set of data.

Possible ***sources of outliers*** [11, 16] are:

- Data recording or entry errors.
- Incorrect assumption about the data distribution can lead to mislabeling data as outliers.



(a) smoothhead.vox (b) smoothrotate.vox

FIGURE 4.10. Results of Hierarchical clustering applied on "Test Case"(8 clusters shown in 8 different colors.)

- "Rare" event syndrome. Another reason for outliers is the "rare" event syndrome - extreme observations may occur that for some legitimate reason do not fit within the typical range of other data values.
- Noise in data sets.

Developing techniques to look for outliers and understanding how they impact data analysis are extremely important parts of a thorough analysis, especially when statistical techniques are to be applied to the data. *Several problematic effects* of outliers include:

- Any statistical test based on sample means and variances can be distorted in the presence of outliers. Regression coefficients estimated that minimizing the Sum of Squares for Error (SSE) are very sensitive to outliers.
- Sums of squares are inflated which make it unlikely you will partition sources of variation in the data into meaningful components.
- They may cause bias or distortion of estimates.

- They may lead to faulty conclusions.

Thus, it becomes really important to detect the outliers in data and remove them in order to get desirable results.

4.5.1. Detecting and removing outliers

One of the commonly used methods to detect outliers is ***Interquartile Range (IQR)***. IQR is the width of the box in a box-and-whisker plot [3, 23]. Statistics assumes that data values are clustered around some central value. The IQR tells how spread out the "middle" values are; it can also be used to tell when some of the other values are "too far" from the central value. These "too far away" points are called "outliers", because they "lie outside" the range in which we expect them. Figure 4.11-(a) shows the clustering of "smoothhead.vox" with boxes representing the clusters and points in *green* being detected as outliers using Inter-Quartile Range shown in elipitical regions.

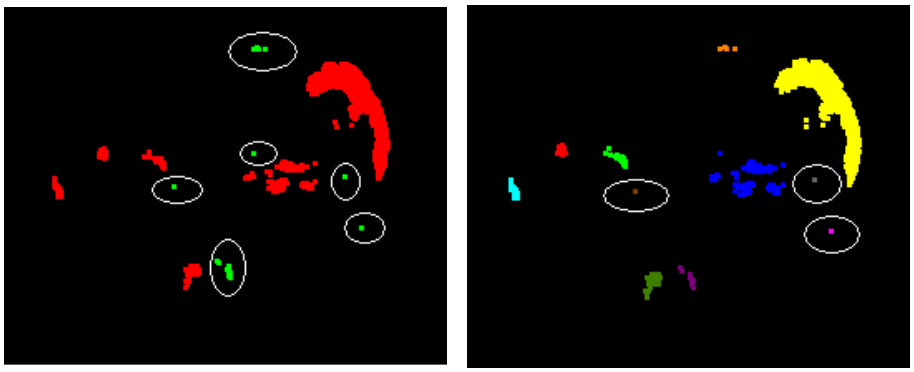
Defining Q_1 and Q_3 to be first and third quartiles, and IQR to be Interquartile Range $(Q_3 - Q_1)$, mild outlier can be identified in the range:

$$< Q_1 - 1.5 \times IQR \quad \text{or} \quad > Q_3 + 1.5 \times IQR.$$

Q_1 and Q_3 define the so-called *inner fences*, beyond which an observation would be labeled a *mild outlier*. *Extreme outliers* are the values beyond outer fences i.e

$$< Q_1 - 3 \times IQR \quad \text{or} \quad > Q_3 + 3 \times IQR.$$

Another method that could be used to *detect outliers with hierarchical clustering* is to ***threshold the distance metric*** used for clustering. This step can be preformed in the beginning, by setting the threshold on Euclidean Distance and ignoring the points whose distance is greater than the threshold value. The hierarchical clustering works in a way that it finds out the two closest points and



(a) Inter Quartile Range (b) Threshold Euclidean Distance

FIGURE 4.11. Detecting outliers for "smoothhead.vox"

clusters them together. This process is repeated until desired number of clusters is achieved. The outliers are the points that are widely separated from the data and hence after clustering they will exist in separate clusters and can also be ignored at this time. These clusters are shown in Figure 4.11-(b).

Finally after clustering, the average points of the boxes were selected and Least Squared methods was applied to determine the final transformation matrix by which the second volume data set was transformed so that it optimally aligned with the first one.

5. OPTIMAL TRANSFORMATION

Given two sets of points in correspondence with each other, it is straightforward to determine the transformation matrix that transforms one set of points to other. But in our application, after successful clustering we are left with only a few points ("8" points for "Test Case") in each of the data sets, with *no knowledge* of which clusters in one data set might correspond best with which clusters in the other data set. Hence, we consider the original cluster average points and all possible *permutations* of transformed cluster average points to yield a transformation matrix for each combination. But there is just one optimal transformation matrix and one optimal ordering on the transformed set of points. We incorporate the *sum of squared error metric* on each transformation matrix and the one with *least error* corresponds to the *optimal transformation matrix*. ***In this chapter, we discuss the following things:***

- To compute all the possible permutations of a set of points.
- Computation of the transformation matrix.
- Error metric to determine the optimal transformation matrix and optimal ordering on transformed points.
- Decomposition of the transformation matrix into translation, scaling, shear and rotation matrices.

5.1. Permutations

The *permutation* of a set is the number of ways that the items in the set can be uniquely ordered. For example, the permutations of the set $\{1, 2, 3\}$ are

$\{1,2,3\}$, $\{1,3,2\}$, $\{2,1,3\}$, $\{2,3,1\}$, $\{3,1,2\}$ and $\{3,2,1\}$. For N objects, the number of permutations is $N!$ (N factorial, or $1 \times 2 \times 3 \times ...N$). Permutation sets are usually calculated via *recursion*.

5.1.1. Algorithm to compute permutations

The algorithm used to compute the permutations is explained below [12]: Lets take a string "123". If we rotate this string in a circular manner, we get 231 and 312. If we find one string of $N!$ permutations, we can produce N permutations by rotating it circularly. This reduces time that is proportional to N. The greater the value of N, the more the algorithm is optimized. In short, the algorithm finds a number and rotates the number circularly N times to get N numbers.

5.1.2. Application of the algorithm

Application of the above algorithm to strings of various lengths can be seen as:

Case 1: Let the string be $\{12\}$ of length 2. The unique pattern is 12 and if we rotate it we get 21.

Case 2: Let the string be $\{123\}$ of length 3. The unique patterns are as follows:

1(23): 123 is unique pattern resulting in (123).

1(32): Rotate the pattern (23) to get (32) resulting in a pattern (132).

(231): By rotating the pattern (123).

(312): By rotating the pattern (123).

(321): By rotating the pattern (132).

(213): By rotating the pattern (132).

The output patterns are:

Level 1	Level 2	Permutations
123		
	1(23)	123
		231
		312
	1(32)	132
		321
		213

5.2. Determining the optimal transformation matrix

We initially start with two matrices, one is a $4 \times n$ matrix consisting of *original* set of points and the other is $3 \times n$ matrix consisting of *transformed* set of points, where n is the number of points in each of them. The original set contains x, y, z and w co-ordinates of points where "w" is the perspective component and transformed set contains x, y and z co-ordinates. Now, our basic goal here is to compute the ***transformation matrix*** (M) as:

$$[P']_{3\times n} = [M]_{3\times 4}[P]_{4\times n}, \tag{5.1}$$

where $P' = 3 \times n$ matrix containing original points,
$M = 3 \times 4$ transformation matrix, and
$P = 4 \times n$ matrix containing transformed points.

Since P is not a square matrix, we cannot directly compute the inverse of P. We have to perform some manipulation on Equation 5.1 so that P turns into a square matrix and we can easily compute the transformation matrix as $[M] = [P'][P]^{-1}$.

The task now is to convert P to a square matrix so that Equation 5.1 can be easily computed. For this, we need to consider the *transpose* of a matrix. (Transpose of a matrix is a matrix formed by turning all the rows of a given matrix into columns and vice-versa. The transpose of matrix P is written P^T)
We ***post multiply*** both sides of Equation 5.1 by transpose of P (P^T).

$$[P']_{3\times n}[P]^T_{n\times 4} = [M]_{3\times 4}[P]_{4\times n}[P]^T_{n\times 4}. \tag{5.2}$$

After multiplying matrix P' by P^T, we get a 3×4 matrix and multiplying P by P^T yields a 4×4 matrix. Equation 5.2 hence reduces to:

$$[P'_{new}]_{3\times 4} = [M]_{3\times 4}[P_{new}]_{4\times 4} \quad , or$$

$$[M]_{3\times 4} = [P'_{new}]_{3\times 4}[P_{new}]^{-1}_{4\times 4}.$$

Now, we have to determine the *inverse of a square matrix.*

This method is incorporated to compute the inverse of P_{new} matrix in our application. Now that we have successfully computed the inverse of the square matrix (P_{new}), we can simply use it to get the final transformation matrix (M) by post multiplying the inverse P_{new} matrix by P'_{new} matrix.

5.3. Error Metric

Given two sets of points, we take the original set of points as a $4 \times n$ matrix (P) and for all possible permutations of the transformed set of points which is a $3 \times n$ matrix (P'), we compute the transformation matrix (T) as given by Equation 5.1. Now out of all these transformation matrices, there is just one matrix that is optimal i.e. there is one best correspondence between the two sets of points. In order to determine this optimal transformation matrix, we make use of an error

metric. After computing the transformation matrix (T) for each permutation, we multiply this transformation matrix by the original set of points to get the new transformed matrix ($\hat{T}$). The ***sum of squared error (sse)*** between the elements of the transformed matrix and new transformed matrix is used as a metric to determine the optimal transformation matrix as shown in Equation 5.3.

$$sse = \sum_{i=0}^{3}\sum_{j=0}^{4}(T_{ij} - \hat{T}_{ij})^2 \tag{5.3}$$

5.4. Decomposition of Transformation Matrix

It is easier to view a transformation matrix in terms of simple transformations such as scale, rotation etc. that reproduces a given transformation matrix. Concatenating a sequence of transformations in the order forms the transformation matrix:

$$Scale(s_x, s_y, s_z)Shear_{xy}Shear_{xz}Shear_{yz}Rotate_xRotate_y$$

$$Rotate_zTranslate(t_x, t_y, t_z)Perspective(p_x, p_y, p_z, p_w)$$

The algorithm [15] works by undoing the transformation sequence in reverse order. It first determines the perspective elements that, when removed from the matrix, will leave the last column as $(0, 0, 0, 1)^T$. Then extracting the translations leaves the 3×3 transformation matrix comprising of scales, shears and rotations. It is decomposed from left, extracting first the scaling factors and then the shearing components, leaving a pure rotation matrix. This is broken down into three consecutive rotations.

For extracting the perspective components, we need to solve the matrix equation:

$$\begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} & M_{1,4} \\ M_{2,1} & M_{2,2} & M_{2,3} & M_{2,4} \\ M_{3,1} & M_{3,2} & M_{3,3} & M_{3,4} \\ M_{4,1} & M_{4,2} & M_{4,3} & M_{4,4} \end{bmatrix} = \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} & 0 \\ M_{2,1} & M_{2,2} & M_{2,3} & 0 \\ M_{3,1} & M_{3,2} & M_{3,3} & 0 \\ M_{4,1} & M_{4,2} & M_{4,3} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & p_w \end{bmatrix},$$

reduces to:

$$\begin{bmatrix} M_{1,4} \\ M_{2,4} \\ M_{3,4} \\ M_{4,4} \end{bmatrix} = \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} & 0 \\ M_{2,1} & M_{2,2} & M_{2,3} & 0 \\ M_{3,1} & M_{3,2} & M_{3,3} & 0 \\ M_{4,1} & M_{4,2} & M_{4,3} & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w \end{bmatrix}.$$

Assuming that upper left 3×3 partition of M is not singular, this can be solved easily for p_x, p_y, p_z and p_w. The next step is to extract the translations. We find $t_x = M_{4,1}$, $t_y = M_{4,2}$, $t_z = M_{4,3}$. At this point, we are left with a 3×3 matrix (M').

The process of finding the shear and the scale factors is interleaved. We first find, $s_x = |M'_1|$. Then we compute the initial value for xy shear factor $s_{xy} = |M'_1 M'_2|$. The second row of the matrix is made orthogonal to the first by setting $M'_2 = M'_2 - s_{xy} M'_1$.

Then, the y scaling factor, s_y, is the length of the modified second row. The second row is normalized and s_{xy} is divided by s_y to gets its final value. The xy and yz shear factors are computed, the third row is made orthogonal to first two rows, the z scaling factor is computed, the third row is normalized and xz and yz shear factors are rescaled.

The resulting matrix is a pure rotation matrix. Finally we need to decompose the rotation matrix into a sequence of rotations about x, y and z axes. If the rotation angle about x is α, about y is β and about z is γ, then the composite rotation is:

$$R = \begin{bmatrix} cos(\beta)cos(\gamma) & cos(\beta)sin(\gamma) & -sin(\beta) \\ sin(\alpha)sin(\beta)cos(\gamma) - cos(\alpha)sin(\gamma) & sin(\alpha)sin(\beta)cos(\gamma) + cos(\alpha)sin(\gamma) & sin(\alpha)cos(\beta) \\ cos(\alpha)sin(\beta)cos(\gamma) + sin(\alpha)sin(\gamma) & cos(\alpha)sin(\beta)sin(\gamma) - sin(\alpha)sin(\gamma) & cos(\alpha)cos(\beta) \end{bmatrix},$$

where, $\beta = arcsin(-R_{1,3})$

If $cos(\beta) \neq 0$, α is derived easily from $R_{2,3}$ and $R_{3,3}$ and γ from $R_{1,2}$ and $R_{1,1}$.

If $cos(\beta) = 0$, then R reduces to:

$$\begin{bmatrix} 0 & 0 & \pm 1 \\ sin(\alpha \pm \gamma) & cos(\alpha \pm \gamma) & 0 \\ cos(\alpha \pm \gamma) & -sin(\alpha \pm \gamma) & 0 \end{bmatrix}.$$

In this case, we can arbitrarily set γ to 0 and derive α from $R_{2,1}$ and $R_{2,2}$. Decomposition process is now complete.

5.5. Results

The ***Test Case*** consists of a volume data file, *smoothhead.vox* and another volume data file, *smoothrotate.vox* which is *smoothhead.vox* rotated by 45 degrees across Y-axis. The original and the transformed points which are the mean of the clusters from the *clustering stage* are considered. Then for all the possible permutations of the transformed set of points, we compute the transformation matrix. *Sum of squared error* determines the *optimal transformation matrix* and the optimal ordering on the transformed set of points. After we get the optimal transformation matrix, we decompose it into rotation, scaling, shear and translation components. In this case, we got the transformation matrix consisting of negligible translation, scale and shear and 44.2 degrees rotation about Y-axis.

The original points are:

P1 = [-0.285282 -0.160508 -0.497274]

P2 = [-0.092172 0.1162570 0.3811950]

P3 = [0.2886090 0.0456690 0.0122830]

P4 = [0.5312800 0.4082460 0.0192060]

P5 = [-0.752663 -0.057897 -0.069013]

P6 = [-0.102362 0.7322830 0.0669290]

P7 = [-0.106299 -0.183071 0.3562990]

P8 = [-0.336333 -0.141357 0.4488190]

The transformed points are:

P1' = [0.3877740 0.4083900 -0.364199]

P2' = [-0.330709 -0.116142 -0.202756]

P3' = [0.2101630 0.0494360 -0.218118]

P4' = [-0.552056 -0.160105 -0.149606]

P5' = [-0.078277 0.7239460 0.1551640]

P6' = [0.1968500 -0.188976 0.3438320]

P7' = [-0.579527 -0.064567 0.4803150]

P8' = [0.0854030 -0.144760 0.5542090]

The least error = 0.001026

The optimal transformation matrix is:

$$\begin{bmatrix} 0.734140 & -0.071126 & 0.716802 & 0.006759 \\ 0.008998 & 0.992474 & -0.005515 & -0.001075 \\ -0.719109 & 0.030140 & 0.715749 & 0.002124 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Decomposition of transformation matrix yields:

tranlate = 0.006759, -0.001075, 0.002124 ≈ (0.,0.,0.)

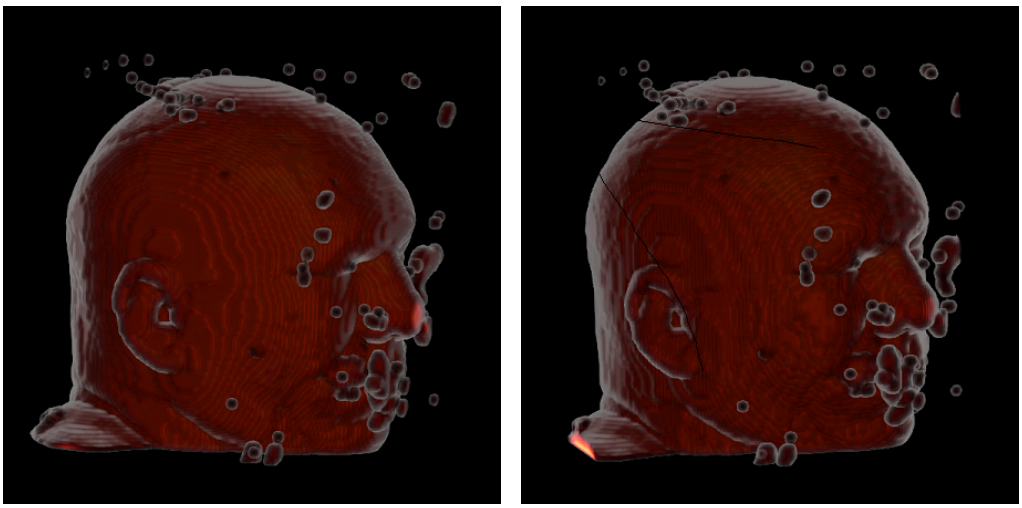
scale = 1.027696, 0.993467, 1.012336 ≈ (1.,1.,1,)

shear = -0.063625, 0.011036, -0.033983 ≈ (0.,0.,0.)

rotate = -0.019852, 0.775018, 0.012256 ≈ (45.,0.1.,0.)

6. RESULTS

Test Case: We have a volume data set "smoothhead.vox" and another volume data set "smoothrotate.vox", which is smoothhead rotated by 45 degrees about the Y-axes. Given just these volumes, we wanted to see if these methods are able to determine the transformation matrix that transforms smoothhead.vox to smoothrotate.vox (i.e. transformation matrix should have only rotation term, 45 degrees about Y axes, with no translation, shear and scaling factor). These two volumes are shown in Figure 6.1.

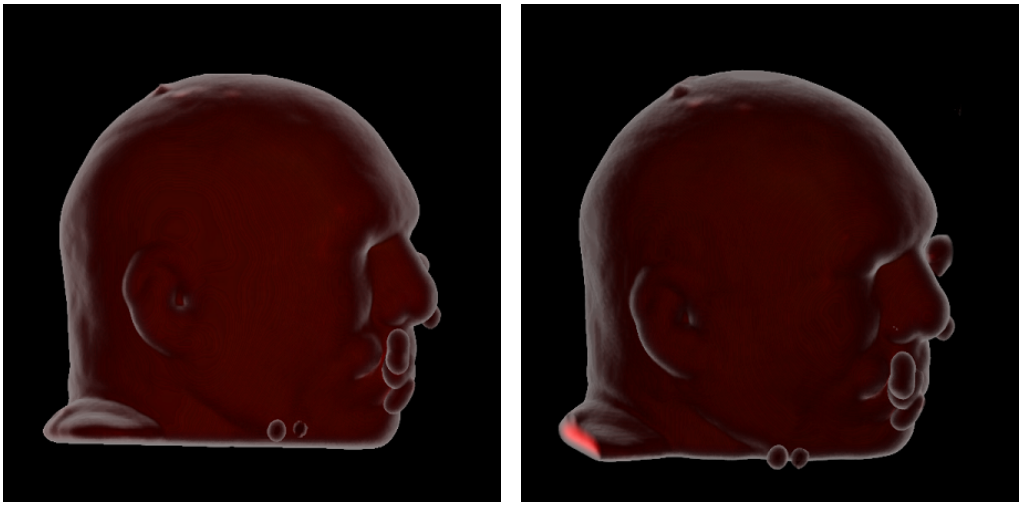


(a) smallhead.vox (b) smoothrotate.vox

FIGURE 6.1. Volume data sets (rotated) for matching

The first step in "volume-volume matching" is smoothing. Raw volumes are often degraded by noise and hence greatly affect the volume matching process. We apply ***Gaussian smoothing*** to smooth the volumes and remove the noise. The results of smoothing are shown in Figure 6.2.

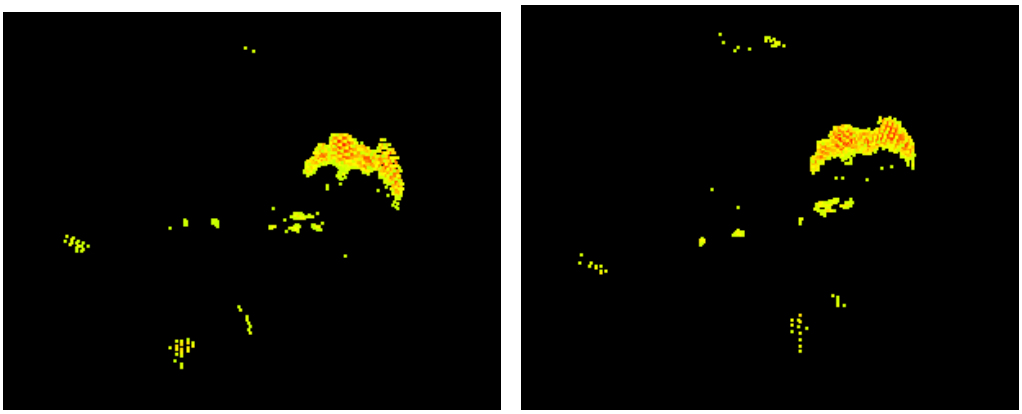
The second step in volume-volume matching is to identify the important subsets of the volume data to perform reliable matching of volumes. This is achieved by considering the maximum eigenvalues of the Hessian matrix as the



(a) smallhead.vox (b) smoothrotate.vox

FIGURE 6.2. Gaussian smoothing applied to volume data sets (rotated)

threshold value as it determines where data values are changing rapidly. The distinctive invariant voxels (***hotspots***) are shown in Figure 6.3.



(a) smallhead.vox (b) smoothrotate.vox

FIGURE 6.3. Identifying hotspots in volume data (rotated)

The combined hotspots for smoothhead.vox and smoothrotate.vox are be shown together in Figure 6.4. The hotspots in smoothrotate.vox are rotated by -45

degrees and then displayed. The figure clearly shows that the hotspots detected in two volumes are indeed rotationally invariant.

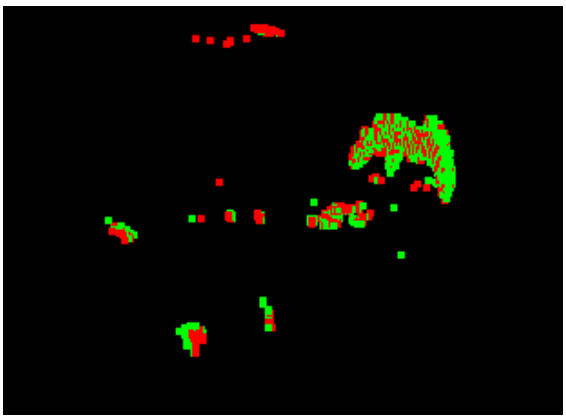
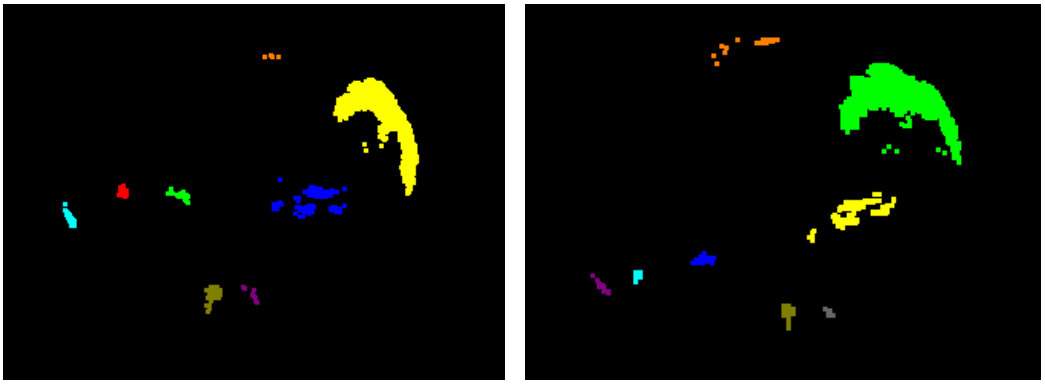


FIGURE 6.4. Combined hotspots displayed in green for smoothhead.vox and red for smoothrotate.vox

After detecting the hotspots in volume data sets, we perform clustering to reduce the number of voxels to a few so that it is feasible to consider all possible combinations of voxels between the two sets. We use ***single-linkage hierarchical clustering*** where the distance between the clusters is defined as distance between the closest pair of points in a cluster. The results of this clustering is shown in Figure 6.5.

After clustering the points into clusters, we take the mean value of these clusters. We then apply ***least squared transformation*** to mean points both the clusters to determine the optimal transformation matrix that transformed the first volume data set to the second. The optimal transformation matrix with least sum of squared error is:



(a) smallhead.vox (b) smoothrotate.vox

FIGURE 6.5. Identifying clusters in volume data (rotated) - 8 clusters in 8 different colors.

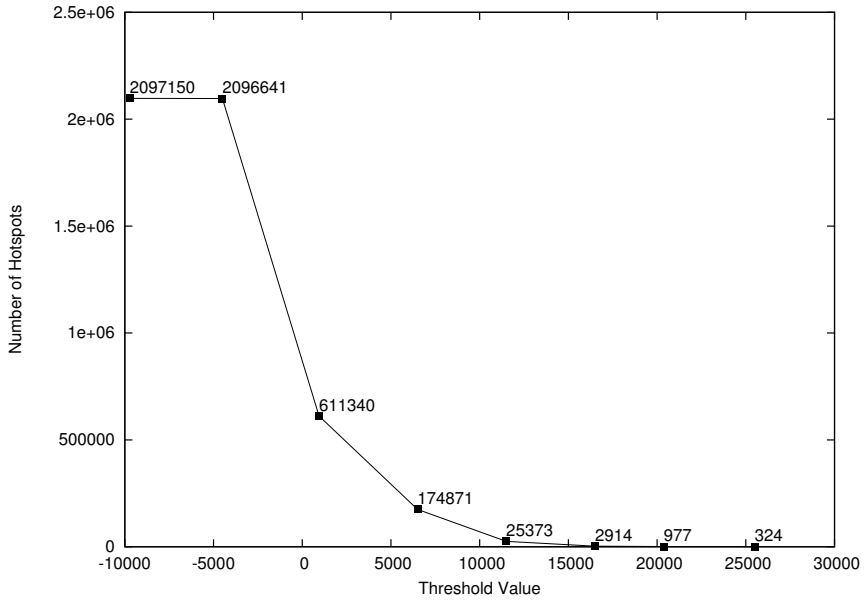
$$\begin{bmatrix} 0.734140 & -0.071126 & 0.716802 & 0.006759 \\ 0.008998 & 0.992474 & -0.005515 & -0.001075 \\ -0.719109 & 0.030140 & 0.715749 & 0.002124 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}.$$

The number of hotspots identified should be *optimal*, neither "too few" nor "too many". If they are too few, then some important details in the volume might be missed and if they are too many, they might over represent the volume data and could create problems in clustering. The maximum eigenvalue at each voxel is taken as a threshold value. Figure 6.6 shows the plot of *Number of Hotspots* and *Threshold Value* for "smoothhead.vox" and "smoothrotate.vox" volume data sets. After thresholding the hotspots, they are clustered and optimal transformation matrix is determined for the two volume data sets. Figure 6.7 shows a plot of the *Number of Hotspots* and *Error*. After clustering, the mean of the clusters are considered for determining the optimal transformation. The number of clusters

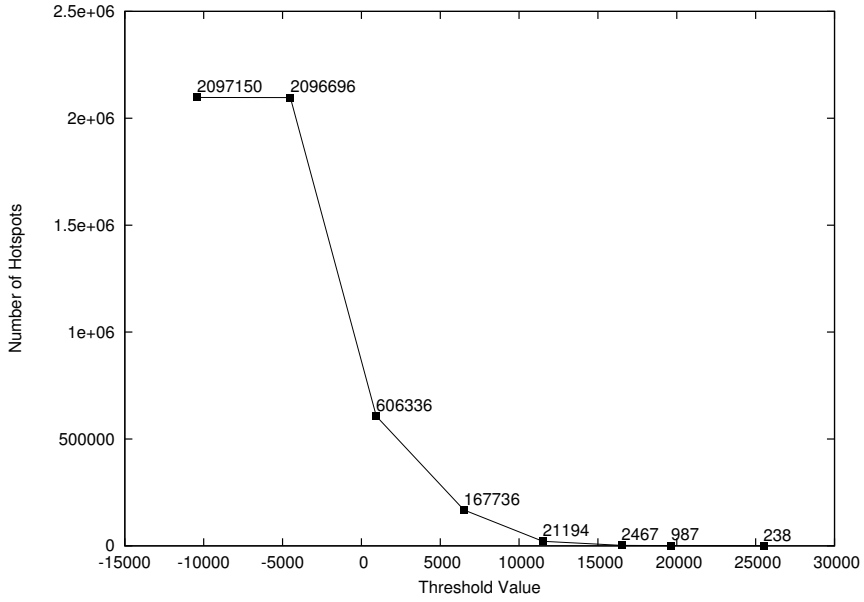
should also be *optimal*, because for determining the optimal transformation and optimal ordering on the points, we consider all the possible combinations of the points in the two volume data sets. Hence, the number of clusters should be such that it takes minimal time and gives minimal error while determining the transformation matrix. Table 6.1 and Figure 6.8 shows a plot of *Number of Clusters* vs *Error* and *Number of Clusters* vs *Time (in sec.)*. The data was taken on Intel[R] CPU with processor speed of 1.83GHz and 1.00GB of RAM.

Number of Clusters	N!	Error	Time(in sec.)
4	24	0.001123	0
6	720	0.000594	0
8	40320	0.001026	8
10	3628800	0.001324	1009
12	479001600	0.002103	3706

TABLE 6.1. Data for plot of "Number of clusters", "Error" and "Time"



(a) smoothhead.vox



(b) smoothrotate.vox

FIGURE 6.6. Plot of “Number of Hotspots” vs “Threshold Value”

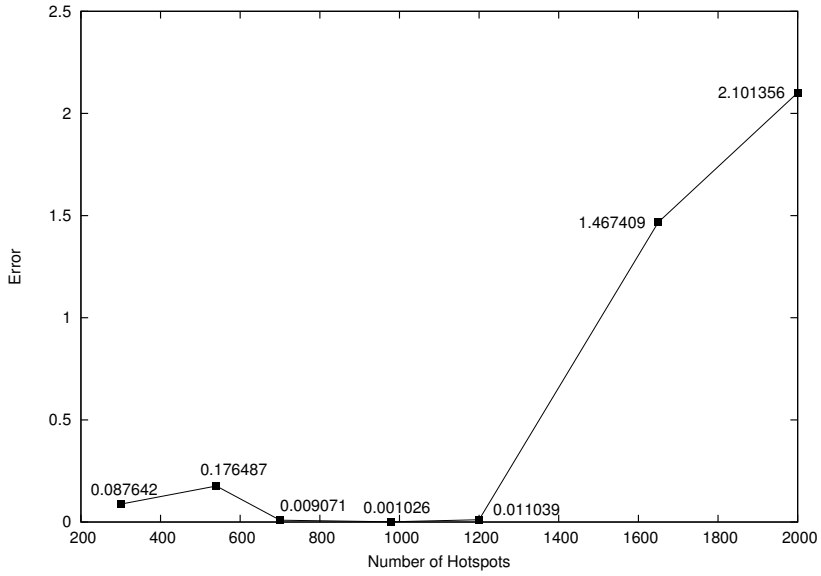


FIGURE 6.7. Plot of "Number of Hotspots" vs "Error"

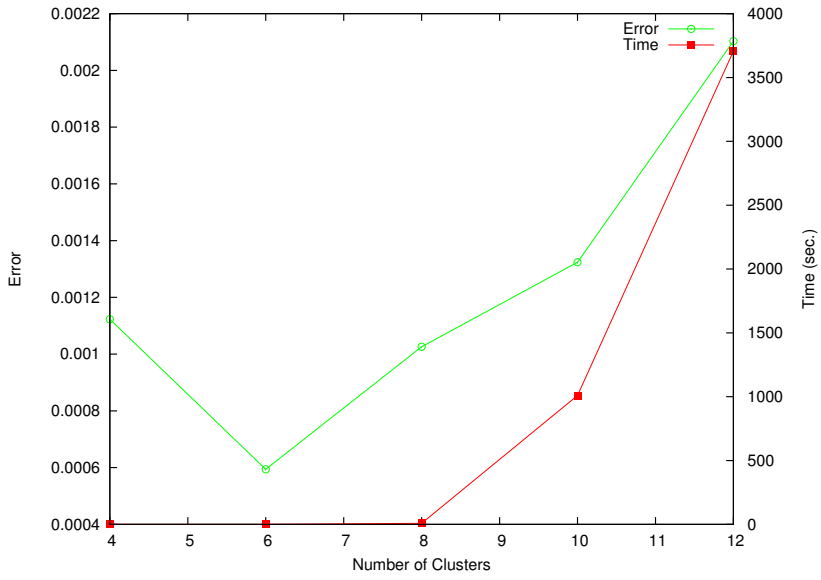


FIGURE 6.8. Plot of "Number of Clusters" vs "Error" and "Number of Clusters" vs "Time"

We also tried to find the optimal transformation matrix considering ***two translated volumes***. The first volume data set is "smoothhead.vox" and the second volume data set is "smoothtranslate.vox", which is head data translated by 0.16 units along Z-axes. The original volume lies between the range -1 and 1. We translate the important information inside the volume by 0.16 units along Z-axes so that the translated volume still is in the same range -1 and 1.

The first step in matching these two volumes is to filter the volumes using ***Gaussian smoothing***. The result of Gaussian smoothing on smoothtranslate.vox can be shown in Figure 6.9.

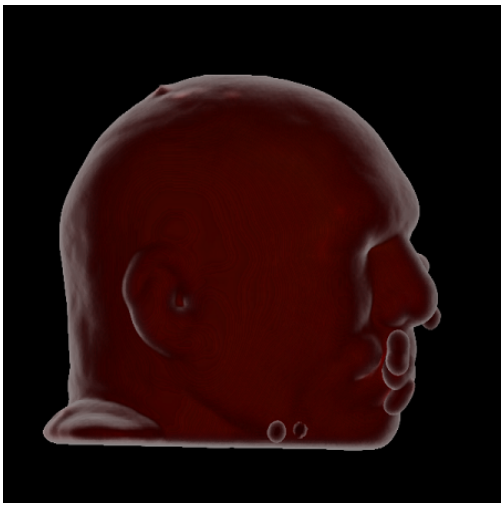
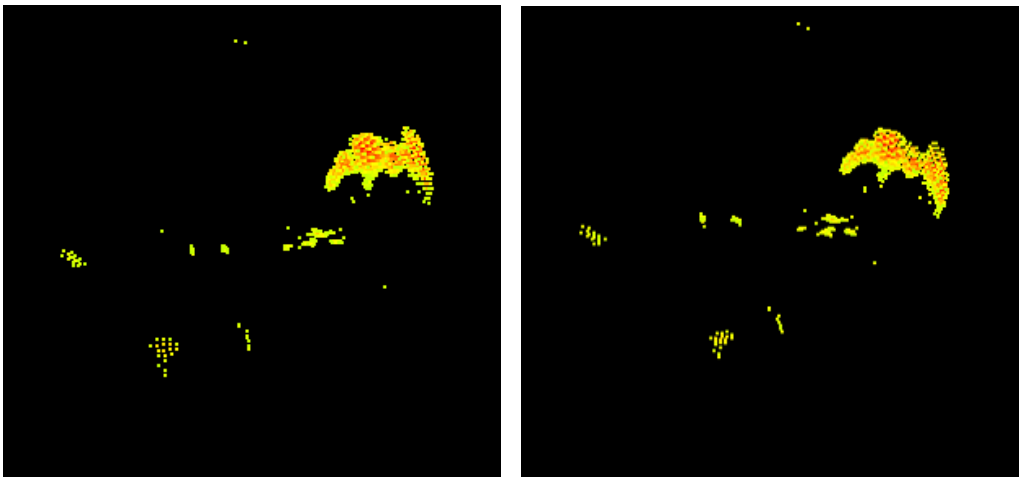


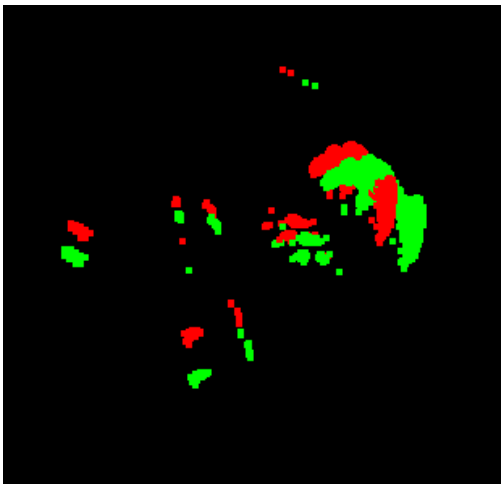
FIGURE 6.9. Gaussian smoothing applied to smoothtranslate.vox (translated head)

The second step i.e. ***to identify the important subsets*** of volume data set (hotspots) is shown in Figure 6.10.

After identifying the important subsets, we still need to reduce the number of points; hence we applied ***hierarchical clustering*** to achieve the same. The results of clustering for the two volumes - original and translated volume can be shown in Figure 6.11.

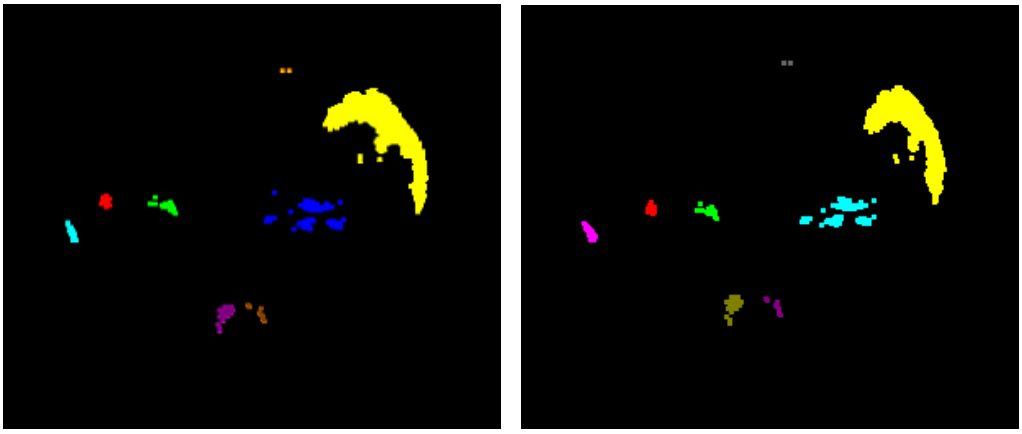


(a) smoothhead.vox (b) smoothtranslate.vox



(c) Combined Hotspots

FIGURE 6.10. Identifying hotspots in volume data (translated) -(a) shows hotspots in smoothhead.vox, (b) shows hotpots in smoothtranslate.vox and (c) shows combined hotspots (red for smoothhead and green for translated head)



(a) smoothhead.vox (b) smoothtranslate.vox

FIGURE 6.11. Hierarchical clustering applied to volume data (translated) - 8 clusters are shown in 8 different colors

We take mean of these clusters and determine the optimal transformation matrix and optimal ordering on these points using the ***least squared method***. The resultant transformation matrix is:

$$\begin{bmatrix} 0.999294 & 0.000379 & 0 & -0.000291 \\ 0.000167 & 0.999910 & 0 & 0.000069 \\ -0.000992 & 0.000531 & 1 & -0.157889 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Splitting this transformation matrix into individual transformations we can see that we get translation of ~ 0.16 about Z-axes with negligible scaling, shear and rotation components.

tranlate = -0.000291, 0.000069, -0.157889 $\approx$ (0.,0.,-0.16)

scale = 0.999294, 0.999910, 1.0 $\approx$ (1.,1.,1.)

shear = 0.000546, -0.000992, 0.000532 $\approx$ (0.,0.,0.)

rotate = 0.000532, 0.000993, 0.000167 $\approx$ (0.,0.,0.,0.)

7. CONCLUSION AND FUTURE WORK

This approach can be used effectively to compare two volumes without having to compare all the voxels in one volume to all the voxels in other volume set. In *chapter* 3, we presented the various filtering strategies to reduce noise from raw volume data and introduced an approach to identify volume-data features (hotspots) based on eigenvalues of the Hessian matrix. In *chapter* 4, we presented the various clustering methods applied to the hotspots to extract the vital features or further reduce the number of voxels under consideration. In *chapter* 5, we determine the optimal ordering on the points and the optimal transformation matrix that transforms one volume data set to the other. Though the volume-volume matching method involves a number of steps, none of these steps is very time consuming.

As *chapters* 3-5 deal with distinct topics, we concluded them separately. Nevertheless, more general conclusions which are based on acquired experience during the development of concepts and implementation can be pointed out as:

- The information in $3D$ volumes is carried by 9 distinct numbers, i.e. 3 gradient components plus 6 elements from the Hessian matrix.
- The barrier which creates difficulties in extending analyses from $2D$ to higher dimensions resides is additional degree of freedom in choosing the direction to be examined.
- Computation of eigenvalues of real symmetric matrices is relatively cheap compared with general matrices. This fact has to be considered during implementation.

- Hierarchical clustering (agglomerative) is better than other clustering strategies (partitional) in the case of raw volume data as it is more robust to outliers and creates an ordering of points that might be useful.

- When no correspondence is known between the two set of points, the optimal transformation matrix and optimal ordering on points is determined by the sum of squared error metric. Sum of squared error metric is computed between the transformation matrix determined by least square approach and the new transformation matrix determined by multiplying the original set of points by the transformation matrix.

In order to make this methodology perfect to match two volumes, we still need to figure out some steps but we keep them as future work. Some of these steps are:

- Do the computation on *parallel processors* to achieve better timing performance. For instance, the computation of the eigenvalues for each voxel can be done in parallel and then results can be combined together and thresholding of hotspots can be done.

- A method to define $3D$ *volume descriptors*, so that we know the correspondence between the voxels in one volume data set and the other. In this case, we need not consider all the combinations of voxels (mean of the clusters) in one volume set and the other.

- A *standard metric* to threshold the number of hotspots and number of clusters.

- Try volume-volume matching algorithm on similar, but not identical volumes.

BIBLIOGRAPHY

[1] Wikipedia. http://en.wikipedia.org/wiki/Hessian_matrix.

[2] Agglomerative methods.
http://www.resample.com/xlminer/help/HClst/HClst_intro.htm.

[3] Box plots. http://en.wikipedia.org/wiki/Interquartile_range.

[4] Data clustering. http://en.wikipedia.org/wiki/Data_clustering.

[5] Drawbacks of k-means clustering algorithm.
http://people.revoledu.com/kardi/tutorial/kMean/index.html.

[6] Gaussian and salt and pepper noise.
http://en.wikipedia.org/wiki/Image_noise.

[7] Gaussian smoothing.
http://www.homepages.informatics.ed.ac.uk/rbf/HIPR2/gsmooth.htm.

[8] Least square plane. http://www.infogoaround.org/JBook/LSQ_Plane.html.

[9] Mean filter. http://www.cee.hw.ac.uk/hipr/html/mean.html.

[10] Median filter. http://www.cee.hw.ac.uk/hipr/html/median.html.

[11] Outliers. http://en.wikipedia.org/wiki/Outlier.

[12] Permutations.
www.codeguru.com/Cpp/algorithms/combinations/atricle.php/c7605.

[13] Single linkage clustering.
http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/hierarchical.html.

[14] Volume rendering methods. http://en.wikipedia.org/wiki/Volume_rendering.

[15] Arvo, J. *Graphics Gems II*. Morgan Kaufmann, 1994.

[16] Barnett, V., and Lewis, T. *Outliers in Statistical Data*. John Wiley & Sons, 1994.

[17] Gerstner, T. Fast multiresolution extraction of multiple transparent isosurfaces. In *Data Visualization 2001, Proceedings of the Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization* (2001), pp. 35–44, 336.

[18] Gonzalez, R. C. *Digital Image Processing*. Prentice Hall; 2nd edition, 2002.

[19] Heckbert, P. Color image quantization for frame buffer display. *Computer Graphics* (1982).

[20] Hladuvka, J. *Derivatives and Eigensystems for Volume Data Analysis and Visualization*. PhD thesis, Vienna University of Technology, 2001.

[21] Hladuvka, J., Konig, A., and Groller, E. Exploiting eigenvalues of the hessian matrix for volume decimation. Tech. rep., Institute of Computer Graphics and Algorithms, Vienna University of Technology, October 2000.

[22] Horn, R. A., and Johnson, C. R. *Matrix Analysis*. Cambridge University Press, 1985.

[23] Johnasson, J., Ljung, P., Jern, M., and Cooper, M. Revealing structure within clustered parallel coordinates display. In *IEEE Symposium of Information Visualization, 2005* (October 2005).

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

[25] Kull, M. Fast clustering in metric space. Master's thesis, University of Tartu, 1999.

[26] Lorensen, W. E., and Cline, H. E. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of ACM SIGGRAPH* (1987), pp. 163–170.

[27] Lowe, D. Scale invariant feature points. *International Journal of Computer Vision 60*, 2 (2004), 91–110.

[28] Mroz, L., Hauser, H., and Groller, E. Interactive high-quality maximum intensity projection. In *Proceedings of EUROGRAPHICS* (2000), pp. 341–350.

[29] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. *Numerical recipes in C:Art of Scientific Computing*. Cambridge University Press, 1992.

[30] Saito, T. Real-time previewing for volume visualization. In *Proceedings of IEEE Symposium on Volume Visualization*.

[31] Salomon, D. *Digital Image Processing*. Springer Verlag, 1999.

[32] Shapiro, L. G., and Stockman, G. C. *Computer Vision*. Prentice Hall, 2001.

[33] TeraRecon, Inc. *Voxel File Format Specification*, August 2001. http://www.terarecon.com/downloads/support/vp500_vol_file_format.pdf.

APPENDICES

APPENDIX A. Volume Data File Format

*Vox*1999*a* is a simple, flexible format for voxel data storage, exchange and input and output filter targeting. The file format is operating system and processor independent. It is capable of storing 1, 8, 16, 32 and 64 bit volumes, each of which can contain multiple fields. This format also enables to store multiple volumes in a single file. Volumes stored in this format are assumed to be consisting of slices of voxels with equal distance between the slices.

The voxel file format consists of three logical sections. The first one is the *file header* that is used to identify the file as being the proposed voxel file format. The header is followed by a *volume description* that describes the first volume and voxel data in it. The *volume data* section follows the volume description section. Each datum in the data section represents a voxel containing one or more fields. The file header and the volume description sections contain text in ASCII encoding, organized as *descriptors*. The parameter types used in descriptors are int, float, word, rest-of-line and quoted-string.

A.1. File Header

- *Signature - required:* The signature (Vox1999a) must appear at the beginning of the file (bytes 0-7) and must be immediately followed by end of line.

- *Comments - optional:* Comments are introduced by // and terminated by the end of line and can appear anywhere in the file header after the signature.

- *Volume Count - optional:* This descriptor (VolumeCount int) specifies the number of volumes stored in the file. If it is 0 then an arbitrary number of volumes are stored in the file. The last volume is followed by the end of file.

- *Title - optional:* This descriptor (Title rest-of-line) is used to hold the descriptions of the overall data, such as the origin of the data, date it was acquired/generated, etc.

- *Copyright Notice - optional:* This descriptor (Copyright rest-of-line) is used to place a copyright notice.

- *Attribute - optional:* This descriptor (Attribute word rest-of-line) is available for application specific purposes.

- *Data - optional:* This descriptor (Data word int) is available for application specific purposes, for binary data related to the entire file. The integer argument represents the size, in bytes, of the data, which must appear after the end-of-section marker.

- *End of Section - required:* This sequence ($\#\#\backslash f$) signals the end of the file header.

A.2. Volume Description

- *Start of Section - required:* This sequence signals the start of the volume description. It must appear on a single line with no leading spaces.

- *Volume Size - required:* This descriptor (VolumeSize int int int) describes the size of the volume. The three arguments are the number of voxels in the x, y, and z dimensions of the volume.

- *Voxel Size - required:* This descriptor (VoxelSize int) specifies the size of voxel data stored in the volume data section immediately following the descriptor block.

- *Endian - required:* The argument (Endian word) must be either L or B to indicate little-endian and big-endian modes, respectively.

- *Volume Scale - optional:* This descriptor (VolumeScale float float float) describes the scale of the three axis. The three floating-point numbers represent the voxel spacing between adjacent voxels in each direction.

- *Volume Position - optional:* This descriptor (VolumePosition float float float) describes the position of the volume.

- *Voxel Field - Field0 required and remaining optional:* The first line of this descriptor must include Field and the field number. The name-value pairs inside the parentheses can appear in any order, and the end of line indicator can appear anywhere that blank space is acceptable. The end of line indicator must follow the close parenthesis.

- *Model matrix - optional:* This descriptor specifies a 4x4 model matrix to describe how the voxel center positions map to a corrected coordinate space that is rectilinear and scaled equally in all three dimensions.

- *Attribute - optional:* This descriptor (Attribute word rest-of-line) is available for application specific purposes, for textual (ASCII) data related to this volume.

- *Additional Data - optional:* This descriptor (Data word int) specifies additional data that is to follow the voxel data.

- *Comments - optional*
- *Title - optional*
- *Copyright - optional*
- *End of Section - required*

A.3. Volume Data

- *Voxel Data:* With the information in the data descriptor block, the total size, in bytes, of the volume can be computed as follows:

$$TotalBytes = Floor((Nx \times Ny \times Nz \times VoxelSize + 7)/8)$$

- *Additional Data:* Additional data, if any, follows immediately after the voxel data.
- *End of Data:* The end of the volume data section is determined solely by the voxel data size and the sizes of all Data descriptors in the volume description.

A simple example of a vox file header and descriptor is:

Vox1999a

\ *f*

##

VolumeSize 128 128 128

VoxelSize 16

Endian *B*

Field 0 (Position 0 Size 16 Name Test_Data)

\ *f*