2011

Development and Testing of a Fractal Analysis Algorithm for Face Recognition

Nicholas J. Hansford
West Virginia University

Development and Testing of a Fractal Analysis Algorithm for Face Recognition

By
Nicholas J. Hansford

Dissertation submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Mechanical Engineering

Dr. W. Scott Wayne, Chair
Dr. Alfred Stiller, Co-chair
Dr. Mary Ann Clarke
Dr. David Walker
Dr. John Kuhlman

Department of Mechanical and Aerospace Engineering
Morgantown, West Virginia
2011

Keywords: Face Recognition Algorithms, Face Identification, Random Walk, Arc-Fraction Circle Method, Fractals, FERET Database, Pattern Recognition

Abstract

Development and Testing of a Fractal Analysis Algorithm for Face Recognition
Nicholas J. Hansford

Following an earlier development for fingerprints by Deal (1) and Stoffa (2), it was suggested that this algorithm may work on faces (or more precisely, face images). First, this work transformed a 2-D electronic image file of a human face into a numeric system via a similar random walk process by Deal and Stoffa. Second, the numeric system was analyzed, and the numeric system may then be tested against a database of similarly converted images. The testing determined whether the subject of the image is part of the database. Finally, the efficiency, quickness, and accuracy of such an algorithm were tested and conclusions about the general effectiveness were made.

The algorithm employed a Random Walk analysis of digital photographs of human faces for a fixed number of binary images which were generated from the source photograph using a Boolean conversion scheme. The Random Walk generated a series of transition probabilities for a particular scale. In short, the numeric system used to describe the face will consisted of two dimensions of data—scale and binary image. The numeric system for a particular source photograph was tested against a database of similarly constructed systems to determine whether the subject of the source photograph was in the database.

For the purpose of this work, a database of 400 images was constructed from 167 individual subjects using the FERET database. The 400 images where then analyzed, and tested against the database to determine whether the algorithm could "find" the subjects in the database. The algorithm was able, in its best configuration, to identify correctly the subjects of 168 of the

400 photographs. However, the total time to run an image (after capture by a digital camera) to database comparison was only 62 seconds, which represents a substantial improvement over previous systems.

Dedications

I would like to dedicate this work to Dr. Stiller who saw something in that freshmen engineering student goofing around with C code in class. I would never have achieved so much without your support as both a friend and professor. I also dedicate this work to my friends and family who helped me throughout my doctoral experience.

Acknowledgements

I would like to acknowledge the people who have helped during the long process of earning a Ph.D. First, my fiancée, Meagan Hubbell, who has endured countless hours of my frustration throughout the writing process. She has been a tremendous support during the entire process. I can safely say that without her help and understanding, I would have easily gone crazy long before now. I cannot thank her enough for all has done and contines to do.

I would like to thank Pacific Northwest National Labs for providing the fudning for this project. Without their support, this face recognition concept would still be relegated to a paste-time of Dr. Stiller and myself. It is unlikely we would have ever spent the time required to bring it to this level. I would also like to acknowledge our research group: Dr. Stiller, Jeremy Hardinger, and Mark Martinez, who were all truly helpful during the code development phase of this work.

Finally, I want my committee to know that I appreciate all they have done for me throughout this process. They have been insperiational in both my choice to persue a doctorate at WVU and my academic experience while there.

Thank you all very much.

Table of Contents

List of Figures

List of Tables

List of Symbols

A	unknown mixing matrix
AFC	Arc-Fraction Circle
B	desired number of classes (PCA), blue band
C	LDA class
D	database matrix
E	function of interest
G	training set of images (PCA), non-quadratic function (ICA), green band
H	Entropy
I	particular image
ICA	Independent Component Anlysis
J	jet (Garbor wavelet)
LDA	Linear Discriminant Anlysis
M	count of integers, number of images in database
N	total number of pixels in an image, normalized determinant
P	matrix of transition probabilities, polynomial
PCA	Principle Component Anlysis
PD	percent difference
R	rotation matrix, red band
S	scatter matrix (ICA), distance otherwise
U	width of bars in vertical bar pattern
W	decoupling matrix
X	Image matrix in PCA
$\bar{X}$	average of all images
$\tilde{X}$	difference of $\bar{X}$ and a particular image
a	coefficient, array of coefficients
b	degree of polynomial
c	constant, detent for converting to binary from non-binary
d	fractal dimension, offset from vertical bar pattern, distance
f	probability distribution function
i	integer, the imaginary number
j	Integer
k	vector in image
l	scale (length of the line segment)
m	number of horizontal pixels in image, number of scales
n	number of vertical pixels in image, number of data points (least squares)

p	number of images in training set (PCA), number of binary detents/images
q	least squares difference
r	fractal pattern ratio
s	unknown vector, slope
t	dummy variable
u	solution to decoupled system (PCA), otherwise unit vector
x	vector in class C (LDA), otherwise normal Cartesian coordinate
x'	modified Cartesian coordinate
y	Cartesian coordinate
y'	modified Cartesian coordinate
Λ	matrix of Eigen values
Ω	discrete domain
$\bar{\Omega}$	average of discrete domain
Φ	count of instances of a particular pixel combination
Θ	angle
α	transition probability
β	transition probability of mixed combinations (when they are equal)
γ	parameter
λ	differentiating parameter (LDA)
μ	maximum allowable percent difference between coefficients
ν	ratio of project axis (ICA), mimimun percent of matches to predict image subject match
φ	plane wave shape vectors
π	3.1415926…
σ	phase shift, standard deviation
Subscripts	
F	relating to a fractal
L	left-hand side
I	relating to an electronic image
R	right-hand side
S	denotes subject's face region
b	first scatter direction
bb	black-black pixel pairing on line segment
bw	black-white pixel pairing on line segment

i	counting index
j	counting index
k	counting index
p	related to the pupil
w	second scatter direction
wb	white-black pixel pairing on line segment
ww	white-white pixel pairing on line segment
Superscripts	
c	related to binary image direction
i	counting index
k	counting index
l	related to scale direction

1. Introduction

This section will introduce the problem which has been studied and documented in this dissertation. The basic goals of the research will be formally stated, and the history of the problem will be discussed. Finally, the overall organization and structure of the document itself will be addressed.

1.1. Problem Statement

Following an earlier development for fingerprints (1) by Deal and an unsuccessful attempt to prove the Random Walk algorithm's effectiveness for the characterization and indentification of fingerprints (2) (Stoffa), it was suggested that the algorithm may work on faces (or more precisely, face images). First, this work transformed a 2-D electronic image file of a human face into a numeric system via a similar random walk process to Deal and Stoffa. Second, the numeric system was analyzed, and then be tested against a database of similarly converted images. The testing (ideally) determined whether the subject of the image is part of the database. Finally, the effectiveness, quickness, and accuracy of such an algorithm was tested so conclusions about the general effectiveness can be made. In parallel to this effort was the development of a Graphical User Interface (GUI) for minimally-trained users.

1.2. Research Goals

There are five specific goals of this work. They are enumerated below:

1. Adapt the fingerprint Random Walk algorithm (1) for digital photographs of faces. (While this work is specific to faces, the resulting approach can be applied to any color digital photograph such as pictures of deformed bodies, quality control checkpoints, crack detection or propagation, or fluid flow analysis.) It

was hypothesized that this can be done by creating fractal images of a series of binary images.

2. Develop a means of normalizing the data presented from Goal 1 so that any face can be compared to another face without the interference of differing numbers of black and white pixels.
3. Develop a rigorous method to explain the patterns that are produced as a result of the random walk analysis.
4. Quantify the algorithm's effectiveness in comparing faces to a database of faces in both accuracy and quickness following a parametric study to determine the optimal configuration of the algorithm.
5. Finally, the algorithm was adapted to GUI to simplify use.
    a. As part of GUI development, as much of the algorithm should be automated and run-time parameters fixed or locked from user input.
    b. The acceptable level of user input are file (face) selection and eye location as these could easily be performed by a user with minimal training.

The first goal of this work was to modify the Random Walk method for fingerprint binary images (Deal) as required, so that it can be applied to face images. Fingerprints are very simple to capture digitally because they are comprised of either digital photographs of fingerprints or infrared scans. Generally as captured, fingerprint images are not binary (a requirement of Deal's algorithm); however binary conversion of fingerprint images is quite easy (2) as fingerprints essentially are binary (ridges or valleys). Face images are much more difficult to convert to binary images. While it is possible to apply the binary conversion scheme as used by Stoffa, it removes too much of the facial curvature and detail, preventing a valid analysis for comparison.

A new method of binary conversion, which does not remove facial curvature, was developed. This method is global, not local, and operates on a single color band (be it red, green, blue, or gray-scale) and uses the average pixel value and standard deviation to develop a series of binary images instead of a single binary image. From this series of images, the Random Walk method was applied to create a system with an additional independent variable which results from the series of binary images.

The second goal of this work was to develop a normalization scheme or parameter to remove the effects of differing numbers of black and white pixels in a binary image. Because there was no control placed on the number of black or white pixels in the binary conversion, the resulting number of either may vary from one image to another. This resulting difference skewed the scale spectra by lowering the probability of either white-white or black-black matches and increasing the probability of the other. So, a normalization scheme which removed this effect was required. The scheme also allowed for the resulting scale-spectrum to be valued between 1 and 0.

The third goal was to develop a rigorous mathematical explanation of the curve features and fractal pattern creations. In doing so, the curves used during the resulting analysis were further be explained. Finally the rigorous solution also demonstrated why it was possible to add or subtract an “average” face from a particular dataset (and the advantages).

The fourth goal of the research was to quantify the resulting algorithm's speed and accuracy using a standardized, repeatable test. There have been numerous attempts to standardize and quantify bio-metric algorithms for faces starting with the creation of the Facial Recognition Technology (FERET) Database in 1993 (3) and the following early facial recognition tests (4). One example is the series of tests conducted for the Facial Recognition

Vendor Tests (FRVT) which began in 2000. These tests are very in-depth and have used proprietary databases of high resolution photographs as recently as 2006 (5).

For the purpose of this work, a much simpler scheme based on the readily available Facial Recognition Technology (FERET) database was proposed. A database of approximately 400 face images of various subjects of varying ethnicities was constructed from the FERET database. These images were full-frontal shots only, had a reasonable pixel resolution to prevent blurriness, and were free from artificial defects such as smudges and adverse lighting effects. In this database each subject was represented at least twice. For the test, the same 400 images were read in and compared to images in the database. Since the number of instances of a given subject was known, the accuracy could then be calculated. Finally, the duration of time required for read-in and comparison was tracked.

The fifth goal of this work was to develop a Graphical User Interface (GUI) such that any reasonably trained person could make use of the system. In doing so, a large portion of the algorithm was automated. While this goal was not strictly related to the research portion, it did anecdotally make a statement about the simplicity of the algorithm.

1.3. Document Structure

The remainder of the document will be devoted to several sections. The first section will document the background of facial recognition. This background will cover a brief history of facial recognition and other two-dimensional methods which specifically utilize digital photographs. The short-comings of these other methods will be addressed. The background section will conclude with a statement of need for the work documented herein. The primary need results from both the high computational time required by other methods and several public failures of the other existing facial recognition technologies. The background will also include a

brief summary of the database of digital photographs to be used for this work and an introduction to digital photographs.

Following the background section, a section covering the development of the basic algorithm as it was initially applied to fingerprints will be given. Then the algorithm was modified to be applicatable to human faces, which are much more complex. During the development of the algorithm, specific components will be documented in detail. Several assessments of the algorithm's accuracy will then be discussed and documented.

Once the algorithm had been configured with the optimal settings, it was stream-lined into a working GUI. The GUI use will be documented after all of the experimental results have been discussed. Finally, the document will conclude with recommendations for future work and any problems encountered during the experimental work.

2. Background

It's a simple question really: how does one person visually indentify another? Unfortunately, there is no clear answer to this question. While researchers are studying the question, both metaphysically and physically-this work focused on specifically developing a computer program which used only images of the human face for peronsal identification. While biometric identification techniques have employed many different featuresn the most popular technologies focus on faces, fingerprints, irises, and DNA (6).

This chapter will open with a discussion of how facial recognition works on the algorithmic level and why a new software system is needed. It will also discuss why facial recognition algorithms for computers are important. It will end with background information on various competing algorithms. These algorithms will be shown to be impractical (either because

they are computationally inefficient or because of the skill level required from the user) for the work which will be performed. As such, a new method was developed which poses to be quicker and more accurate.

2.1. Human Face

The human face is generally considered to be the area at the front of the head which includes the nose, eyes, cheeks, mouth, lips, and curved region towards the temple. A typical face is shown in Figure 2-1. This image also shows the "face" as defined by this work. The image used was trimmed with an ellipse to eliminate the background. Obviously, the face is not a perfect ellipse, but the elliptical shape is a close match for most faces.

Figure 2-1: Example Human Face

Using the face as the metric for identification of individuals presents a more intuitive approach than a fingerprint, iris, or DNA since people generally do not look at other's fingerprints before deciding who someone is. But more to the point, fingerprints, iris scans, and DNA examination require a cooperative subject. Conversely, to take a picture is quite trivial, even if the subject is uncooperative, a series of frames (a movie) would eventually lead to a

useable picture in a matter of seconds. Facial recognition has the added benefit of source image collection in that video surveillance is nearly ubiquitous, whereas systems to collect fingerprints DNA, or iris scans are not.

2.2. Face Recognition

For the purpose of this work, face recognition is the attempt to sample, analyze, and compare one digital photograph of a face to another facial photograph. These photographs may be taken from a digital camera, surveillance system, or some pre-existing database such as the FERET database. In its simplest form, the facial recognition algorithm (even among humans) consists of 4 to 5 steps.

1. Acquire an image of a face (or any biometric feature)-be it electronic, film, movie, or through the eye.
2. Isolate the region or unique features of interest.
3. Using the information from the isolated region, compare the captured image to previously captured images in a database which may be digital, analogue, or inside the human brain.
4. Determine whether the captured image belongs to the database and if so, who it is.
5. (Optional) Verify whether the result is correct. For instance, a person may forget an old friend's face, or a computer system may incorrectly reject someone who has recently begun to wear glasses.

The basic approach is demonstrated graphically (without step 5) in Figure 2-2.

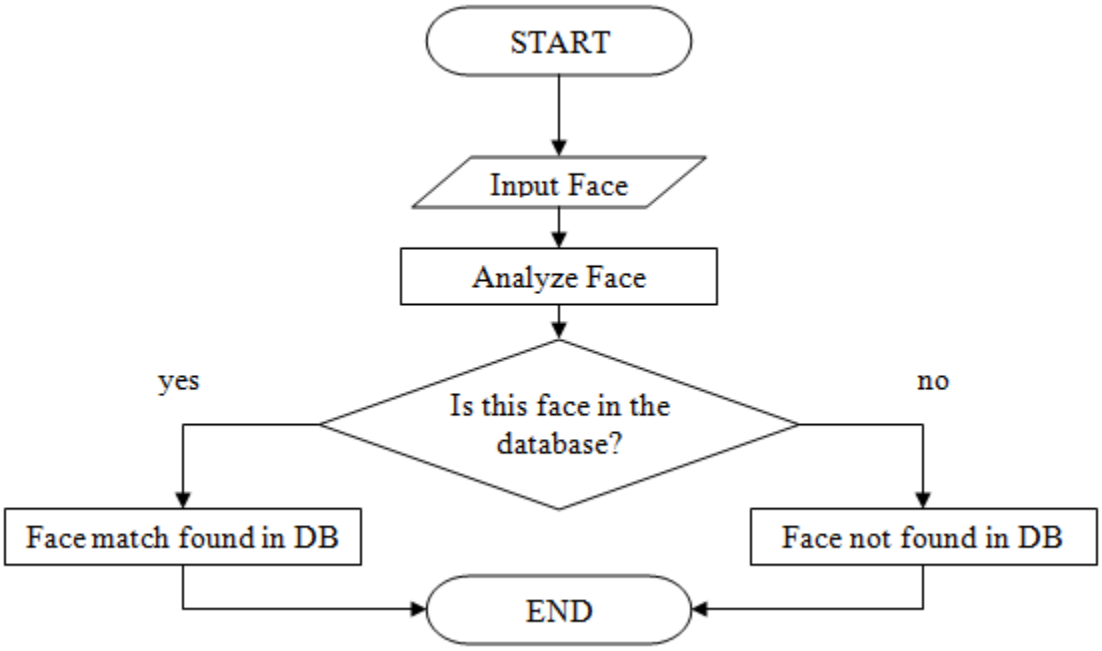

Figure 2-2: Simple Face Recognition Algorithm

2.3. Importance of Face Recognition

Many different applications of face recognition technology already exist or are being developed. Developers of security systems have begun to implement face recognition codes to help detect wanted criminals at large events, like professional sports games. Additionally, some countries (such as the United Kingdom) have begun to record countless hours of video footage where persons go about their normal lives (7). With this ever-increasing database of source images, the need to quickly and accurately determine where wrongdoers and wanted persons are becomes evident.

Facial Recognition Technology (FRT) is not limited to analyzing continuous video streams for specific persons. FRT can also be used on much smaller scales. For instance, web

cameras (small video cameras) and fingerprint scanners have become mainstream laptop computer components. These laptops can be configured to require a user's fingerprint to be correctly matched to a stored profile before logging on; thus preventing a thief or mischievous person from accessing the computer--even with a password. Conversely, it would be just as simple to use the computer's on-board camera and FRT to accomplish the same goal or provide a redundant system.

The most useful implementation of would likely be airport (or other heavily trafficked areas) security systems. Current technologies require would-be passengers to go through magnetic scanners which detect large metal objects, provide an acceptable identification card, and submit to either a full-body pat-down or scan (8). These highly invasive technologies have met sharp criticism from both the general public and airline employees. Concerns include over-exposure to radiation, reduction in personal privacy, and infringement of civil liberties (9). However, facial recognition may alter the situation. Instead, passangers could submit to a facial test for known suspects.

Currently implemented Facial Recognition Technologies have not been without its critics or public failures. In the case of London England, where video camera systems have been combined with FRT to aid in catching terrorists, no arrests were made between the system's implementation in the early 1990s and 2002, when the system was shut down. The city of Tampa, Florida also dabbled with a similar system to London starting in July 2001 then had the surveillance system removed from service less than a year later. One critic even went so far as to call FRT "high-tech snake oil" (10). However, a reliable, accurate, and quick method for identifying an individual in some form of electronic database of facial images remains extremely useful if a method that is both accurate and fast can be found.

While the algorithm documented here was developed specifically for faces, it is not limited to face recognition. The methodology could be used to develop a signature (scale spectrum) for any digital image, be it an image of a crack in a beam, widget in a factory, etc. For example, by knowing the signature of a "good" widget, any widget can be checked for quality control. Similarly, a strain gage could be developed which relies on deformation of a known pattern attached to a deformable body. Another application would be camouflage detection. Again, by knowing a scale spectrum for a region which did not contain anything, a region of interest could be compared. Any differences may indicate the presence of a camouflaged object.

2.4. Other Facial Recognition Algorithms

The following section provides a detailed look at four of the popular facial recognition algorithms in use today. These algorithms are the primary choices for facial recognition starting with an electronic image as the source--the means by which the research documented herein will also sample faces. However, there are other systems in development which do not use an electronic image. Methods which exclude electronic images have been purposefully omitted from this document.

This section will open with a note about what constitutes an electronic image and a description of the database selected for this research. It then will proceed with a detailed discussion of projection methods which are based on first, second, and higher order statistical methods. These projection methods will constitute the bulk of the section, but are not the only methods documented. The final method documented instead makes use of a wavelet transformation of specific points in a grey-scale image.

2.4.1. Electronic Images

Because modern computer facial recognition algorithms (FRA) make heavy use of electronic images, a brief introduction to electronic images is presented. While different file formats vary, at the basic level, an electronic image is nothing but a 2-D array, whether the image has been stored on a hard disk or is in RAM. This 2-D array contains a finite number of pixels, the smallest addressable elements in a digital image (11). Generally, a pixel is considered to be a rectangle, although the pixel may be given any geometric shape in the context of a specialized case.

In the case of a binary image, the pixels are fixed at either black (0) or white (1). However, most electronic images are not binary; most images today are color or grey-scale. In the case of a grey-scale image, each pixel has an integer value which corresponds to a specific grey-scale level. The number of integer levels is controlled by the number of bits used to track the pixel value. For instance, a typical 8-bit image would have 256 discrete pixel values ranging from 0-pure black-to 255-pure white (12).

Color images follow a similar construct to grey-scale images. Typically, three color channels are isolated and treated the same way as the grey-scale. These colors are normally red, green, and blue (hence the RGB spectrum). Color images then have three 2-D arrays of integer values that are stored independently. When called to the computer screen, the three separate color channel arrays must be interlaced in some fashion in order for the "true" color to be perceived by the human eye (12).

The differences in various file formats (ex. jpeg, tiff, bmp) arise primarily from the method of compression. In order to save computer physical memory, image files are not stored directly as the 2-D array form (except for tagged image file format (tiff)). These various formats

also contain different header information or meta-data (e.g. location, photograph date, photographer).

2.4.2. The Facial Recognition Technology (FERET) Database

The Facial Recognition Technology (FERET) database (3) is a database of facial image files. It contains approximately 14,000 tiff image files and is distributed on PC DVD-ROM discs (and may be requested from <http://face.nist.gov/colorferet/>). The database was created by the DOD's Counter Drug Technology Program, conducted at DARPA for the purpose of providing a standard database for development of facial recognition programs and algorithms (3). The FERET database is available for free and many other facial recognition algorithms have been tested using the FERET database for verification and validation. Several major competitions have also used this database (13), (14), and (5), making the FERET database the ideal choice to test the new algorithm presented herein.

The FERET database contains multiple images of the same individual face. Some are taken at different angles; see below (Figure 2-3). There are also images of the same individual in different lightings, with different facial expressions (shown below in Figure 2-4), and some subjects are in both color and black-white images.

Figure 2-3: Sample poses from the FERET database

Figure 2-4: Sample expressions from the FERET Database

Each image file is given a unique name that generally takes the form of *XXXXX_YYYYYY_ZZ_a*, where the "_a" is not always present and is used to denote a subset or replication of conditions. The first 5 digits XXXXX are used for an integer that is unique to each subject in the database. There are approximately 1,000 subjects in the database. The next 6 digits YYYYYY are used to store the date the image was taken in the form 2-digit year, month, and day. The last two digits ZZ are used to describe the conditions of the image. Table 2-1 gives a brief summary of these conditions (3).

Table 2-1: Pose Code Letters Overview

Code (ZZ)	Meaning
fa	Primary frontal image of the subject
fb	Alternate facial expression, same pose and lighting as fa
ba	Secondary frontal image, no change in facial expression
bj	Alternate facial expression, same pose and lighting as ba
bk	Different illumination of ba
bb	Subject is faced 60° to the photographer's right
bc	Subject is faced 40° to the photographer's right
bd	Subject is faced 25° to the photographer's right
be	Subject is faced 15° to the photographer's right
bf	Subject is faced 60° to the photographer's left
bg	Subject is faced 40° to the photographer's left
bh	Subject is faced 25° to the photographer's left
bi	Subject is faced 15° to the photographer's left
ql	Quarter turn left
qr	Quarter turn right
hl	Half turn left
hr	Half turn right
pl	Profile (full) turn left
pr	Profile (full) turn right
ra	Random images at various angles
rb	
rc	
rd	
re	

2.4.3. An Overview of 2-Dimensional Computer Algorithms for Face Recognition

Computer or algorithm-based facial recognition has existed conceptually for some time (15). An early attempt at facial recognition used polls of actual people to determine which facial features were important in identifying a person. Researchers combined this information with graph theory to predict correlations required for face identification. Algorithms then made comparisons between photographs in the rather obvious manner of checking these identified features for concurrence. Somewhat ironically, almost 30 years later, the research and development documented herein will return to a similar scheme. However, it is important first to discuss some of the more recent advancements of facial recognition algorithms.

Due largely to the ever-growing size of electronic photographs, storing and working with an entire 2-D electronic face image is overwhelming and computationally difficult (16). During the rapid development of computers in the 1980s, many different schemes for facial recognition were proposed and tested. The majority of these early schemes focused on object extraction such as the physical location of the nose or eyes, rather than a global approach using the entire face (17). By the end of the 1980s, facial recognition algorithms had been confined to simple appearance based statistical methods which further developed into the projection methods used today, most notably with the work of Kirby and Sirovich (17), (18).

2.4.4. Principle Component Analysis

One popular scheme is principle component analysis (PCA) (17). In PCA, a face image (or rather any electronic image) is reduced from its 2D form to an image subspace. In order to perform PCA, a group of "training" images is required. These training images must be

representative of images used in comparison to the database of images. If one starts with a dataset of *p* images which are said to belong to a training set *G*, for PCA to work, each image in *G* must have both the same number of pixels and dimensionality, which will be specified by the number of rows and number of pixels per row ($m \times n$). This implies that the total number of pixels which comprise a particular image is simply $N = m * n$.

First, a digital image is reduced to a vector by concatenating either the rows or columns (usually columns). This results in $X_i = [x_1, x_2, x_3, \dots, x_N]^T$. Once image concatenations have been computed for all images which will comprise *G*, the mathematic average image vector (the so-called training image, Equation 2.1), $\bar{X}$ is found and subtracted (Equation 2.2) from all image vectors in the database leaving a data set of arrays which represent the difference between that particular image and the average of all images.

$$\bar{X} = \frac{\sum_{i=1}^{N} X_i}{N} \qquad \text{Equation 2.1}$$

$$\tilde{X}_i = X_i - \bar{X} \qquad \text{Equation 2.2}$$

These $\tilde{X}_i$ are then placed as the columns in a large matrix of all images in the data set, *G*. This is commonly referred to as the image space. Once *G* is formed, the covariance matrix GG^T is formed and the principle components of the covariance matrix are found using Equation 2.3. (Note that Λ is the diagonal matrix of eigenvalues and *R* is a rotation matrix.) The original 2-D image coordinate system (typically a Cartesian system) is thus transformed through *R* into the subspace defined by selecting a series of the available eigenvalues - usually by choosing from the decreasing order of principle component of the eigenvalues until the desired number have

been selected. The eigenvectors associated with the chosen eigenvalues then define the orthogonal axes of the subspace coordinate system (18), (19). In Figure 2-5 there is an example of an average image and several "Eigenfaces" which are the subspace projections from a series of eigenvalues / vectors.

$$R^T(G\,G^T)R = \Lambda \qquad \text{Equation 2.3}$$

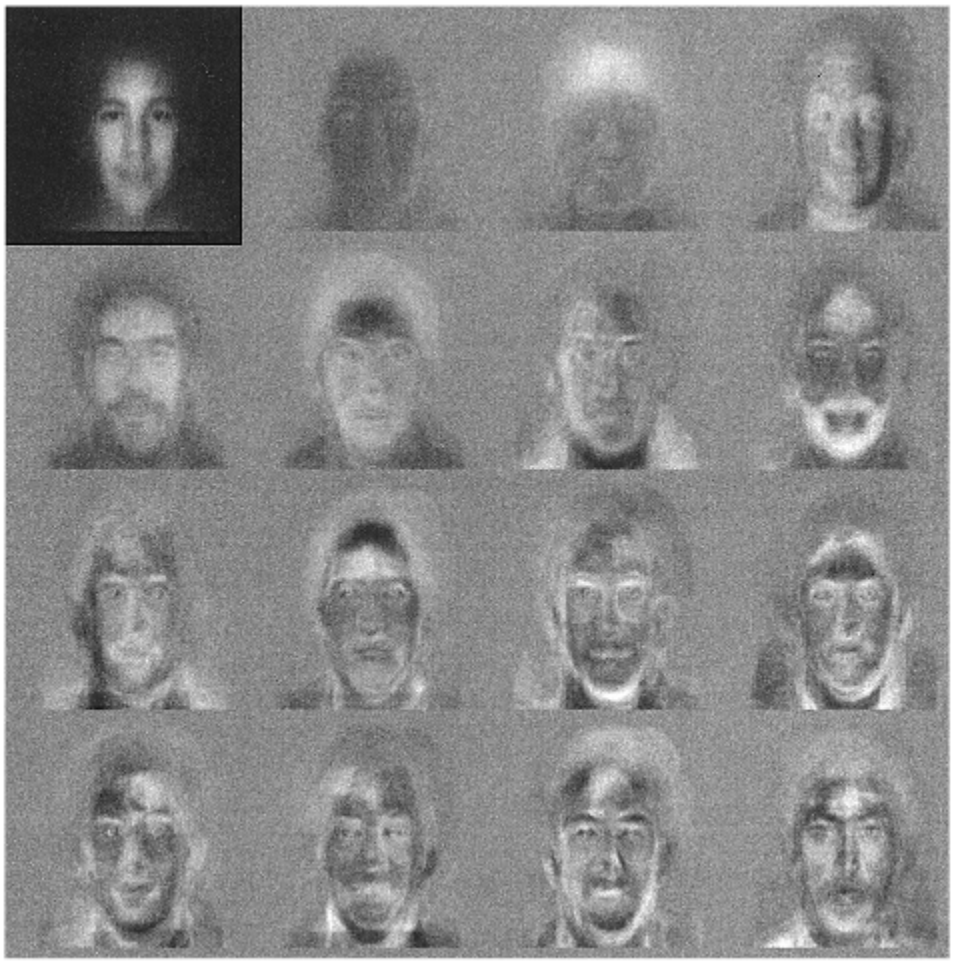

Figure 2-5: Eigenfaces (projections from the Eigenvectors produced solutions of Equation 2.3) and Average Image (top left) from PCA (17)

The problems with this approach arise primarily out of its relative simplicity. An image space has large dimensionality if either the data set is big or the number of pixels is high (a high resolution photo). While image space will eventually be reduced to a subset of eigenvalues and vectors, for a new image to be compared to the dataset, this computationally expensive process (Equation 2.1 to Equation 2.3) must be repeated (whether or not the trained average image is altered). Essentially, for every new test image, the eigenvalues must be found again. However, once complete, the comparison of one image to another is straightforward because the strength of the correlation between two images is inversely proportional to the Euclidean distance between them (18).

Since the projections into the subspace are dependent upon the original Cartesian image, any changes in rotation of the subject face relative to the observer (camera) would drastically affect the transformation into the subspace. Because each image in *G* must have exactly the same dimensionality, an additional pre-processing step is required to scale any non-conforming image. For instance, if the number of horizontal pixels of a test image were twice the dataset standard, some additional step would be required to remove pixels. Removing pixels could be extremely detrimental to an image and great care must be taken to do this properly. Changes in lighting and reflectivity (among others) can also cause PCA to fail (17).

2.4.5. Independent Component Analysis

Because PCA relies on a Euclidean distance to describe the correlation between two images, PCA only minimizes the mean-squared errors (18). Therefore, one of the logical extensions of PCA minimizes both the second-order (mean-square) and higher order errors. Independent Component Analysis (ICA) uses such higher order statistics to match faces (20). Essentially, the goal of ICA is to find a basis where the data are statistically independent in

higher orders. In doing so, it creates a linear, non-orthogonal coordinate system. However, ICA retains the linearity of PCA.

In this formulation, *s* is an unknown vector, *A* is an unknown mixing matrix, and *x* is a known source which results from the mixing as given in Equation 2.4. The goal of ICA is to decouple this mixing by finding another matrix, *W* (Equation 2.5), such that a reasonable estimation, *u*, of the original unknown *s* can be found.

$$x = As \qquad \text{Equation 2.4}$$

$$u = Wx = WAs \qquad \text{Equation 2.5}$$

In order to increase this formulation to higher than 2nd order (PCA), the signals must be statistically independent. The signals are said to be statistically independent when they satisfy Equation 2.6, where f_u is a probability density function in *u* (18).

$$f_u(u) = \prod_i f_{u_i}(u_i) \qquad \text{Equation 2.6}$$

In practice it can be very difficult to analytically solve for a *W* which satisfies Equation 2.6. The primary differences in ICA arise then from the various iterative approximations which are subsequently used. Three basic approximations are InfoMax (entropy-based) (21), JADE (kurtosis-based) (18), and FastICA a general and robust algorithm (22). These three algorithms will be covered briefly below.

For the InfoMax formulation, the entropy $H(u)$ (Equation 2.7) is maximized , which in turn, maximizes the signal independence. InfoMax uses the gradient ascent of *W* to accomplish the maximization of entropy (21). Conversely JADE relies on the kurtosis (or simply how

sharply peaked the distribution curve f_u is). In minimizing kurtosis of f_u by a joint diagonalization of the fourth order cumulants, JADE also maximizes the independence (18).

$$H(u) \stackrel{\text{def}}{=} -\int f_u(u) \log f_u(u) du \qquad \text{Equation 2.7}$$

Finally, FastICA minimizes Equation 2.8 where G is some general non-quadratic function, v is a Gaussian random variable (Gaussian random variables are normally distributed over a fixed interval), c is a positive constant, and E is any function of interest. Bartlett et. al. have shown that in maximizing Equation 2.8, the statistical independence is also maximized (22). For specific information on algorithm implementations, the reader is referred to several examples which can be found in Bell and Sejnowski (21), Chichocki, Unbehauen, and Rummert (23), and Comon (24).

$$J(y) \approx c[E\{G(y)\} - E\{G(v)\}]^2 \qquad \text{Equation 2.8}$$

ICA methods, as applied to facial recognition, generally fall into two categories, referred to as "Architecture I" and "Architecture II" (20). The difference between these two approaches is essentially the frame of reference. In Architecture I, the database is arranged into a matrix where each row is an image (similar to PCA). Each pixel can be chosen and compared to another. For instance, two images are independent of each other if the pixels from one image cannot be used to successfully predict the pixels from the other. Similarly, Architecture II places the images in columns of the matrix (exactly like PCA). Images are tested, instead; for example two pixels are independent of each other if it is not possible to predict the value of one pixel from another in the same image. Figure 2-6 shows this method pictorially.

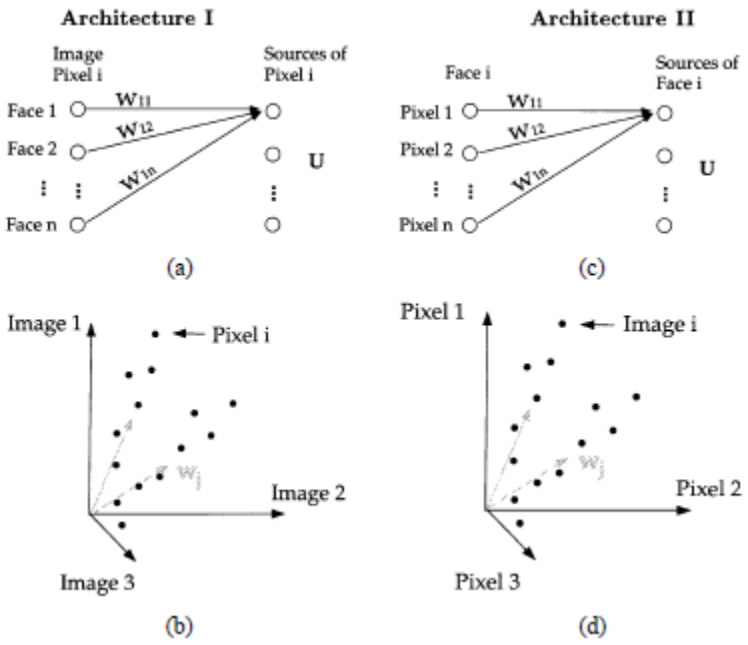

Figure 2-6: Two architectures for ICA (a) and (b) are examples of Architecture I. (c) and (d) are examples of Architecture II (20).

ICA suffers from a similar set of problems as PCA. Again, it is computationally intensive and sensitive to the rotation of the subject. The iterative processes used to solve for *W* are usually chosen based on a specific goal or feature of the database. While these processes are similar, they may produce different results under different circumstances. However, as with any iterated process, the computational time required may be quite great, especially when applied to a database containing a large number of digital images.

2.4.6. Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is another extension of PCA, however much more abstract. LDA attempts to find the projection axes that maximize the distances of the different classes, which are a set of delineating features chosen from the images. In general these classes might be the location of the subject's eyes, or the curvature of the cheeks, or any "feature" the

researcher finds useful. In application though, the classes are restricted to the projection axes resulting from the eignenvalue approach of PCA. In other words, LDA locates vectors which best differentiate between images (25), instead of simply choosing from the maximum to the minimum eignenvalues as PCA does. To accomplish this, LDA maximizes the Fisher Discriminant Criterion (FDC), which is why LDA may also be referred to as Fisher's Linear Discriminant (FLD) (19). It should be noted that FLD is basically a ratio of signal to noise (Equation 2.9).

For solution of a system involving only one class, the FLD is defined through a series of successive projection axes (similar to PCA) and a solution parameter to be maximized, γ. Note that u here is a unit vector in the direction of projection. Recall that in PCA these projection axes are derived from the eigenvalues and vectors. LDA makes use of the same projections. Equation 2.9 gives the definition of γ as a ratio of the so-called scatter matrices (Equation 2.10 and Equation 2.11). Note that B is the desired number of classes (projection axes); P is a probability associated with a certain class, C; $\bar{k}$ is an average vector over all C; and x is a vector specific to class C (25).

$$\gamma(u) = \frac{s_b(u)}{s_w(u)} \qquad \text{Equation 2.9}$$

$$s_b(u) = \sum_{i=1}^{B} P(C_i)\{(k_i - \bar{k})u\}^2 \qquad \text{Equation 2.10}$$

$$s_w(u) = \sum_{i=1}^{B} P(C_i)E[\{(x_i - k_i)u\}^2] \qquad \text{Equation 2.11}$$

In practice the scatter matrices are defined differently (Equation 2.12 and Equation 2.13) to simplify the algebra and allow an alternate representation of γ (Equation 2.14). Note that T is the usual transpose operator.

$$S_b = \sum_{i=1}^{N} P(C_i)\left(k_i - \bar{k}\right)\left(k_i - \bar{k}\right)^T \quad \text{Equation 2.12}$$

$$S_w(u) = \sum_{i=1}^{N} P(C_i)E[(x_i - k_i)(x_i - k_i)^T] \quad \text{Equation 2.13}$$

$$\gamma(u) = \frac{u^T S_b u}{u^T S_w u} \quad \text{Equation 2.14}$$

Finally, the solution to the system is presented in Equation 2.15. Here, the w are the various vector solutions from the eigenvalues and vectors; commonly referred to as eigenfaces. The λ are any successive parameter being used to classify (or differentiate) an image and the k are simply used to denote a large system counting a particular k image (25).

$$S_b w^k = \lambda^k S_w w^k \quad \text{Equation 2.15}$$

As with ICA solutions the selection of classes may vary among research groups and application. For instance, Navarrete and Ruiz-del-Solar choose for the case of B classes B-1 independent values of λ (number of projected spaces) (25), as do Delac, Grgic and Liatsis (19). Delac, et. al. go on to note that "...this approach [LDA] can produce some problems." They list said problems as:

1. This eigensystem does not have orthogonal eigenvectors (recall that is advantageous to PCA for differentiation among images) because $S_w^{-1}S_b$ is, in general, not symmetric.
2. Matrics S_b and S_w are usually too big to be computationally tractable.
3. S_w may be singular and therefore non-invertible (19).

As a means of working around these problems, LDA often falls back to PCA to decompose the large image space matrix to the eigenvalues (26), (19). The problems of LDA are then essentially the same as PCA.

2.4.7. Elastic Bunch Graph Matching

Computational face recognition is not limited to the subspace projections used by PCA, ICA, and LDA. Another popular method is called Elastic Bunch Graph Matching (EBGM). In EBGM, an image is selected for analysis (either to be stored to a database or analyzed individually) and features of interest are isolated. These features may be specific locations of eyes or nose, or any distinguishing physical feature of the face. The (x, y) locations of each feature are recorded to be used later during transformation.

Once all locations of interest have been mapped for a particular image, a Jet (a simple convolution based on a wavelet transformation, symbol J) is calculated according to Equation 2.16 for each location and desired number of rotations. Note that I refers to a specific grey-scale image, x refers to a specific location and t is a dummy variable for integration. The domain of t should include the location (pixel) of interest. For this derivation, a variable which appears with an arrow is a 2-D Cartesian vector, but when appearing without the arrow is the magnitude of the vector (27). k is just a book-keeping integer used to track a particular Jet in a series of Jets which will be developed for a particular location.

$$J_k(\vec{x}) = \int I(\vec{t})\psi_k(\vec{x} - \vec{t})d\vec{t} \qquad \text{Equation 2.16}$$

$$\psi_k(\vec{x}) = \frac{\phi_k^2}{\sigma^2}\exp\left(-\frac{\phi_k^2\, x^2}{2\,\sigma^2}\right)\left[\exp\left(i\,\overrightarrow{\phi_k}\,\vec{x}\right) - \exp\left(-\frac{\sigma^2}{2}\right)\right] \qquad \text{Equation 2.17}$$

The ψ_k are the vectors which describe the shape of a plane wave. They are given in Equation 2.18. It should be noted that the j are the integer values for each of the desired frequencies (assume there are *M* frequencies selected), the *i* are the integer values between 0 and *N*-1 where *N* is the number of desired rotations for a desired frequency (27). σ is used as a phase shift during the transformation and taken to be 2π; σ will be ignored later.

$$\overrightarrow{\phi_k} = \begin{pmatrix}\phi_{k_x}\\ \phi_{k_y}\end{pmatrix} = 2^{-\frac{j+2}{2}}\begin{pmatrix}\cos i\frac{\pi}{N}\\ \sin i\frac{\pi}{N}\end{pmatrix}\pi \qquad \text{Equation 2.18}$$

This process must be completed for each of the chosen locations of interest and desired number of images in the database. If one had 15 images and 10 locations, there would be 150 iterations of this algorithm. However, each iteration of the algorithm has *M*N* calculations. In Wiskott, et. al. (27), the researchers chose 5 frequencies and 8 orientations resulting in 40 coefficients for each location of each image. So, using 15 images, and 10 locations, and the 40 coefficients, there would be 6,000 numbers to describe the faces in the dataset. This number is completely scalable and generally left to the researcher.

Wiskott et. al. (27) also developed a method for comparing Jets among face images. First, the notation of Equation 2.18 is simplified to Equation 2.19. In this compact form the a_k

are the coefficients developed from Equation 2.17 and the constant valued term (σ) has been removed.

$$J_k = a_k \exp(i\,\phi_k) \qquad \text{Equation 2.19}$$

Wiskott et. al. then use a similarity function to calculate one number, *S,* which describes the difference in two images for one specific feature (Equation 2.20). It should be noted that Equation 2.20 ignores the effect of phase and assumes the two jets are directly correlated spatially (i.e. two features of different images which are at the same relative point). However, Wiskott et. al. have also developed a means of correcting for a small distance error (the subject's face may not be in exactly the same location in image-space in different photographs). The interested reader is referred to source (27) for more information on this distance compensation.

$$S(J,J') = \frac{\sum_k a_j a_j'}{\sqrt{\sum_k a_k^2 \; \sum_k (a_k')^2}} \qquad \text{Equation 2.20}$$

EBGM does not suffer the same problems of large matrix manipulation as the projection methods do; however, the locations of interest in the image must be identified by hand. The pre-processing step inherent therein makes the method unsuitable for person of low-training. The success and accuracy of EBGM depend upon experienced selections ranging from the locations of interest to the desired number of rotations to use during analysis. Finally the integral of Equation 2.16 may not have an analytic solution. Consequently, some form of quadrature would be required, which may be computationally expensive--offsetting any gains of its compact notation.

The simplicity of a single number (similarity) for comparing two images (faces) does provide a much simpler form than calculating the Euclidean differences in large eigenvector matrices.

2.5. Summary

As has been shown in this Chapter, the currently available algorithms are very computationally intensive or difficult to use for minimally trained users. Any time a new image is added to a dataset, the algorithms must recalculate large matrix systems or solve imposing integrals. The next chapter will detail a new algorithm, which had both high accuracy potential and eliminated many of the computational requirements of algorithms documented in this chapter.

3. Algorithm Development

This chapter will discuss the development of the facial recognition algorithm used, as enumerated in Goal 1. It opens with a brief discussion of fractals, which are the focus of the identification process, and develops the algorithm in successive steps by modifying the random walk to create a fractal. Fractals are used to estimate the transition probabilities (the result of creating fractals with the Random Walk). Finally, the transition probabilities are normalized and plotted. The resulting normalization was used to compare one image to another.

The core of this algorithm was developed and documented by Deal (1) specifically for fingerprints. Following Deal's development, Stoffa validated the algorithm for the specific case of parallel lines with uniform spacing with Buffon's Needle problem (2), and Stoffa attempted to verify the fingerprint application by comparing the algorithm to a standardized fingerprint testing sequence. He found that the algorithm was ill-suited, performing near the bottom in his tests

compared to other fingerprint algorithms. However, Stoffa neglected to normalize the transition probabilities; which would have removed the effects of differing numbers of white or black pixels between fingerpint image files.

3.1. Fractals

For the purpose of this work, the word *fractal* will be somewhat limited compared its normal definition and scope. Here, a fractal will be a regular geometric pattern which is self-similar and has a repeated motif (28). To be self-similar, the pattern must be repeated within itself at a scale ratio, e.g. the process of dividing an equilateral triangle into 4 smaller equilateral triangles which are exactly 25% of the original area. If the triangle dividing process were repeated, there would be 16 total triangles with 6.25% of the original area, each. A repeated motif is a grouping of these self-similar patterns into a form or object which is scaled throughout the fractal. In this example, the motif could either be the triangles themselves or the ratio of areas.

The Sierpinski Triangle is a classic example of one of the simplest fractal patterns. The process is as follows and is demonstrated graphically in Figure 3-1 (28). With each repetition, the number of triangles increases; however, the number which can be displayed in a digital image is limited by the number of pixels.

1. Create a triangle (a) and then create three copies which are exactly 50% in width and height of the original triangle.
2. Arrange the copies so that they meet point-to-point (b). Notice that these triangles can only cover 75% of the area of the original triangle.
3. Repeat steps 1 and 2 for each series of scaled triangles (c-e)

Figure 3-1: Sierpinski Triangle fractal curve (progression is from a to e)

Each step of the fractal generation (i.e. going from b to c) is known as a "pre-fractal." A progression of pre-fractals (as shown in Figure 3-1) is known as a fractal curve. It is worth noting that as the fractal curve reaches infinity (or the number of pre-fractals becomes infinite), the area of the black triangle sub-sections becomes zero and the perimeter of the white sub-sections becomes infinite. The effects (scaling, multiplicity) associated with moving from one pre-fractal to another (i.e. advancing one iteration) is known as the fractal dimension. The general formulation for fractal dimension, *d*, (for self-similar pre-fractals) is given in Equation 3.1. Here, *r* is the ratio of the scale factor of the fractal (in this example the ratio of triangle widths, or ½), and *M* is the number of repeated parts (or sections) of the original shape (in this example 3) (28).

$$d = -\frac{\ln M}{\ln r} = -\frac{\ln 3}{\ln \frac{1}{2}} = 1.58\ldots \qquad \text{Equation 3.1}$$

However, a fractal pattern (one pre-fractal) can also be generated by invoking a Random Walk process. A random walk is an iterated process in which an object travels a fixed distance, stops, arbitrarily selects a new trajectory, and repeats the entire sequence (29).

A common freshmen engineering assignment at WVU is the creation of the Sierpinski Triangle through the random walk. To create the Sierpinski Triangle from the random walk, start with an empty triangle with known coordinates of each corner (vertex). For this example, consider a triangle 1,000 pixels wide by 1,000 pixels in height (Figure 3-2).

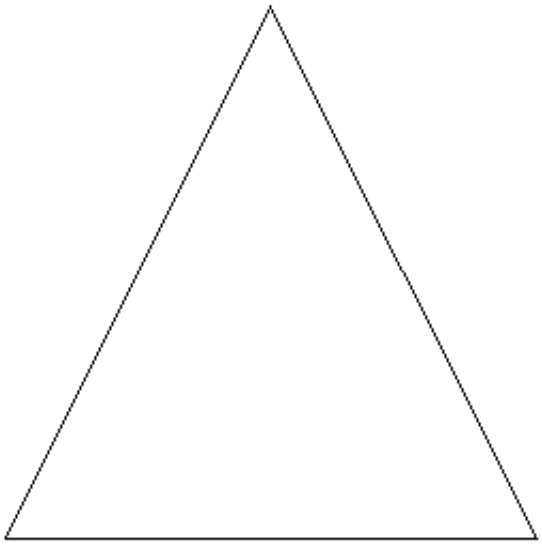

Figure 3-2: Seirpenski Triangle Domain Example

To start, one point in the triangle must be selected. The starting point may be selected randomly from the triangle; however this has no effect on the resulting fractal. The middle of the triangle is commonly used as the first point. From there, select one corner randomly from the three and move half of the distance from the current point toward the corner. This is the second point (or new point). The selection of moving half the distance to the randomly chosen corner is

a rule known as the Chaos Game. For visualization, the new point should then be marked by changing the pixel value associated with it. Once the new point is marked, the process is restarted by selecteing a new corner randomly and repeated until the desired approximation of the fractal is complete (Figure 3-3).

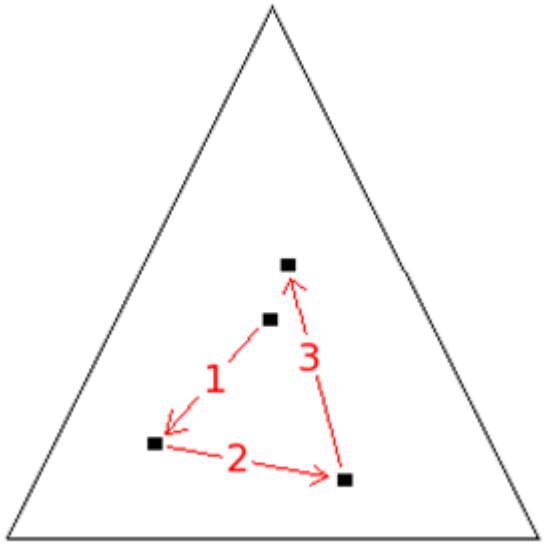

Figure 3-3: Sierpinski Triangle (Blown Up View, performed by hand)

In other words, the fractal is simplified to a finite list of random numbers (corner selections, say 1, 2, or 3) and Cartesian coordinates. This list is a dataset, which when plotted in a digital image, represents the one pre-fractal in the fractal curve. To move along the fractal curve, the resolution (number of pixels) must be altered (increased or decreased).

The number of iterations required will be dependent on the number of pixels in the image and the desired clarity (or resolution) of the fractal. For this example there were *1,000 x 1,000* pixels, (1,000,000) in the entire image. Since the pixel is the smallest addressable screen element, there is a limit to the resolution which can be achieved. For this example 1,000,000 iterations were used to ensure adequate resolution of the approximated fractal (Figure 3-4);

however, there appears to be little qualitative difference between 100,000 iterations and 1,000,000 iterations images (Figure 3-4 c and d, respectively). Note: the borders of the triangle have been included for clarity.

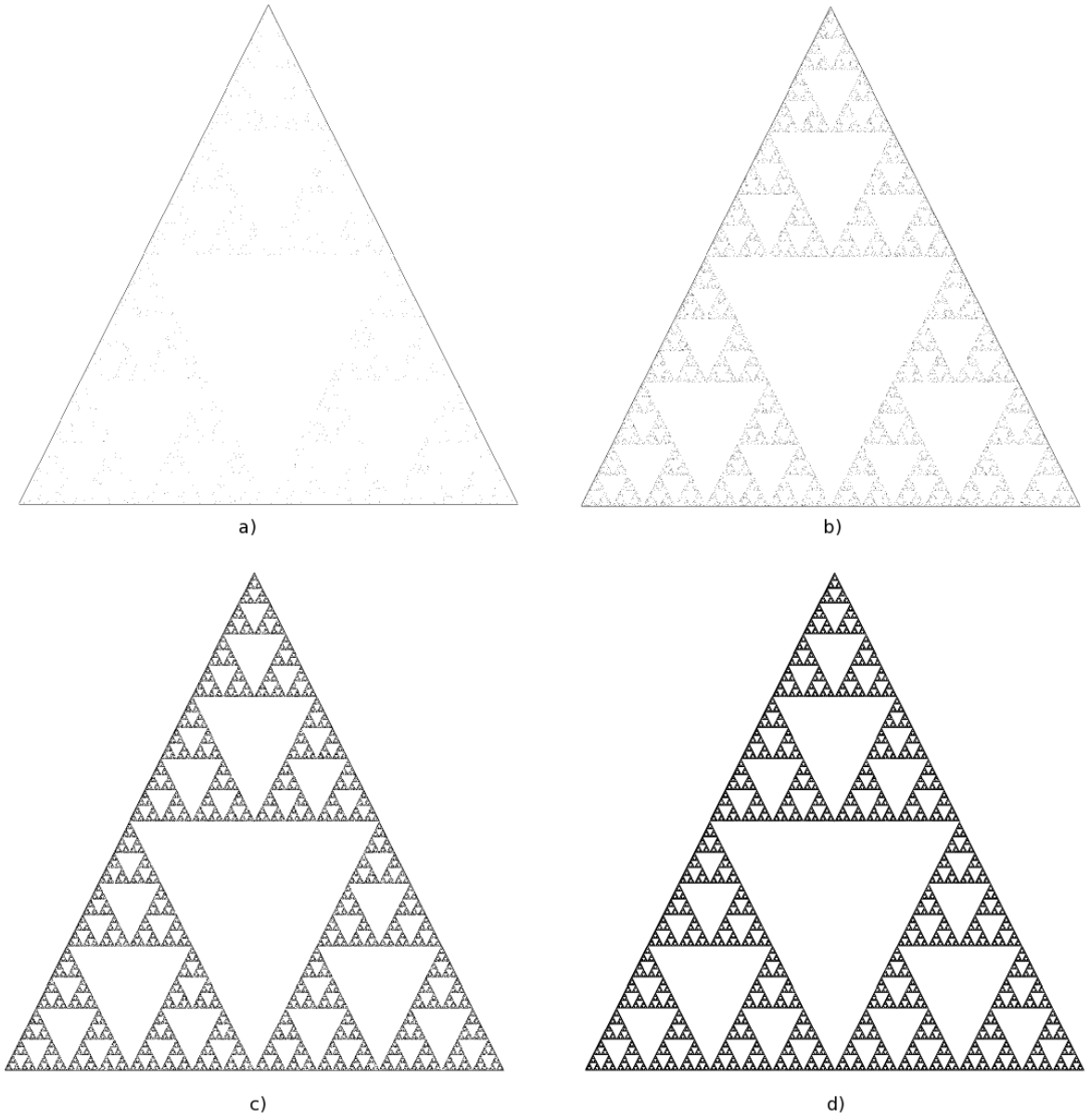

Figure 3-4: Approximated Sierpinski Triangle after a) 1,000 iterations, b) 10,000 iterations, c) 100,000 iterations, d) 1,000,000 iterations. Note: the image quality was reduced to insert into this document.

This foundation of creating a fractal from a random walk was modified so that a binary image can be used to dictate the fractal creation instead of choosing corners randomly from the fractal (1).

3.2. Obtaining the Dataset for Binary Images

Consider an electronic image of some shape which can be represented in binary, and define it as an Image Space with domain $\Omega_I(x, y)$. Ω_I then is a discrete 2-D Cartesian region of a finite dimension, $(m\ x\ n)$ contianing only Boolean values (0 or 1). Figure 3-5 gives a small example of such an Ω_I; note that the pixels do not in reality posses border lines, which have been added for clarity.

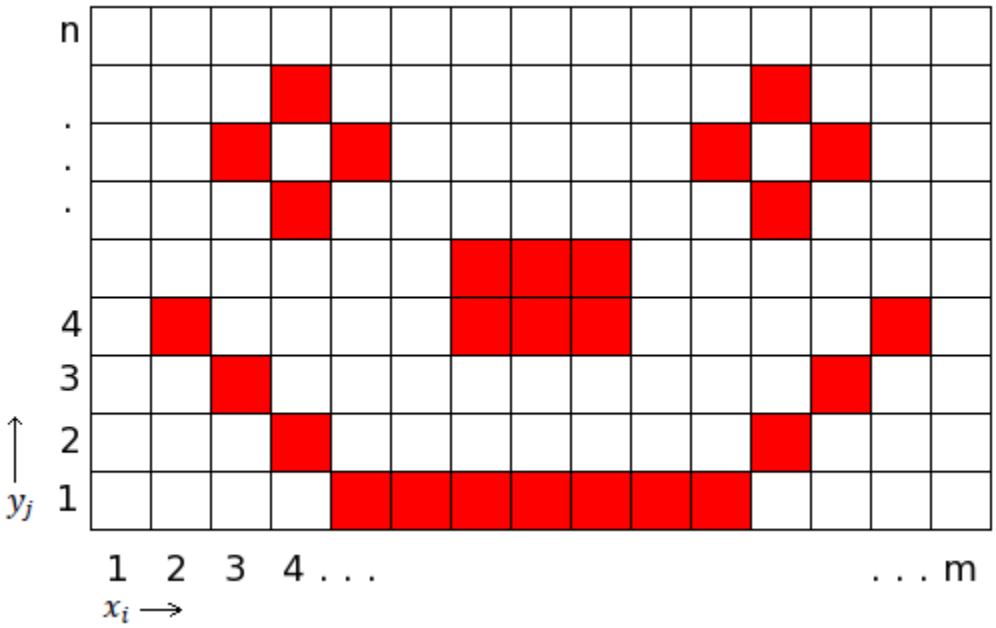

Figure 3-5: Example Image Space (red has been used to denote black because of pixel border lines)

3.2.1. Generating a Fractal from a Modified Random Walk

First, a pixel (x_i , y_j) from Ω_I is selected randomly (blue pixel in Figure 3-6) just like the corners from the triangle in the random walk. This pixel is the mid-point of a line segment of length *l*, which is known and fixed prior to the modified random walk. The length of the line-segment is also called "scale". Next, an angle of rotation for that line segment Θ is also selected

randomly from $\Theta \in [0\,, \pi\,]$. This second random selection is different from the Sierpinski triangle. The line segment is then rotated by Θ as shown pictorially in Figure 3-6. Finally, the values (0 or 1, black or white) of the pixels at the endpoints of the line segment are noted. In Figure 3-6, these endpoint pixels are marked in green. The coordinates of the pixels are found from Equation 3.2.

$$(x_1\,, y_1) = \left(x_i + \frac{l}{2}\cos\Theta\,,\; y_j + \frac{l}{2}\sin\Theta\right)$$

$$(x_2\,, y_2) = \left(x_i - \frac{l}{2}\cos\Theta\,,\; y_j - \frac{l}{2}\sin\Theta\right)$$

Equation 3.2

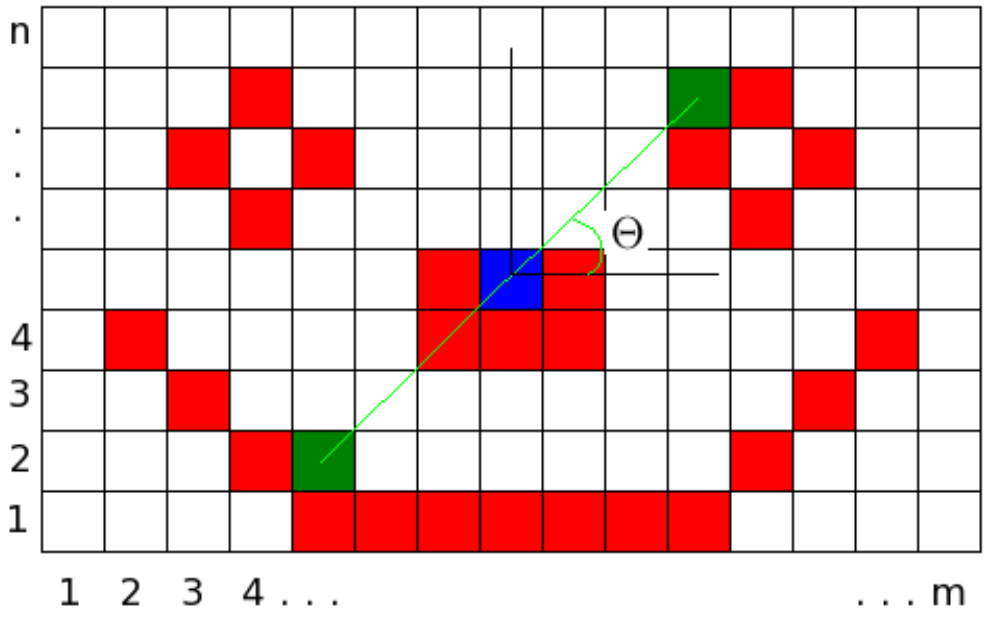

Figure 3-6: Sample Image Space with selected pixel (blue), rotation, and end-points (green).

Note: the local coordinate axis has been provided at the randomly chosen (blue) pixel to graphically demonstrate the angle of ration.

Because Ω_I was chosen to be Boolean Image Space, there are only two possible values for the end-points of the line segment: black (0) or white (1). The four possible combinations of the end point values then are white-black (WB), white-white (WW), black-white, and black-black (BB). It is these combinations which are used to create the fractal (as opposed to randomly selecting one corner of the triangle). Because there are four combinations, the fractal will be extended to a rectangle from the triangle. Figure 3-7 gives a representative fractal-space where the corners are labed based on the aforementioned combinations.

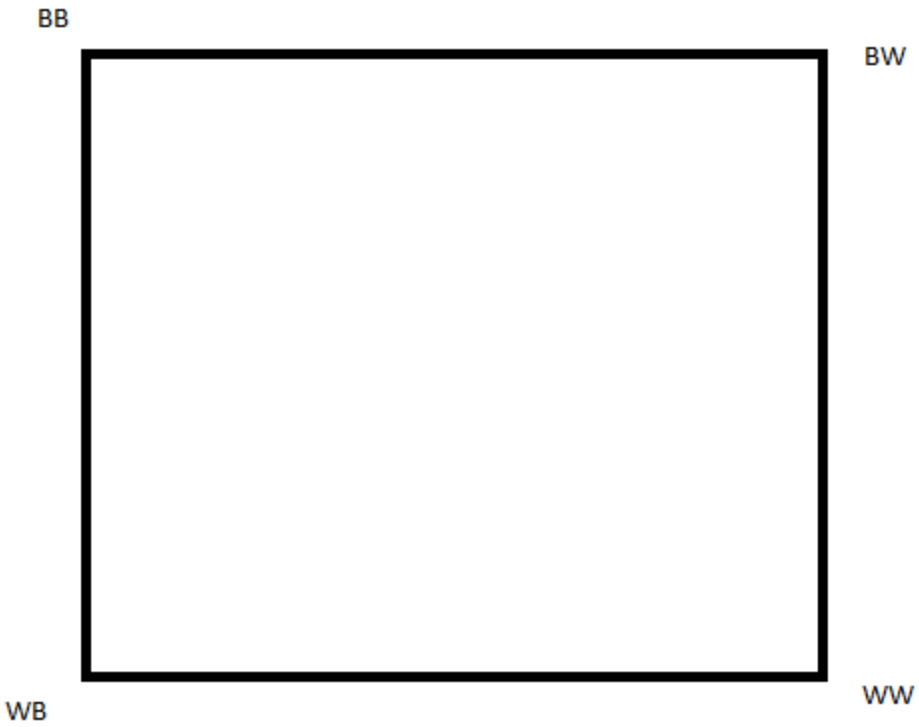

Figure 3-7: Sample Fractal Space

Now, the fractal will be created in a separate space from the image. This new space will be generically referred to as fractal-space with domain Ω_F. Just as with the Sierpinski triangle, the fractal-space is discrete (repsented by pixels). For the examples which follow, fractal-space

was chosen as a rectangle with 1,000 pixels per side. And as with the triangle, the resolution of the fractal-space will dictate the clarity of the resulting pre-fractal image.

Just as with the triangle, once a pixel combination has been selected from image-space (BB, BW, WW, or WB), half the distance to the corresponding corner in fractal-space is traveled and the new point (pixel) is marked. Once the point is marked in fractal space, a new point and angle are selected randomly in Ω_I, a new pixel combination is found, and the next point in fractal-space is marked by traveling half the distance the corresponding corner. This process is repeated for a fixed number if iterations. In a similar manner to the Sierpinski triangle example presented, the fractal pattern may be thought of as a list of selected pixel value combinations and marked points.

When a pixel is selected in Ω_I near the boundary, the possibility of crossing the boundary with the line segment exists. Line segment endpoints beyond the boundary are not physical (the electronic image does not exist there and as such is not valued there). To avoid excessive computational time to determine the exact distance from the boundary a particular angle may permint without the line segment endpoints crossing the boundary, the limiting case of $\Theta = 0$ is chosen, implying that all randomly selected pixels must be selected at least $l / 2$ pixels from the boundary of Ω_I.

3.2.2. Example Fractal from Vertical Bar Pattern at a Fixed Scale

For example, consider a square image space of 1,000 pixels across with a repeated pattern of vertical lines. The first vertical portion will be 5 pixels wide and black. The second portion will be 10 pixels wide and white. This pattern will then repeat across Ω_I from left to right (Figure 3-8). For the analysis, a scale value of 7 pixels will be used because 7 is approximately half-way between the bands of 5 black and 10 white pixels. Figure 3-9 gives the progression of

the fractal in a similar manner to the Sierpinski example. Here, the fractal image is the same size as Ω_I.

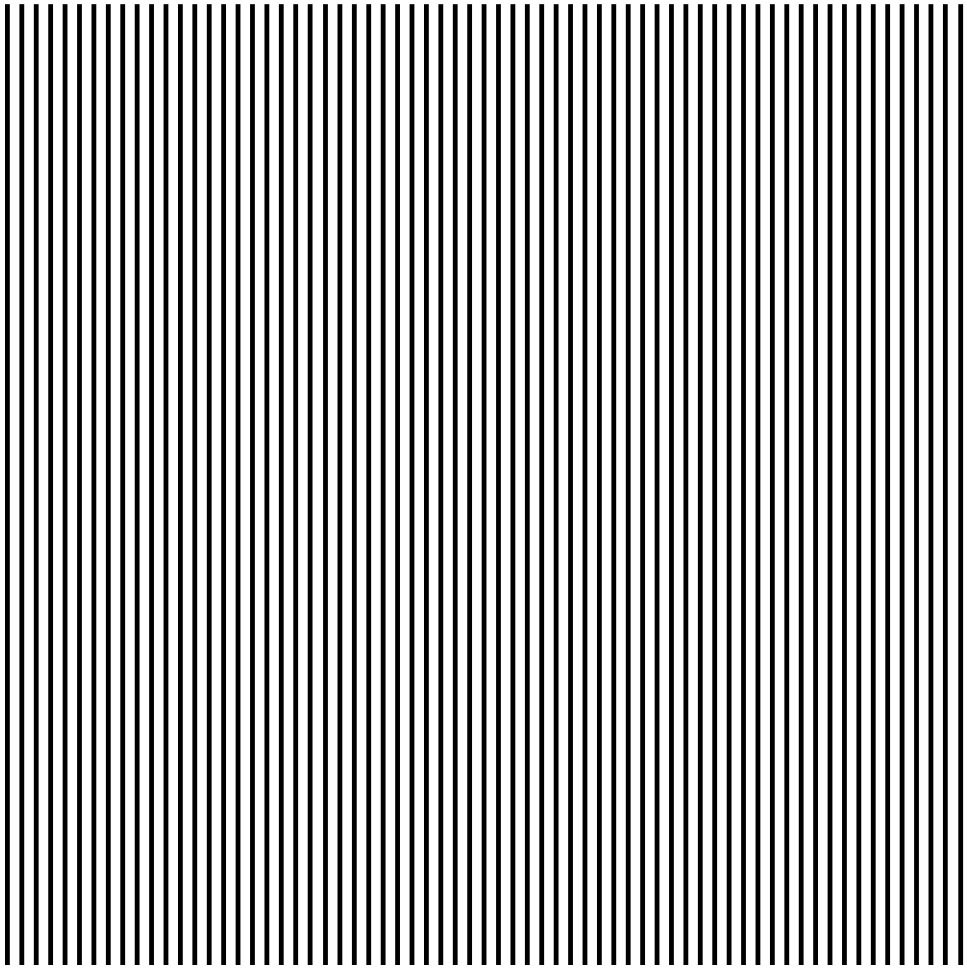

Figure 3-8: Sample Ω_I

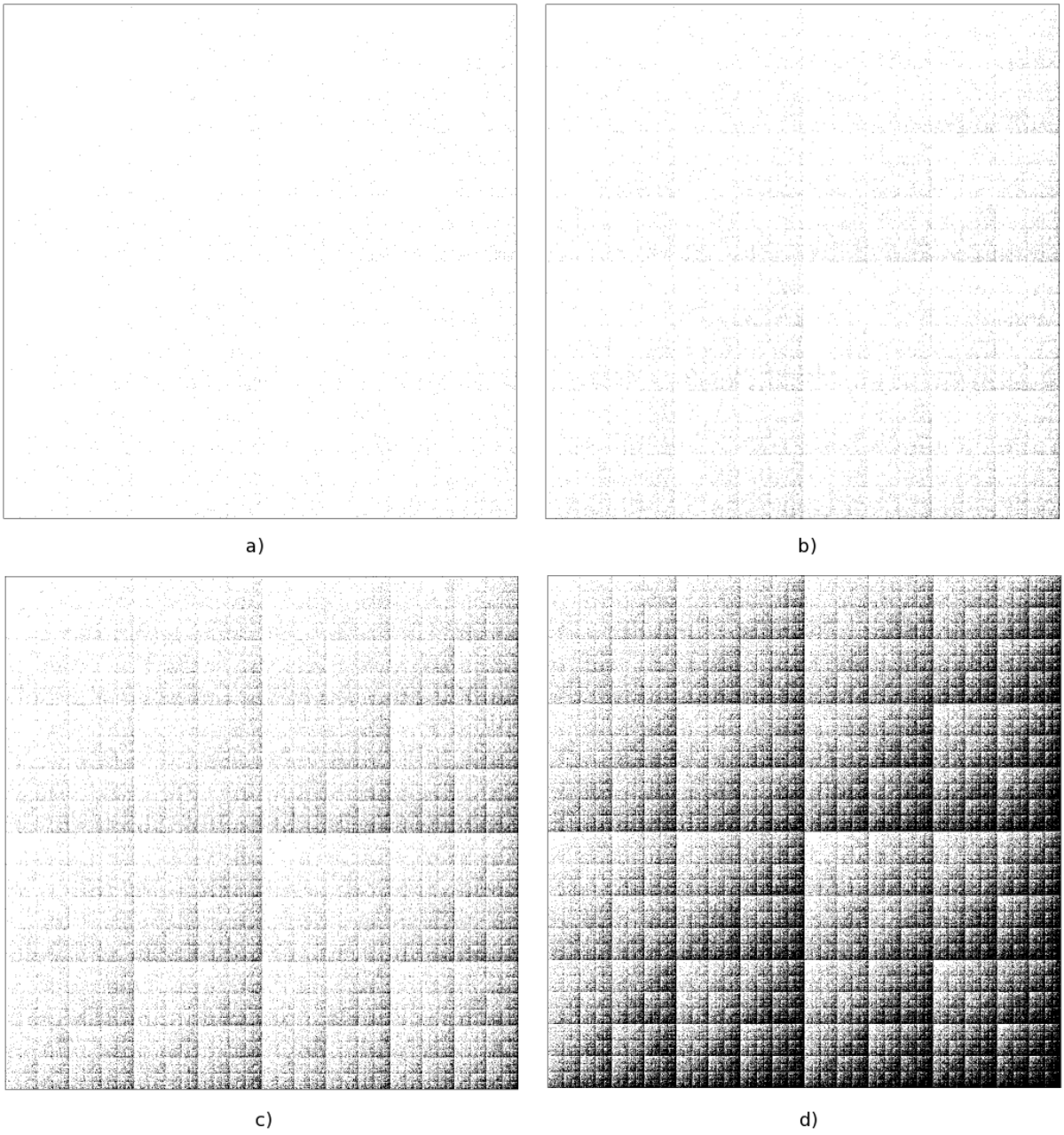

Figure 3-9: Fractal generated from the pattern above after a) 1,000 iterations, b) 10,000 iterations, c) 100,000 iterations, and d) 1,000,000 iterations. Note that the borders have been artificially added for clarity.

Recall that the top-left corner of fractal space is the black-black pairing, top-right corner is black-white, bottom-left is white-black, and bottom-right is white-white and each time a particular combination is found in image space during the random walk analys, a pixel is

changed from white ot black in the corresponding corner in the fractal. As shown in Figure 3-9 d) there are substantially more pairings of white-white than black-black (count the number of blackened pixels in those quadrants). This makes sense physically, as image space had twice as many white pixels as black pixels; thereby increasing the odds of selecting a white-white over black-black combination at a particular scale (41% versus 8%, respectively).

Further, the white-black and black-white quadrants appear to have similar counts. The ends of the line segment which are used to generate the color combinations are calculated by rotating through the random angle (Equation 3.2) which is chosen to be measured relative to some local horizontal axis. However, this choice is entirely arbtrarly and imposes a "left" and "right" side of the line segment. It should not (and does not) matter whether the black pixel is on the left or right of the line segment. Logically then, the mixed quadrants (black-white and white-black) will always have the same patterns in the fractal.

3.2.3. Effects of Scale

A natural extension of creating a fractal using the Random walk methodology then is to vary the scale length (l). When the scale is changed, the fractal will also change because the frequency of the end-point black and white pairings of the line segment will also change. Consider a geometric example using the same simple, zoomed Ω_l from Figure 3-5. Randomly select a pixel (marked blue) and choose two different scales (5 pixels and 9 pixels) for the same point and angle (marked in green). By observation, for $l = 5$, the combination is white-black, but for $l = 9$, the combination is white-white (demonstrated in Figure 3-10). By extension from this simple pictorial example, different scales will produce different fractals since the change in scale alters the frequency of the combinations. To demonstrate this scale dependence, fractal images for various scales for the vertical bar pattern Figure 3-8 are shown in Figure 3-11.

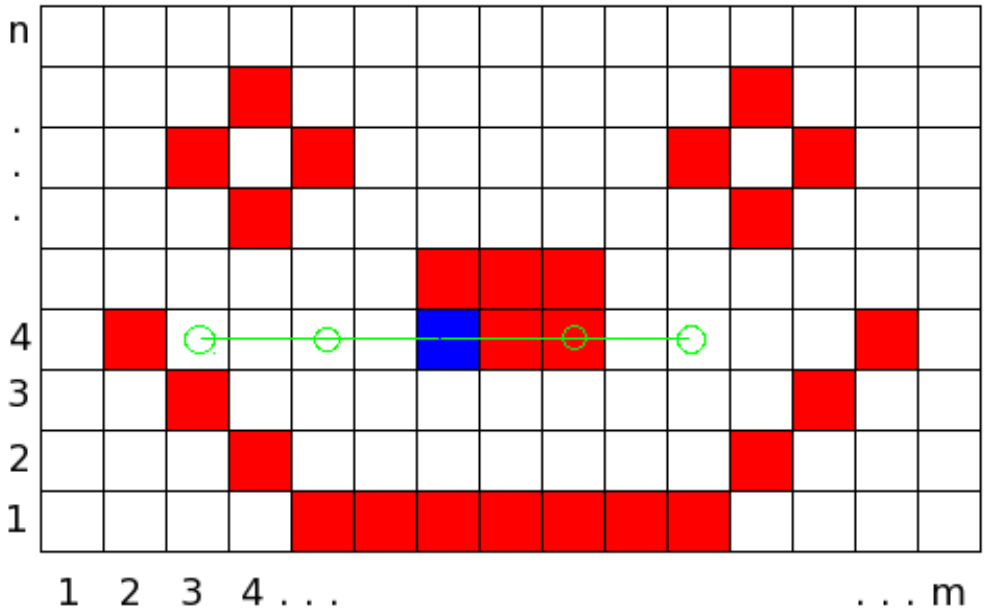

Figure 3-10: Image space with different scales marked for a random pixel for the same angle of ration

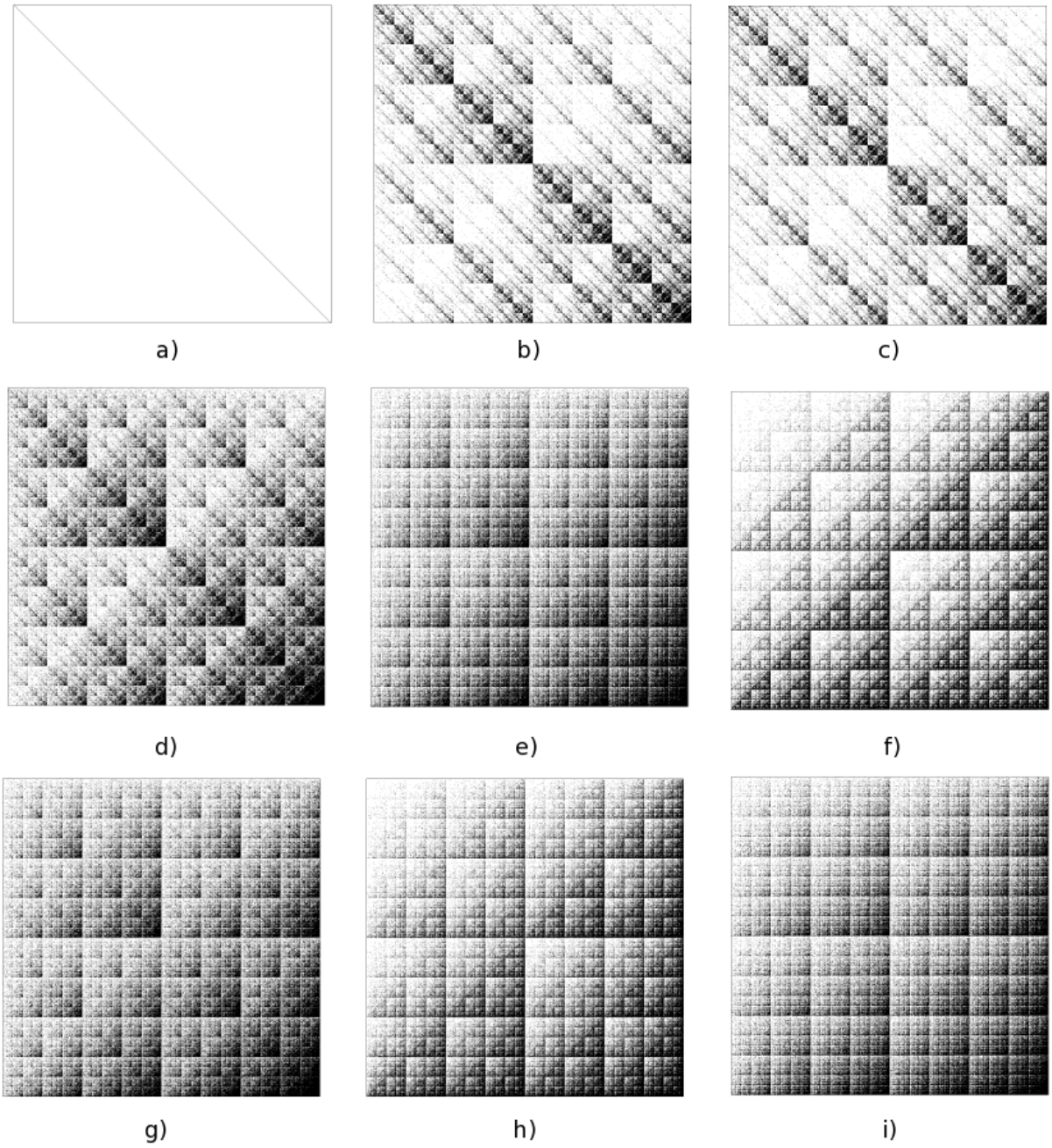

Figure 3-11: Fractal images of vertical bar for scales: a) 0 pixels, b) 1 pixel, c) 2 pixels, d) 3 pixels, e) 5 pixels, f) 10 pixels, g) 15 pixels, h) 25 pixels, and i) 50 pixels

Figure 3-11 a) is particularly important. In it, there is only a line instead of the square motif of the other examples in Figure 3-11. Because the fractal was started in exactly the middle

of the fractal-space domain, and because the scale was exactly 0 pixels, a) represents a count of black and white pixels in the image-space. The zero scale means that no line can be constructed which is able to leave the pixel which was randomly selected. This in turn means the ends of the line segment are also inside of the selected pixel. Since the end points have never left the selected pixel, the combinations can only be black-black or white-white for a black or white pixel respectively. The pertentange of pixels in image space which are black (or white) in a particular image then dictates the odds of choosing a black (or white) pixel at zero scale. For example, if 25% of image space were black pixels, then the odds of ranomdly choosing one black pixel would be 25%. Conversly, if during the random walk analysis 25% of the zero-scale combinations were black-black (black pixels), that would imply that 25% of image space were black pixels. Effectively, zero scale provides an estimate of the black and white pixel counts.

Notice also the trend of Figure 3-11 from small to large scale. The fractal images begin a transition from light areas in the mixed combination quadrants (top right and bottom left or black-white and white-black) to a uniform darkness throughout the fractal. This tendency results from the frequency of the various combinations. At small scale, the chance of selecting a mixed combination is small compared to a pure combination because the line segment tends to stay within a pixel or vertical bar. As the scale is increased, the line segment may overlap multiple vertical bands—in effect making the possibility of selecting a mixed combination higher. As scale becomes very large (on the order of the image), the possibility of selecting any of the four corners becomes equal.

3.2.4. Transition Probabilities

Stoffa demonstrated this modified random walk method for creating a fractal image is a Markov Chain process (2). A Markov Chain is simply an iterated process in which a future state

(location of the next point to be marked in the fractal) can be predicted from the current state (location of the current point) with a group of known transition probabilities (in this case, the corner which is chosen randomly) which describe all possible future states (30). In the example of the Sierpinski triangle, each of the three corners are equally probable (assuming the random numbers are uniformly distributed); so the transition probabilities or the corners are 33.3% each. However, the four corners of the feature-driven fractal from the random walk are in fact not equally probable. If each corner were equally probable, the resulting fractal would be completely filled in with black pixels or become grey space.

Difficulty arises in establishing values for transition probabilities in any fractal application. In order to establish values which have no inherent error, the modified random walk would have to be performed indefinitely or a closed-form solution must be determined. Stoffa was able to determine a closed-form solution for the particular case of vertical lines (Figure 3-8) by modifying Buffon's Needle, which is a classic statistics problem which strives to determine the probability of dropping a needle onto a crack in a wooden floor (31). However, the stringent requirements of Stoffa's solution are not practical for any other pattern, and large scales on the vertical bar pattern (specifically, Stoffa required the scale to be less than the width of the smallest band of black or white pixels).

Stoffa also demonstrated that the abbreviated (fixed number of iterations) process which is used to create the fractal from image-space, can also be used to predict the transition probabilities for a given scale according to the properties of a Monte Carlo integration (2). A Monte Carlo integration demonstrates that a series of random samples (in this case selecting the pixels from image space randomly) can be used to approximate the actual transition probabilities

or fractal image, with a predictable error (32). The order of the error for such a process is dictated by Equation 3.3 (informally known as *the Law of Large Numbers*).

$$Error \approx \frac{1}{\sqrt{Sample\ Size}} \qquad \text{Equation 3.3}$$

Intuitively from Figure 3-9 such a formulation of error makes sense. As the sample size increases, the fractal pattern becomes moredefined. By inspection, the cases for an interation count of 10^5 and 10^6 are qualitatively quite similar. This similarity is described by the error (Equation 3.3) associated with each: 0.00316 and 0.001, respectively. The small difference between these errors quantitatively explains what is qualitatively observed.

Essentially then, the transition probabilities for each quadrant of a fractal can be estimated with a finite number of iterations for a fixed scale with predictable error. Once the transition probabilities are estimated for a series of scales (in effect a series of fractal images such as Figure 3-11), the probabilities can be plotted as a function of scale. This ability to characterize a series of fractal images produced from binary image space with plots of four (BW, WW, WB, and BB) approximations of probabilities is important.

As shown in Equation 3.4, these probabilities (α_i) can be estimated by counting the number of instances a particular corner is selected during analysis, Φ_i, and dividing this number by the total number of iterations ($\sum_{j=1}^{4} \Phi_j$) for a particular scale. This counting method is only a first-order approximation of the true probabilities (2).

$$\alpha_i = \frac{\Phi_i}{\sum_{j=1}^{4} \Phi_j} \quad i = 1, 2, 3, 4 \qquad \text{Equation 3.4}$$

3.2.5. Scale Spectrum

A plot of the four transition probabilities as a function of the scale at which the probability was sampled is the scale spectrum for a given image-space. As already mentioned, Stoffa has provided a means of estimating these probabilities for individual scales. Returning to the example image-space of vertical bars (Figure 3-8), selecting 501 values for $l \in [0, 50]$ pixels, and using 10^6 iterations per scale, the resulting scale spectrum is presented in Figure 3-12. Note that *i* from Equation 3.4 has been replaced with letters which correspond to the 4 combination possibilities (ww, wb, bw, bb). In Figure 3-12 the probability curves of white-black and black-white lie on top of each other. As already noted in 3.2.2, the count of WB and BW are equal for a particular scale; meaning that the transition probabilities (which are calculated from those counts) are also equal. It should be noted that at scale = 0, the mixed probabilities are zero as predicted by the pre-fractal in Figure 3-11 a. Additionally, the transition probabilities at zero scale add up to 1 ($\alpha_{ww} = 2/3$ and $\alpha_{bb} = 1/3$).

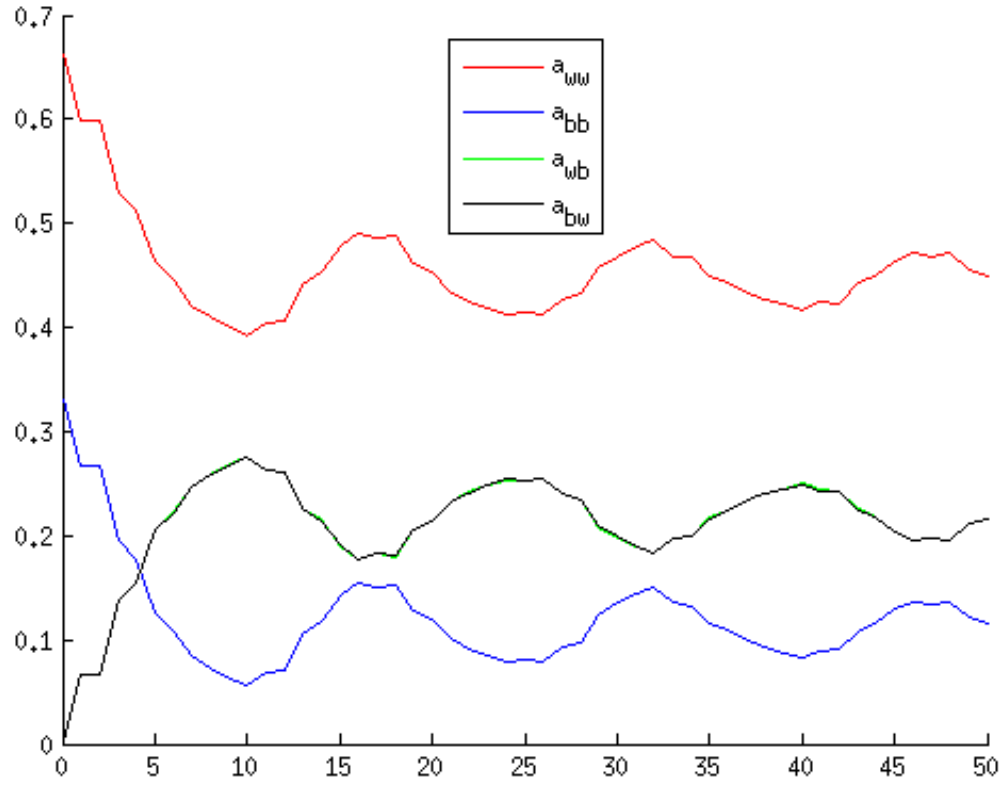

Figure 3-12: Sample scale spectrum generated from the vertical line pattern

These probability curves have an interesting shape. First, they appear to imitate a sinusoid and have comparitively the same periodicity. Likewise, as scale becomes large, a damping effect drives the amplitude of the curves to zero. The mixed probability (black/white and white/black) curves and pure probability curves are inverse (or possibly exactly out of phase such as cosine and sine) to each other. The BB curve has a local maximum while the BW curve is at a local minimum.

Since the mixed terms have equal probability, the notation is simplified as α_{ww} (white-white transition probability), α_{bb} (black-black), and β (either black-white or white-black). For convenience, the four transition probabilities can be arranged in a 2x2 matrix, P (Equation 3.5)

as a function of scale, l. Because Stoffa found that the determinant of P and the second eigenvalue of P are both equally effective at delineating one image-space from another, the trace and first eigenvalue, which were very poor delineators, will be ignored (2). The calculation of the second eigenvalue is directly related to the determinant, only the determinant (Equation 3.6) will be considered for delineation of one image-space from another to conserve computational time.

$$P(l) = \begin{bmatrix} \alpha_{ww}(l) & \beta(l) \\ \beta(l) & \alpha_{bb}(l) \end{bmatrix} \qquad \text{Equation 3.5}$$

$$D\big(P(l)\big) = Det(P) = \alpha_{ww} * \alpha_{bb} - \beta^2 \qquad \text{Equation 3.6}$$

In a desire to use fratal analysis of binary images to compare similar subjects, the determinant alone is not sufficient to delineate among image spaces. Two separate binary image-spaces of similar subjects may have differing numbers of black (or white) pixels. Any such difference may skew respective scale spectra. This discrepancy in black pixels is not necessarily indicative of different subjects which are presented in the image spaces. For instance, the vertical bar pattern already presented makes use of alternating bands of 10 white pixels and 5 black pixels, which effectively means that 33.3% of the image space area is black and the remainder white. To effectively compare this pattern to another pattern which may have equal areas of black and white, the determinant is normalized by these area percentages. In practice, the $l = 0$ probabilities are used because they are representative samples of the number of black and white pixels in the image-space.

One simple normalization scheme is dividing the determinant by the product of the black and white image areas (Equation 3.7). Because β is zero when the scale is zero, the

normalization then starts at unity. Drawing from the qualitative argument presented in section 3.2.3, as scale becomes large, all probabilities become equal, likewise, as scale becomes large, the normalized determinant becomes 0. A sample normalized determinant *N* is shown in Figure 3-13 for the alternating vertical bar pattern. Notice that *N* tends toward 0 as scale becomes large. Figure 3-13 has similar features to Figure 3-12 whereby the curve maintains decaying sinusoidal behavior and similar periodicity.

$$N(l) = \frac{\alpha_{ww}(l) * \alpha_{bb}(l) - \beta(l)^2}{\alpha_{ww}(0) * \alpha_{bb}(0)}$$ Equation 3.7

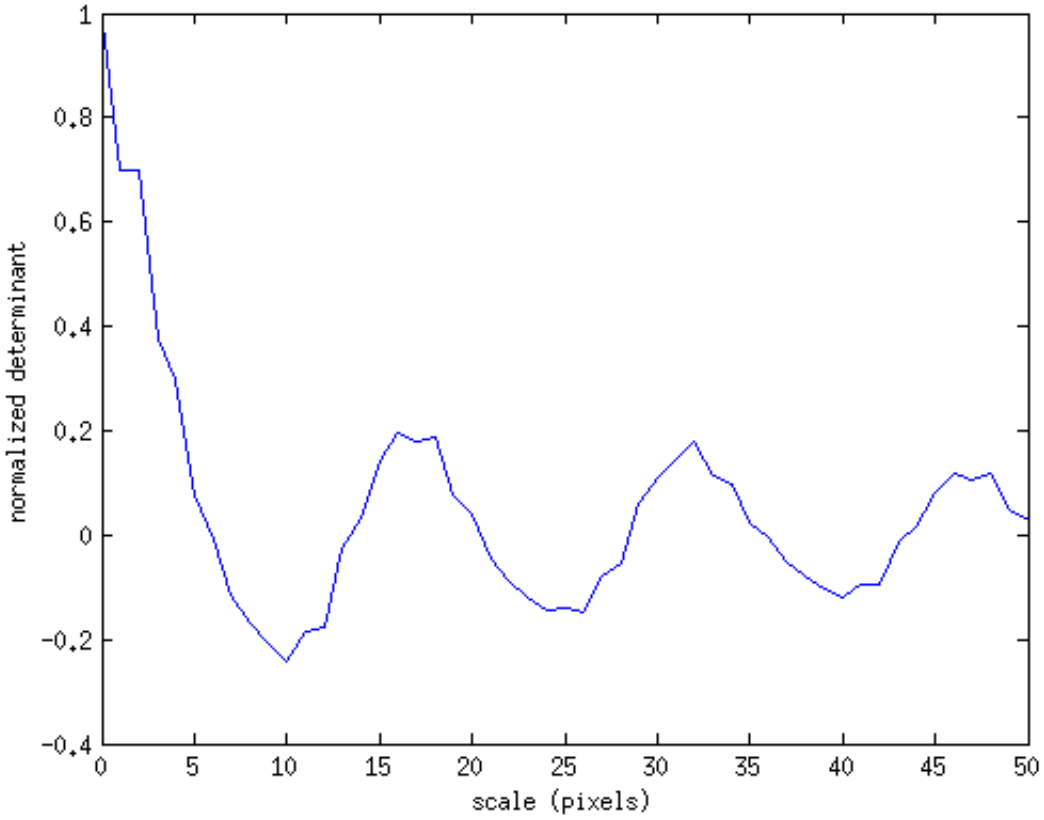

Figure 3-13: Sample normalized determinant

This analysis reduces 4 unique probability curves to a single normalized determinant curve.

3.2.6. Non-binary Images

All development of the modified random walk has been specifically for a binary image space. Binary images represent a significant reduction in the available information compared to a grey-scale or color image space. Ostensibly one way to rectify the modified random walk would be to directly include other transition probabilities. For example, consider the extension to an image space which has p possible values: $[c_1, c_2, c_3, \ldots, c_p]$. The fractal-space domain would have p^2 corners to represent all of the two-value combination possible pairings of the line segment endpoints. Finally, all of the transition probabilities could again by a first order approximation through the modified random walk. These combinations are presented in a probability matrix as shown in Equation 3.8. Here α_{ij} is the transition probability for a c_i to c_j combination.

$$P(l) = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \ldots & \alpha_{1p} \\ \alpha_{21} & \alpha_{22} & \ldots & \alpha_{2p} \\ \vdots & & \ddots & \vdots \\ \alpha_{p1} & \alpha_{p2} & \ldots & \alpha_{pp} \end{bmatrix} \qquad \text{Equation 3.8}$$

There are two main problems with extending binary images to color images in this manner. Older 8-bit images (such as the ones from the older FERET database) have at most 256 pixel values (c_i) and transitition probabilities. A probability matrix method is tractable because at most 256 pixel values (c_i) could be selected for analysis. However, modern electronic images often have 24 million distinct pixel values, meaning a probability matrix might have a total of

$576x10^{12}$ elements. Because the proposed algorithm must be quick, such extremely large matrices are computationally intractable at this time.

The second problem arises from the normalization scheme already presented (Equation 3.7). Without spartsity, the determinant of a large matrix is difficult to compute. Even if a determinant could be found, there is no obvious extension of the normalization.

Instead of considering all of the possible pixel value combinations, one could reduce a multi-valued electronic image to a binary space through some form of preprocessing. Stoffa and Deal determined the binary value of a particular pixel by calculating the average of that pixel's neighbors in the source image (usually a band of about 9 pixels by 9 pixels). Then, if the pixel of interest was below the average, it was rounded to 0 and similarly to 1 if it was greater than the average (1), (2). This method resulted in only one binary image space per electronic image file. For a subject like a fingerprint, only one binary image space per subject is logical. Fingerprints are deternied by a ridge or valley construct in real practice.

However, subjects such as faces are very complex geometrically. It was difficult to isolate a geometric feature of face which could be represented in binary. Ideally then, an image-space representation of a face would account for this complexity. Instead, a series of binary image spaces were associated with a singular electronic image. This series of binary images was generated by selecting a vector of pixel values which are referred to as contrast values: $[c_1, c_2, c_3, \ldots, c_p]$ where p is the desired number of contrast values. These c_k need not be uniformly distributed between c_1 and c_p, but p should not exceed the number of possible pixel values for the original electronic image. To create a particular binary image space Ω_I^k for a particular grey-scale electronic image Ω, all of the pixels in the original image were compared to c_k. All pixels greater than c_k in the original image were set to the value of 1 in the corresponding

pixel location in the binary image. Conversely, values below c_k became 0. This process is demonstrated by the pseudo-code in Figure 3-14.

```
for i  = 1, 2, 3, …, m
   for j = 1, 2, 3, …, n
      if Ω(i , j) >= c
     Ω_I(i , j) = 1
      else
      Ω_I(i , j) = 0
      endif
   endfor
endfor
```

Figure 3-14: Pseudo-code to create a binary image space from a grey-scale source image

Figure 2-1 shows an original electronic image already in grey-scale, and Figure 3-15 shows a series of binary image spaces created using the method described on the original image. While this derivation has been limited to source images in grey-scale, it can be extended directly to any group of color bands (RGB). Each of the separate color channels could be used to produce a series of binary image spaces. Finally then, the modified random walk can be used on each binary image space in series to create a normalized determinant which is a function of both scale and contrast level.

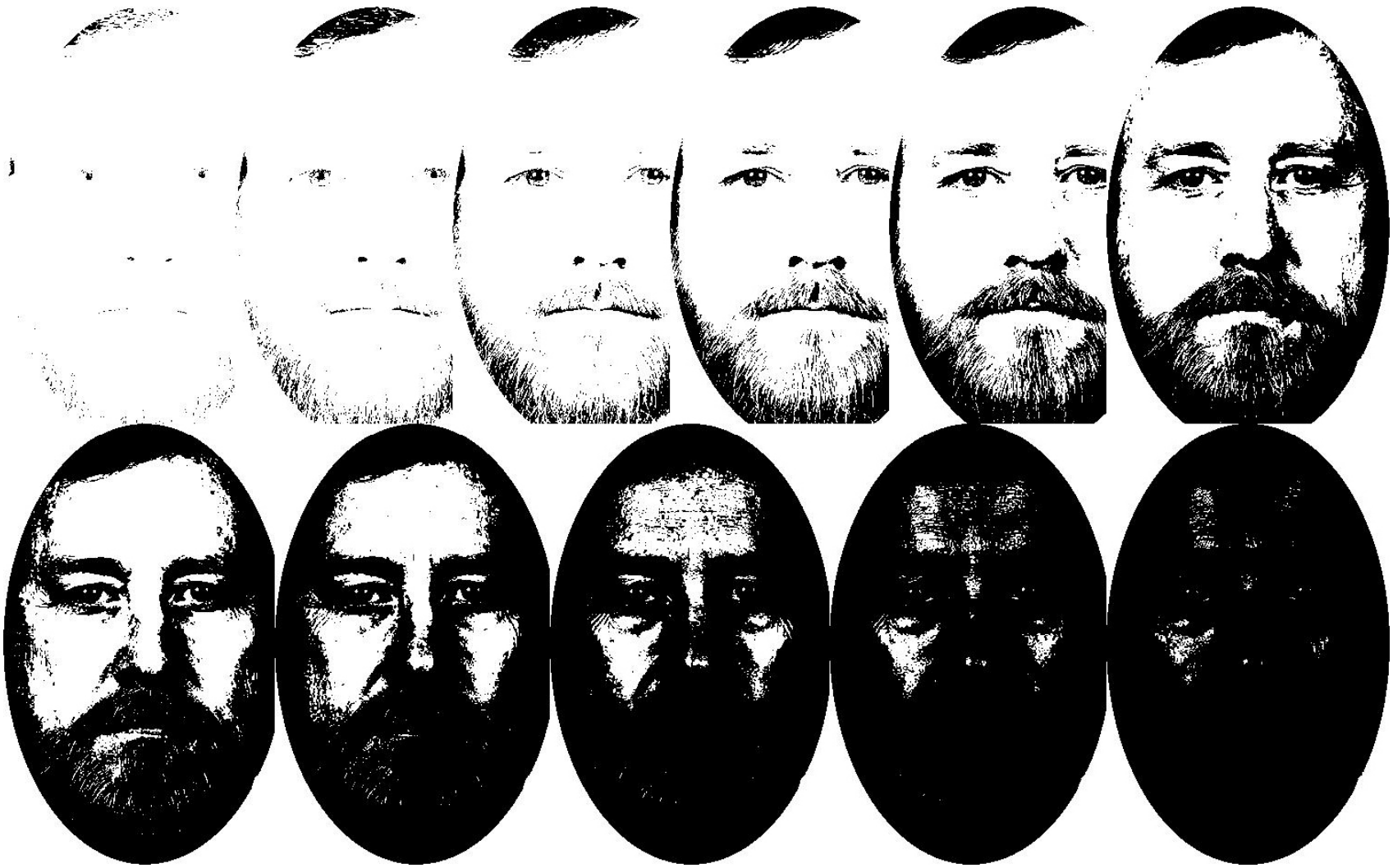

Figure 3-15: Series of binary images created from one source image

3.2.7. Summary

The previous three chapters of this document presented a basic introduction to facial recognition algorithms (or technologies), a background survey of other methods, and concluded with a detailed explanation of the algorithm proposed. In the background section, each method documented was found to be inadequate for computational speed requirements and/or ease of use. The new method is computationally quick to enroll a new face image into a database. However, no mention of how one face is compared to another is made.

This chapter presented the development of an algorithm and a normalization scheme, full-filling goals one and two set forth for this dissertation. The next chapter will explain the mathematical solution and outline the specific features of the scale spectrum curves.

4. Arc-Fraction of a Circle (AFC)

One goal of this work was the development of a rigorous procedure, which did not rely on the random walk, to explain the scale spectrum so that all features of the curves could be understood. Consider what it means to allow the random walk to repeat for an infinite number of iterations for one scale (aside from the basic problem that non-terminating instruction sets are by definition not algorithms). Essentially, every pixel would be selected and every value of Θ between 0 and π would eventually be selected for each pixel. Instead of a line segment of length l at one angle, picture a circle of diameter l centered at a particular pixel in the image (the net effect of choosing all the possible angles sweeps out a circle). The circumference of this circle then contains all of the endpoints of the line segment which would normally have been selected one-at-a-time. The eventuality of selecting all pixels in image space requires that such circles be created for each pixel in image space.

4.1. Implementation

From the circle being created at each point in Ω_I, the transition probabilities at that point can be calculated exactly. In Figure 4-1 a circle has been drawn over a local portion of some Ω_I, which is a vertical bar pattern of alternating 10 white pixels and 10 black pixels. The circle's center (origin of the red coordinate axes) can be thought of as the randomly selected point from the random walk (but remember that AFC will eventually select all points in image space).

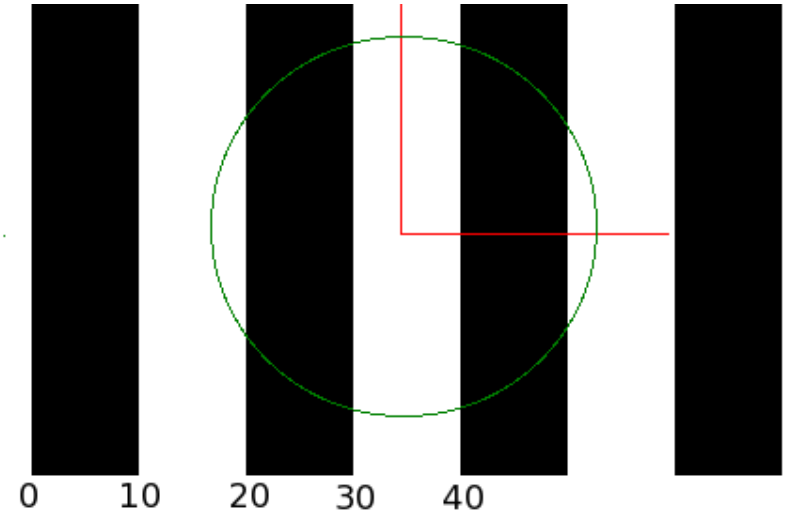

Figure 4-1: Circle overlaid on pattern (note: the red lines representing a local coordinate system to this selected point have been added for clarity)

The transition probabilities can be calculated exactly by marching around the circumference of the circle and checking each possible combination of endpoints. (Imagine starting on the "x" axis in Figure 4-1 and walking around the circle's circumference while marking all end-point combinations with each step.) It is easier to calculate each transition probability by determining the portion of the entire circle's circumference a particular paring occupies. The portion is a fractaion of the circle's arc (hence the name). A pictorial explanation is given in Figure 4-2. Here some of the various bands have been marked and numbered.

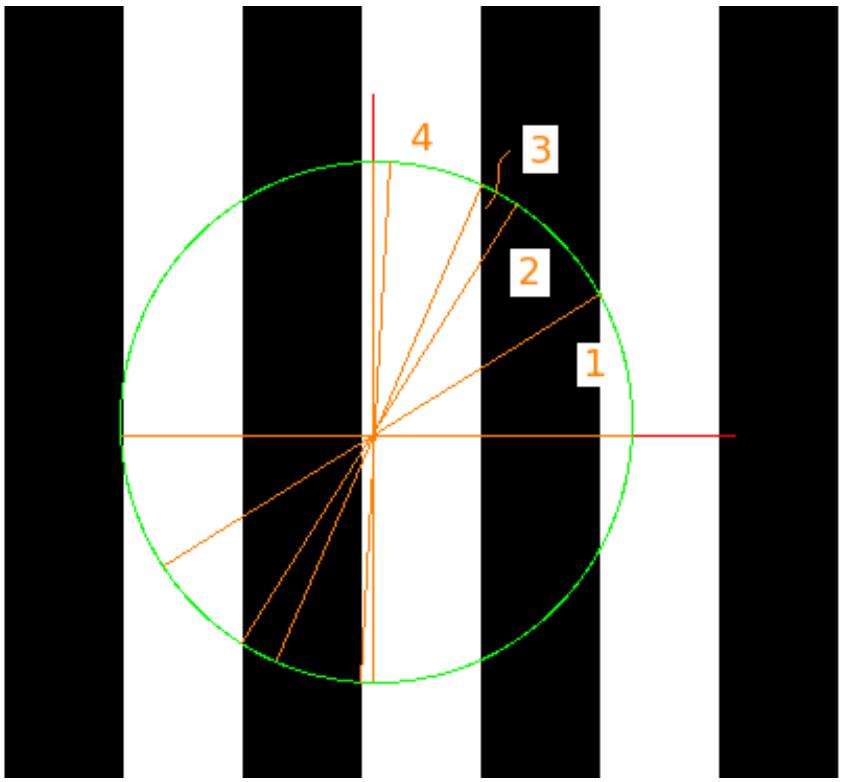

Figure 4-2: Circle with bands marked

Consider band 1: all of the line segments which can be created in it would have end pairings of white –white. Similarly, in band 2 all of the line segments would have end pairings of black-white. In band 3, all pairings would be black-black, and finally in band 4, white-black. This indicates, the probability for a certain pairing, i, can be calculated by:

$$\alpha_i = \frac{\sum_j^m \theta_{ij}}{2\pi} \qquad \text{Equation 4.1}$$

where θ_{ij} are the angles which sweep out the m bands which correspond to end-pairing i.

Essentially, the transition probabilities for a particular point (center point of the circle) in image space can be found exactly. This is unlike the Random Walk method which yields an estimate of a set of transition probabilities for an entire image space. Now one must consider the limitations of this particular example. The vertical bars are only 10 pixels wide, which means

that the combination of one black and one white bar is 20 pixels. This means that only 20 unique circles (per scale) can be created in the image because the image is invariant vertically and repeated horizontally. There are a small number of unique sets of transition probabilities per scale for this contrived image space, but the number of unique sets of transition probabilities (circles) may number as high as quantity of pixels in image space provided a pattern in image space possess no repeated features.

To return to the global nature of the scale spectrum generated through Random Walk, the sets of transition probabilities for an entire binary image are averaged together for all of the points (or the sub-set of points required to completely describe the image space). For example, if an image was 100 x 100 pixels in size, the total number of individual pixels (and arc-fraction based transition probabilities to be averaged) would be 10^4 per scale. Obviously, this becomes computationally expensive quickly for image spaces which cannot be simplified like the vertical bar patterns.

Finally, to create a scale spectrum, the scale (diameter) of the circle is allowed to vary. Similar to the Random Walk method, at zero scale the AFC produces a count of black and white pixels because a circle with no diameter is a point. The pre-fractal also tends towards grey-space as the scale becomes large. A rigorous examination through AFC can further explain this behavior.

A given image feature (say the repeated vertical bars) has a certain "size" associated with it—as already stated, 10 pixels (Figure 4-1). As the diameter of the circle becomes large with respect the size of the image feature, the number of repetitions of the feature incased in the circle increases, and the circumference of the circle is less likely to intersect a particular feature as the center point of the circle moves throughout image space. Imagine if the green circle in Figure 4-2

grew much larger while allowing the vertical bars to remain constant. The number of black/white bars the circle intersects would increase. At some scale, the circle becomes so large the each individual vertical bar (or image feature) intersects only a small fraction of the circumference of the circle. At that point, the numbers of black and white pixels on the circumference of the circle become close to equal (Figure 4-3).

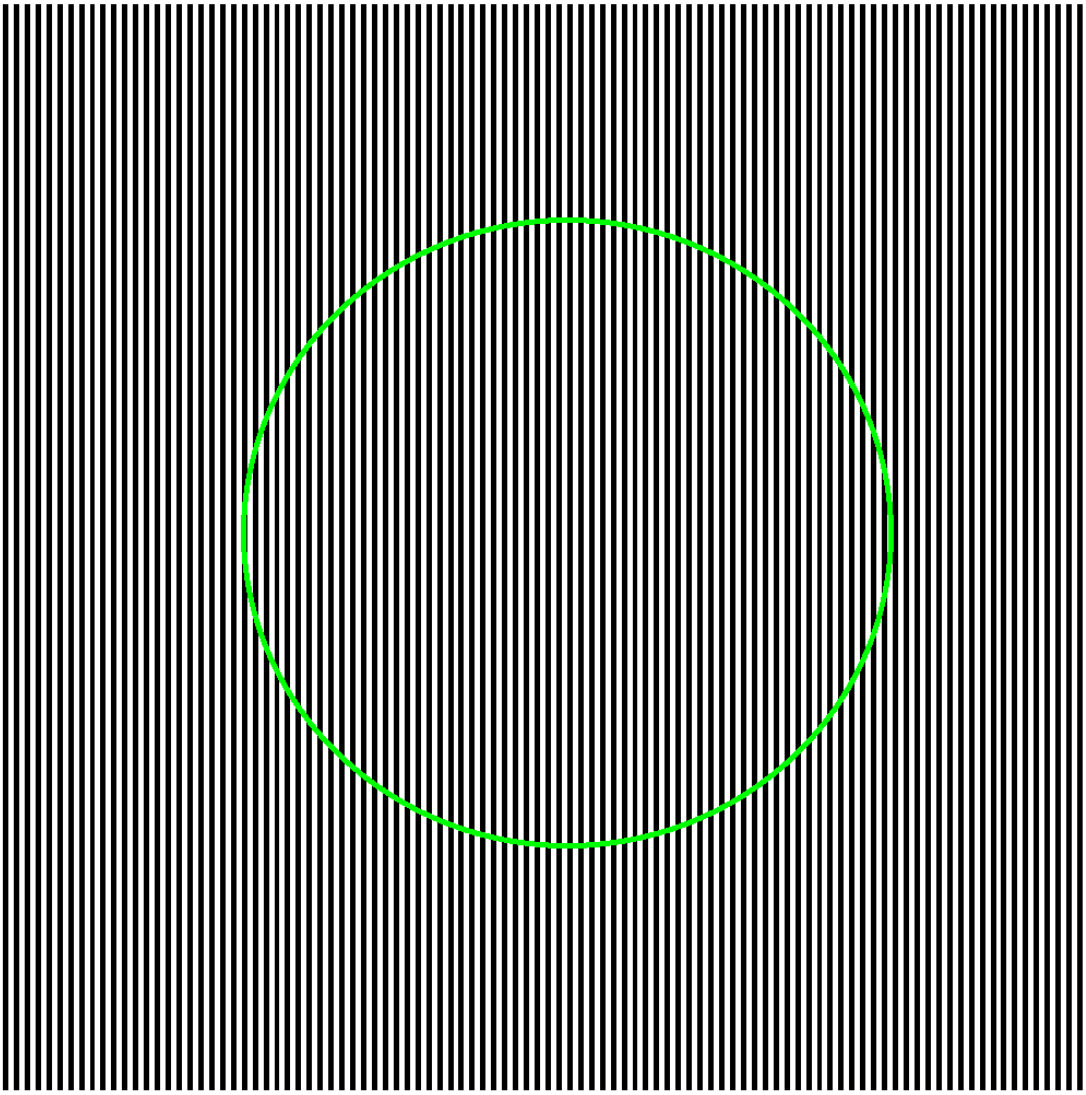

Figure 4-3: Large circle on vertical bar pattern (note that the circle has been bolded so it is easy to see)

When this occurs, no transition probability is any more or less dominant. In short, all probabilities become equal. Think back to the way in which the fractal was created with the Random Walk. Each corner was chosen based on a particular pairing of black and/or white. If all corners where equally probable, the number of points selected in that quadrant becomes the same as any other quadrant. In doing so, the fractal looses definition and turns to grey space.

As a means of comparison (and validation of the random walk), the AFC analysis of the vertical bar pattern (from Figure 3-8, not the 5-5 pattern being using in the present narrative) is presented in Figure 4-4 for the entire scale spectrum. The AFC was run with 25 scale values between 2 and 50 because of the massive computational requirements. Figure 4-5 shows the normalized determinant for both the AFC and the random walk. The slight variations between the two normalized determinants can be easily be explained by the rounding errors associated with working in discrete space. In the case of AFC, a discrete circle can be difficult to create at small scale.

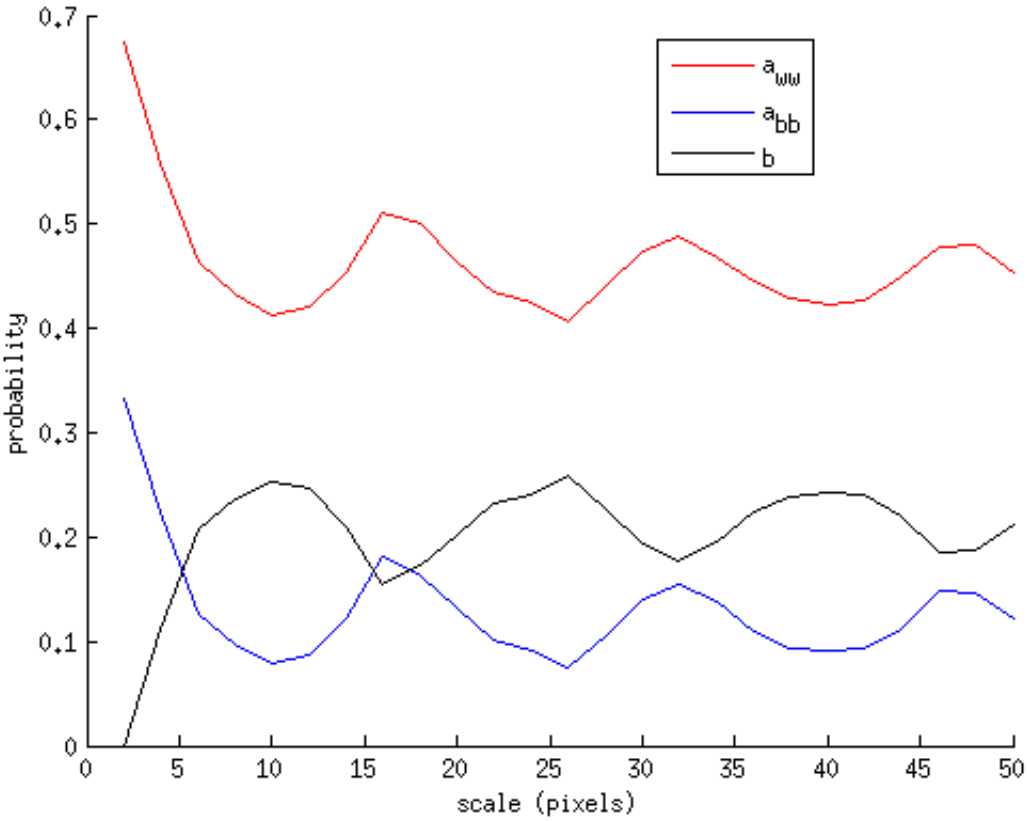

Figure 4-4: AFC generated scale spectrum

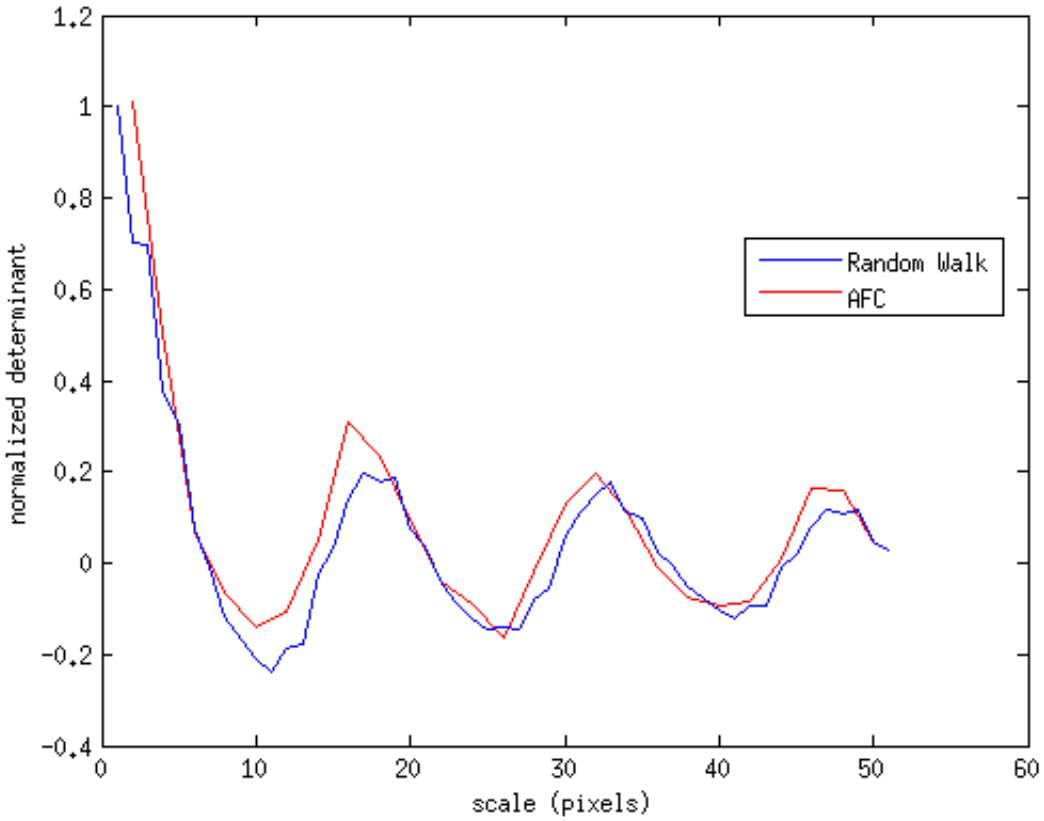

Figure 4-5: AFC and Random Walk Determinants

From Figure 4-5, the Random Walk solution and AFC normalized determinants lie very close to each other. Because the AFC curve was developed through a rigorous mathematical explanation, it validates the random walk curve. AFC has much greater computational requirements compared to the random walk; as such AFC will not be used directly to compare faces. However, it does present a rigorous means of validating any source code developments for the random walk and explaining why the normalized determinants (and scale spectra) take the shape they do for a particular pattern.

4.2. Analytic Solution

This simple 5-5 pattern used in 4.1 allows for a rigorous explanation of the shapes of the scale spectra curves. Figure 4-6 gives the scale spectra for the 5 pixel vertical bar pattern (Figure 4-3). In the example, the repeating bars were 5 pixels wide. This distance will be referred to as the motif width, U. The distance of the center point of the circle from the bar interface left of the center will be referred to as the offset, d (Figure 4-7). The number of possible values of d is $2U$ because $2U$ gives the number of bars required for the pattern to repeat horizontally. In the example used, the width of each bar was 5 pixels, which means the number of unique circles (values of d) was 10, as already mentioned. Finally, the diameter of the circle (scale) continues to be l.

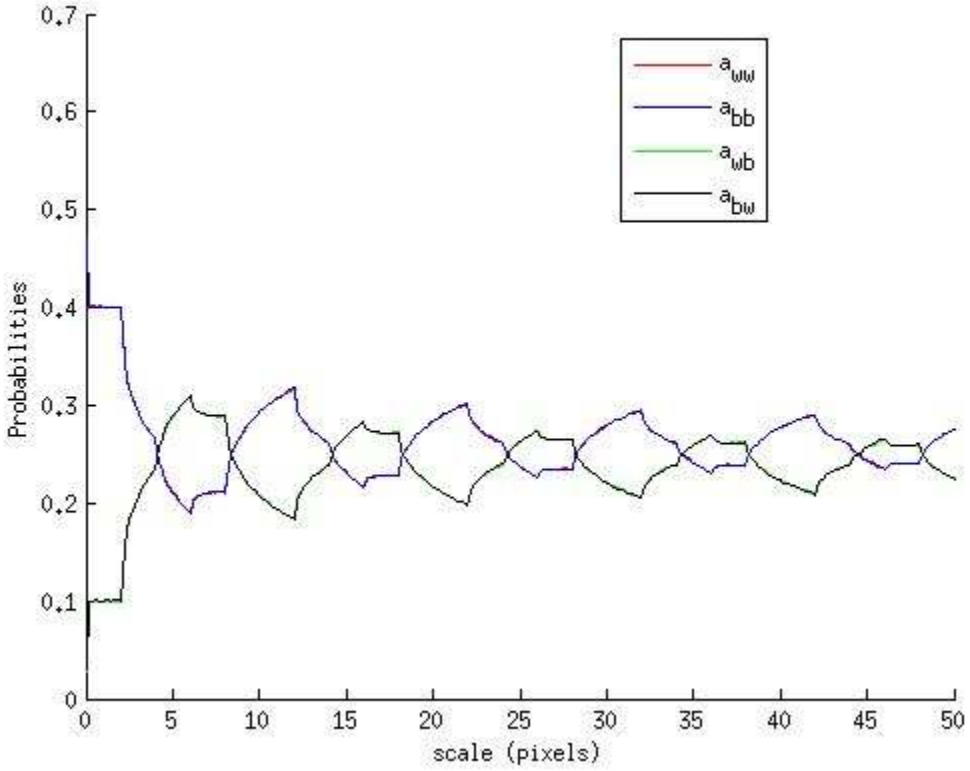

Figure 4-6: Scale Spectra of 5-5 pattern

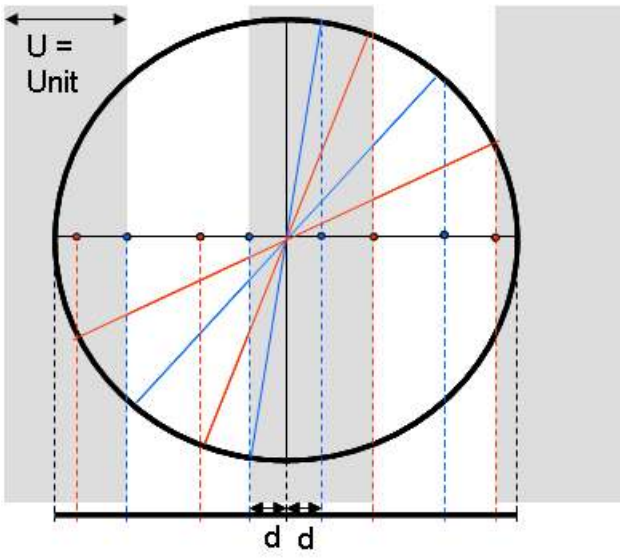

Figure 4-7: Example of circle with labeling

The horizontal axis of symmetry of the circle and bar pattern allows the rigorous model to be constrained to the first and third quadrants of the circle. In Figure 4-7, the offset from the bar interface must be less than one-half *U*. Once this distance has been specified, the remaining distances from the end-points of the arcs to the horizontal axis of the circle can be mapped in the following manner:

1. Reflect *d* about the vertical axis using symmetry properties of pattern and circle (Figure 4-7)
2. Remaining distance on initial vertical bar must be $U - 2d$ (Figure 4-8 a)
3. The distance from the center of the circle to the right-hand side of the first bar must be U-d. Using the same symmetry as step 1, the distance to the next left-hand point on the horizontal axis must be a total of U-d. Since *d* units have already been removed offsetting the circle from the bar interface, the distance from the next intersection point must be U-$2d$ between the interface and the point

on the horizontal axis (Figure 4-8 b, with a rectangle added to illustrate the symmetry being used).

4. Remaining distance on current bar must be U-(U-2d) or 2d.
5. Reflect 2d distance to right hand bar using the same logic as step 3.
6. Continue repeating steps 3, 4, and 5 until the circle is completely broken into all unique arc fractions (Figure 4-8 c).
7. The final section will be the difference between l / 2 and the sections already broken. In this example, l / 2 is a multiple of U, so the remaining section is d wide.

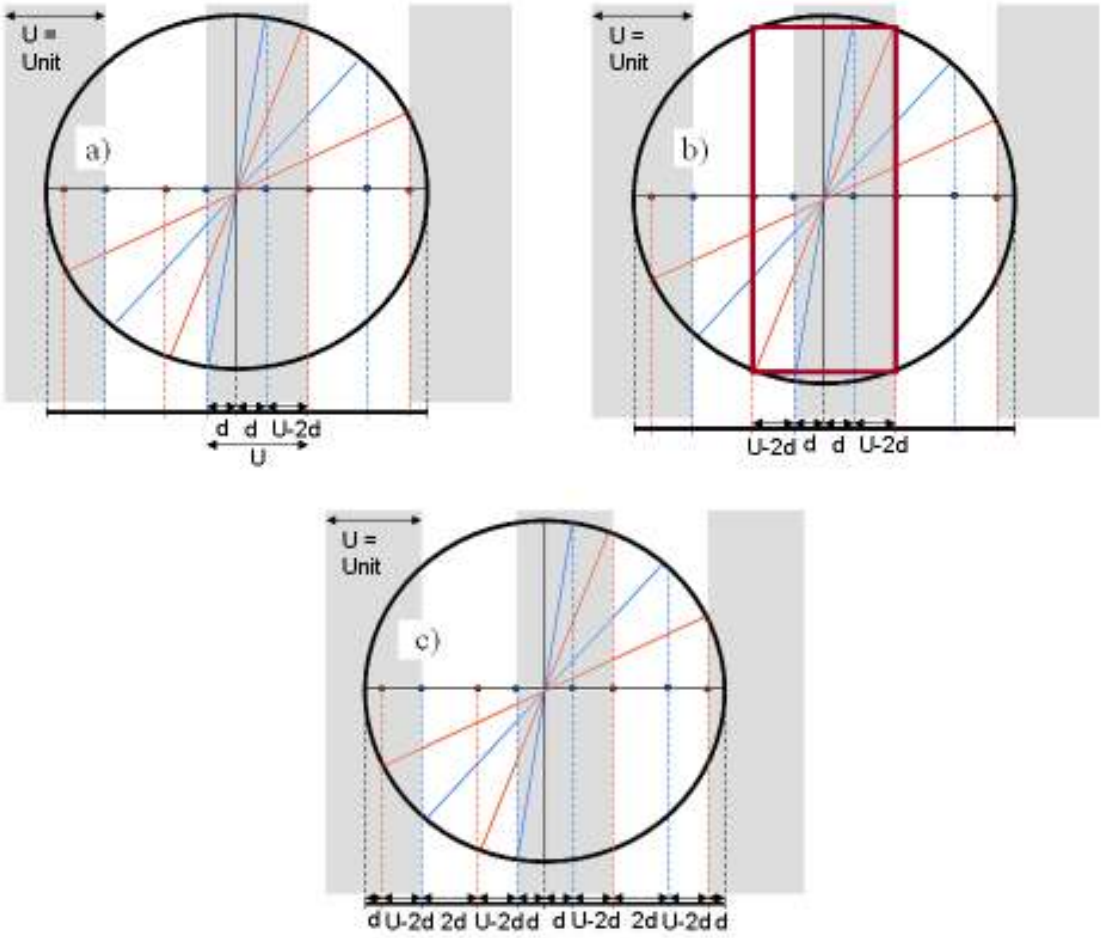

Figure 4-8: AFC mapping of circle to vertical pattern

Once the circle has been completely broken into arc fractions, a table of the horizontal distances and end-pairings was created. Using this table, the various probabilities for a certain location and diameter can be calculated directly. Table 4-1 starts with the center of the circle and works outwards. Note, this table is only valid for *d* which are less than ½ *U* and the "distances" are measured as the addition from the previous entry. Table 4-1 gives a general example and shows the repetition of the pattern. For a particular scale, the arc fractions will be completely mapped when the sum of distances becomes *l* / 2. Essentially, rows may be added or subtracted in order to map circles of varying diameter. Reading Table 4-1 from left to right yields the end-

pairing of a particular arc fraction. Table 4-2 gives an example of this application for $d = 2$, $U = 5$, and $l = 26$. Note that the Cumulative Sum adds up to $l / 2$ as expected.

Table 4-1: Demonstration Pattern for AFC for $d < ½ U$

Right-Hand Color	Left-Hand Color	Right-Hand Distance	Left-Hand Distance
B	B	d	d
B	W	U-2d	U-2d
W	W	2d	2d
W	B	U-2d	U-2d
B	B	2d	2d
B	W	U-2d	U-2d
W	W	2d	2d
W	B	U-2d	U-2d
B	B	2d	2d
B	W	U-2d	U-2d
W	W	2d	2d
W	B	$l / 2$ - sum of column	$l / 2$ - sum of column

Table 4-2: Example application (Note that table has been shortened to applicable scale)

Right-Hand Color	Left-Hand Color	Right-Hand Distance	Left-Hand Distance	Cum. Sum.
B	B	2	2	2
B	W	1	1	3
W	W	4	4	7
W	B	1	1	8
B	B	4	4	12
B	W	1	1	13

Using basic trigonometry, the angles which sweep out the individual sections were found starting with the first quadrant, horizontal axis, then moving around the circle. Table 4-3 gives the arc angles for the same configuration as Table 4-2. Finally, the probabilities can be found using Equation 4.1 by substituting π for 2π as only one-half of the circle is needed in this example.

Table 4-3: Angles of Arc Sections

Section Number	Combination	Angle Swept by Section
1	B-W	34.571
2	B-B	12.214
3	W-B	2.403
4	W-W	6.589
5	B-W	0.842
6	B-B	33.381

In the case where $d > U / 2$ (Figure 4-9), the offset distance is switched so that it is measured from the right-hand interface. In this case the color values of Table 4-1 simply switch order and the pattern may be used in the same way (as shown in Table 4-4). The arc angles may be calculated in the same way. Note that the only major change is the ordering of the black and white end colors.

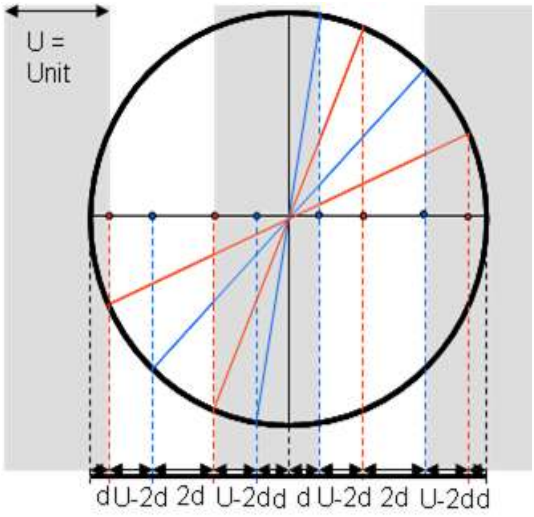

Figure 4-9: Offset greater than *U*/ 2

Table 4-4: Modified table for d > *U* / 2

Right-Hand Color	Left-Hand Color	Right-Hand Distance	Left-Hand Distance
B	B	d	D
W	B	U-2d	U-2d
W	W	2d	2d
B	W	U-2d	U-2d
B	B	2d	2d
W	B	U-2d	U-2d
W	W	2d	2d
B	W	U-2d	U-2d
B	B	2d	2d
W	B	U-2d	U-2d
W	W	2d	2d
B	W	*l* / 2 - sum of column	*l* / 2 - sum of column

4.3. AFC and the Scale Spectra

Using the analytic solution developed, the scale spectra of the white-white probabilities were plotted for each of the 10 unique points along the vertical bar pattern. For this study, point 1 was immediately right of the black to white interface. The points were numbered sequentially across the horizontal to 10, which was the point directly left of the next black to white interface. Figure 4-10 gives a plot of the white-white scale spectra for scales ranging from 0 to 50. The "average" curve is the average of all 10 curves.

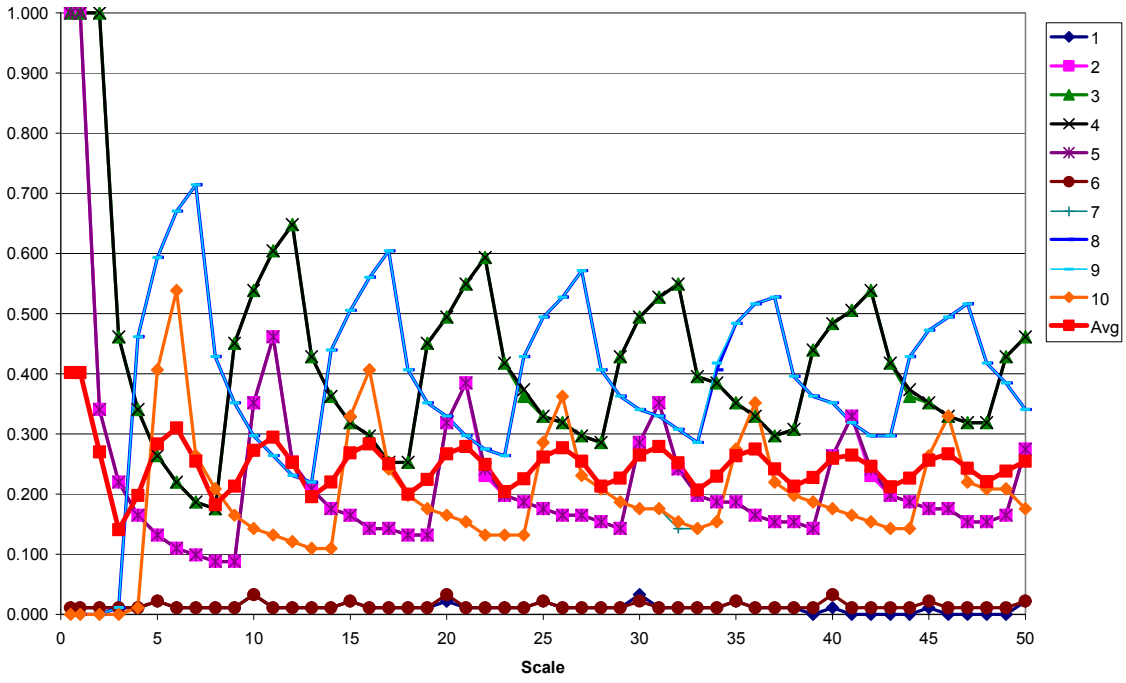

Figure 4-10: Ten point probabilities for white-white combination

Curves 1 and 6 were nearly zero across all scales because they were circles near the boundaries of black and white vertical bars. In these regions, the mixed probabilities were much higher. From there, curves 1 and 6 were treated as zero, and curves 3, 4, 8, and 9 (which have been reproduced in Figure 4-11) were considered. In Figure 4-11, it is easier to see that curves 3 and 4 lie directly on top of each other, as do 8 and 9. Curves 8 and 9 are almost exactly out of phase with 3 and 4. So when the average of all 10 curves was found, 8 and 9 and 3 and 4 nearly canceled each other out to a horizontal line. Figure 4-12 gives the resulting pairs which influence the resulting white-white scale spectra. It also has the average plot reproduced.

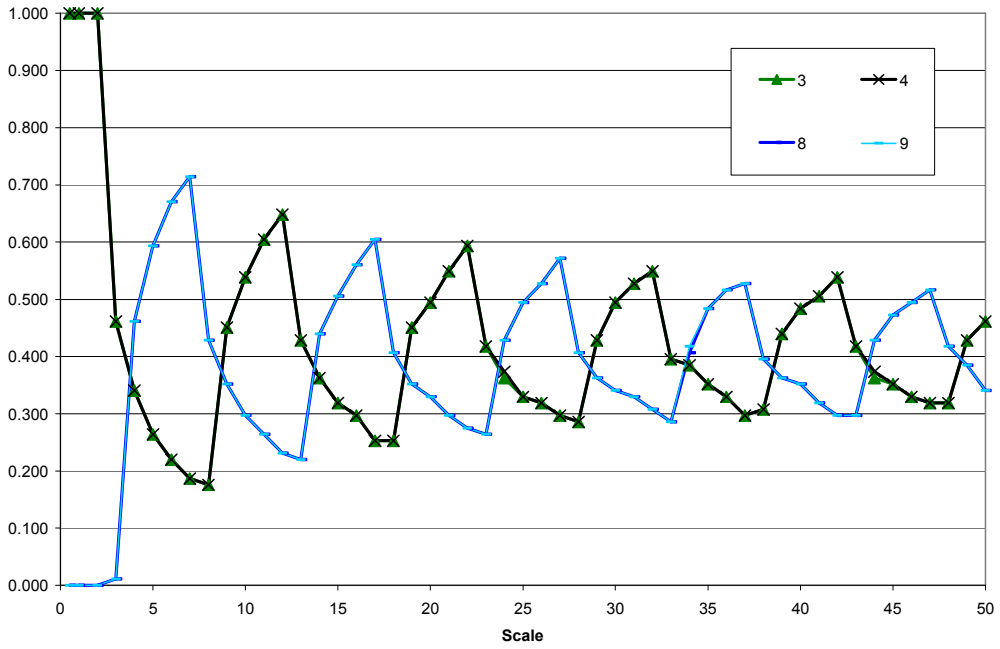

Figure 4-11: Curves 3, 4, 8, and 9 for white-white

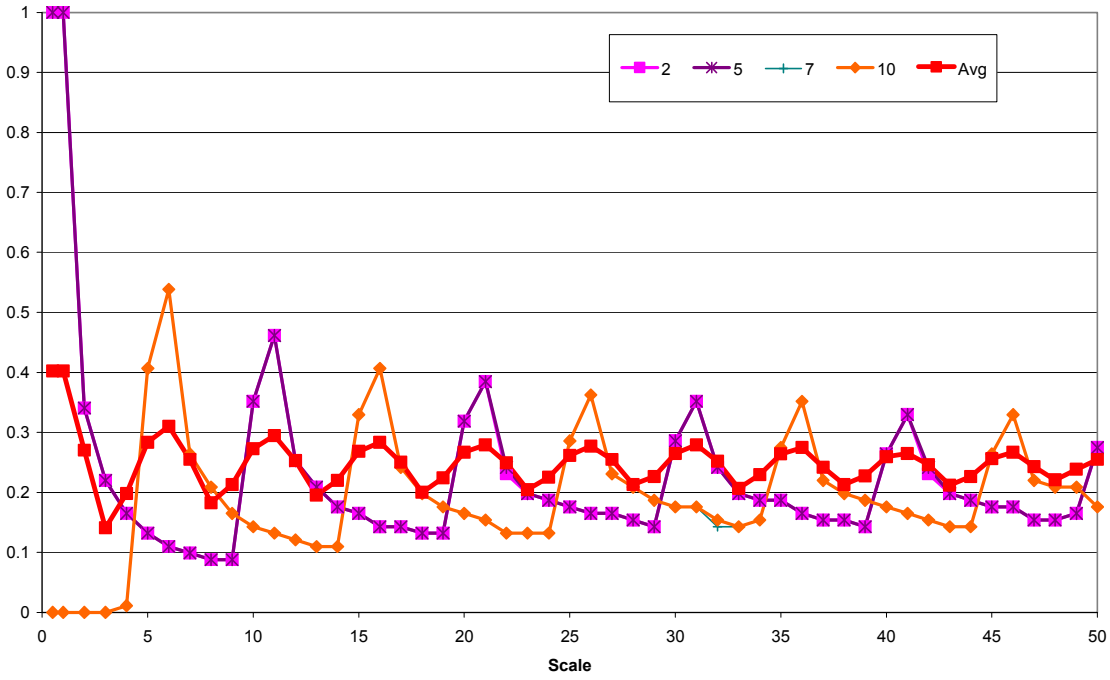

Figure 4-12: Remaining curves and average

When the orange or purple line pairs peak, the average curve peaks and when neither line pairs are peaking, a minimum occurs. The distance from one orange maximum to the next is 10 scales (the width of the repeating pattern). The same is true from purple maxima to the next purple maxima. The offsetting of the maxima for each line pair explains why the area under the scale spectra is 2.5 per wavelength instead of 5 (the bar width). Each individual line pairs would have a period twice as long as the average, due to the offset of the curves. This also holds true for the black-white curves and the black-black curves.

4.4. Combined Patterns

Until now, the patterns used have been contrived to only contain one motif. The new pattern is a combination of 5-5 vertical bars and 10-10 vertical bars (with equal areas for each pattern). The pattern is given in Figure 4-13. The resulting scale spectrum is given in Figure 4-14. The spectra is quite different than the previous vertical bar pattern spectra.

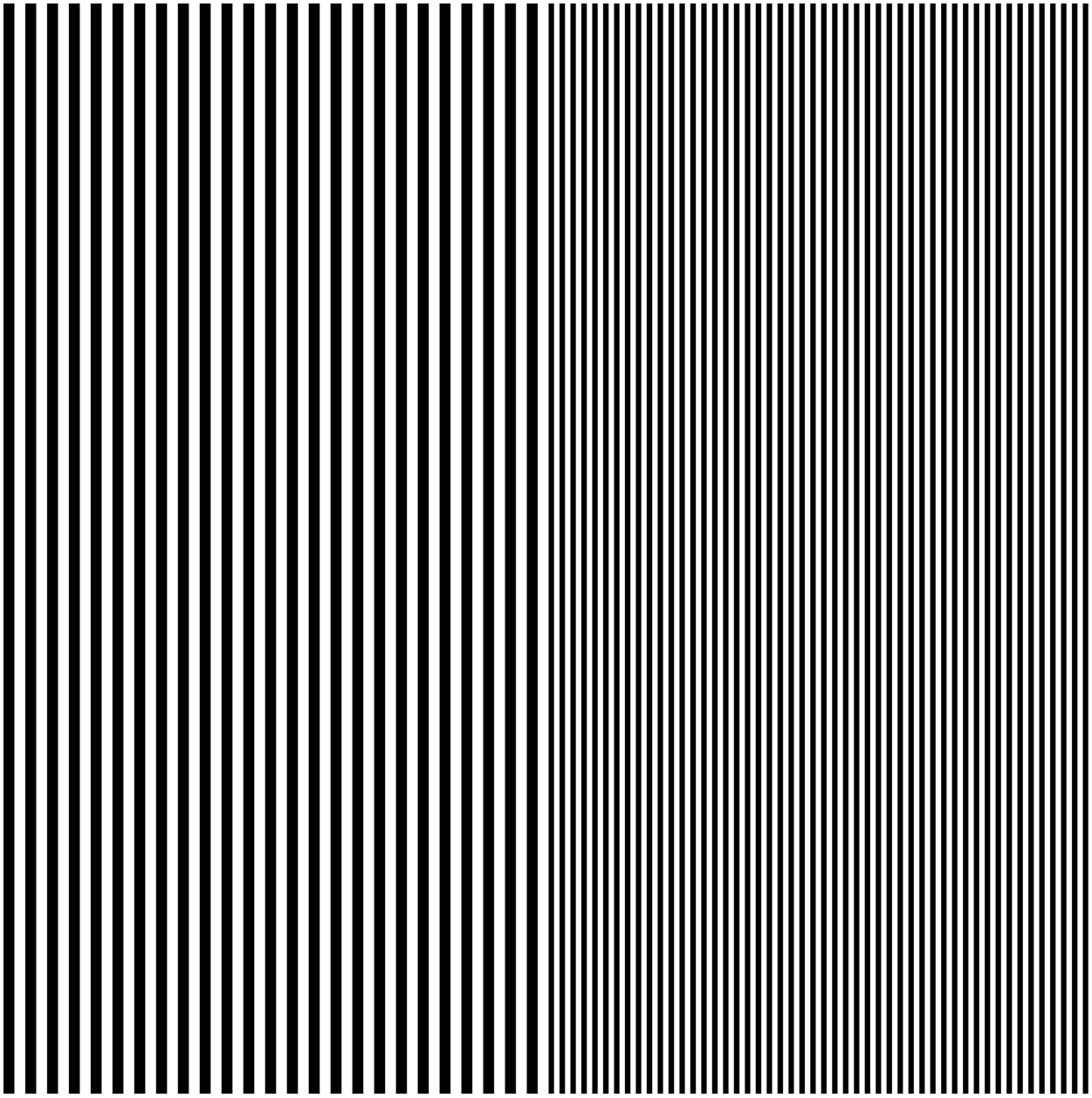

Figure 4-13: Combined pattern

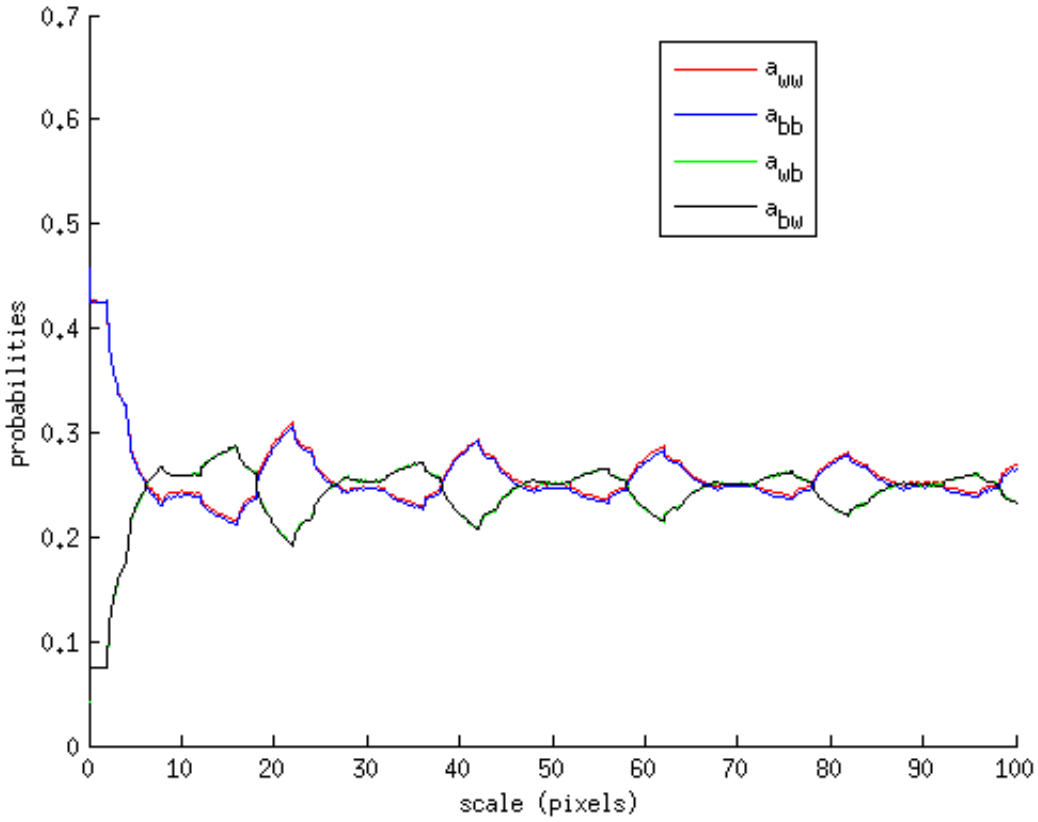

Figure 4-14: Combined pattern scale-spectrum

Unlike Figure 4-6, which shows a pseudo-sinusoid, Figure 4-14 shows the same peaking behavior at 20 pixels, but flattens out for 10 pixels in between. There also appears to be an inflection before the large peaks (present in both the white-white, black-black and mixed probability curves). Again, the mixed probability curves are direct inflections on the white-white, black-black curves.

Consider the vertical bar pattern generated from the combined scale spectrum (reproduced and zoomed in Figure 4-15). Far away from the interface of the two vertical patterns (Region II) in regions I and III, the scale spectra can be calculated using the analytic solution already presented. Any circle which includes a portion of region II cannot be predicted

using the analytic solution. As already presented, the resulting scale-spectrum for an entire image space is the average of all individual arc-fraction scale spectra from each point in image space. This means that a large portion of the scale spectrum for the combined image can be predicted with the analytic solution for regions I and III.

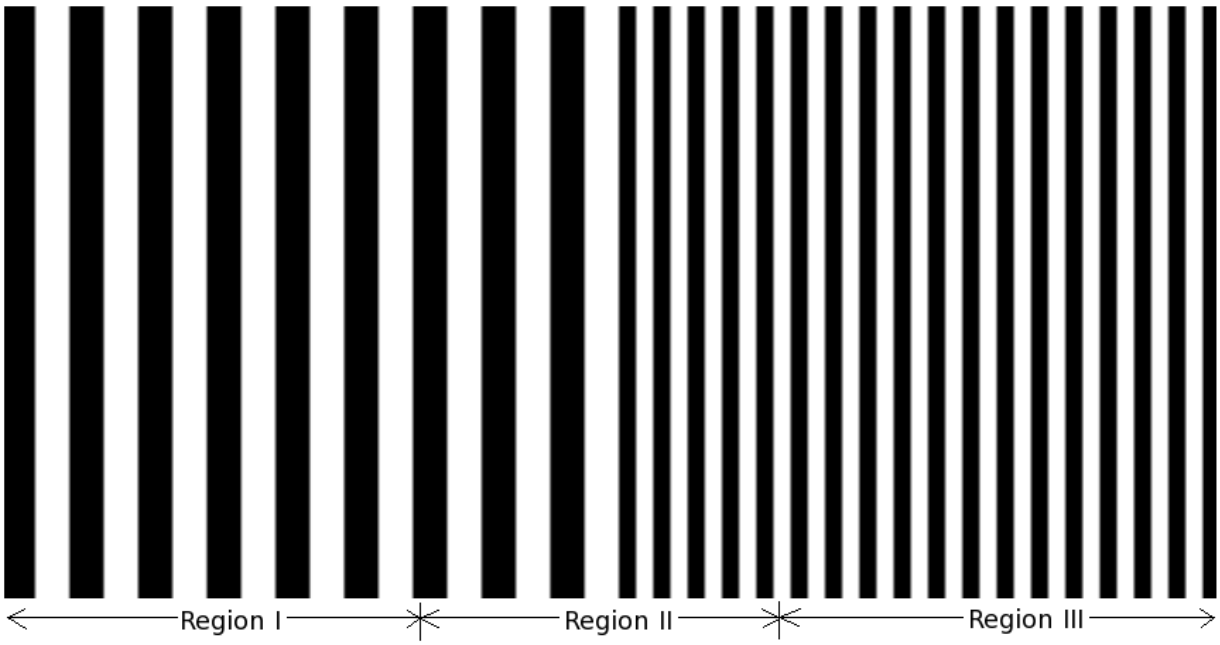

Figure 4-15: Combined pattern

Region II has no simple analytic solution (although one could be developed). But the width of this region is limited by the scale of the circle. Imagine sitting firmly in Region I, constructing circles at a fixed scale. As the circles move closer (say marching horizontally pixel-by-pixel), they will eventually touch the interface between the two patterns (exact center of region II). The same can be mirrored in region III. Effectively, then the width of region II can be limited to two times the scale. Figure 4-16 gives a pictorial example. The green and yellow

circles have the same diameter and are the result of approaching the interface between the two patterns.

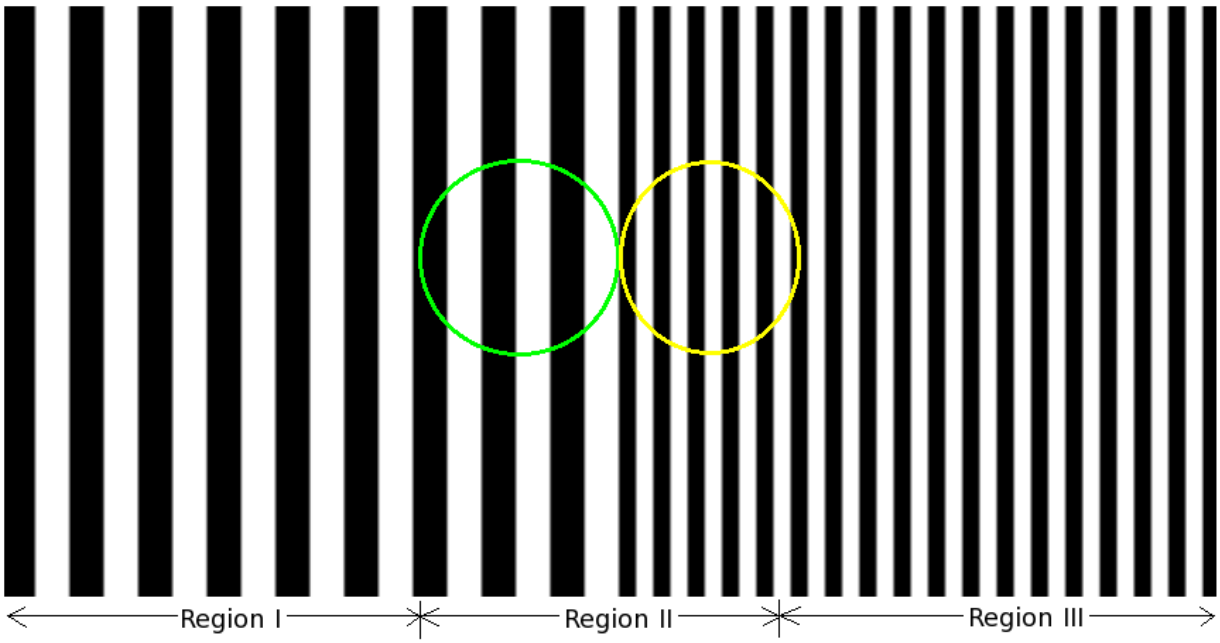

Figure 4-16: Sample domain with circles marked in Region II

For small scales (diameters) region II will also be small. Finally, the scale-spectrum for the combined pattern can be estimated by averaging (weighting by area) the spectra for regions I and III and ignoring the effects of region II, but only for small scales.

The white-white probabilities for the 5-5 pattern and 10-10 pattern are reproduced in Figure 4-17. As expected from the analytic solution, the 10-10 pattern has exactly twice the period of the 5-5. In Figure 4-18 the white-white probability from the combined pattern has been over-laid with the average of the two curves from Figure 4-14.

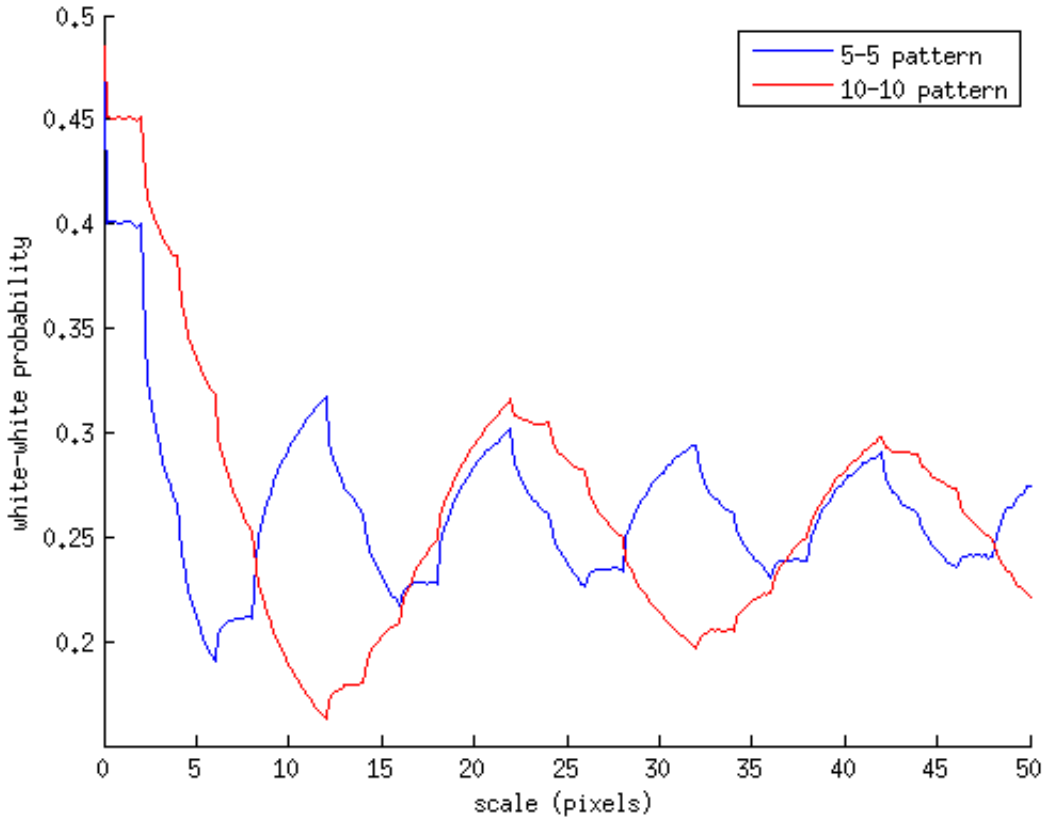

Figure 4-17: Reproduced white-white probability curves from 5-5 and 10-10 patterns

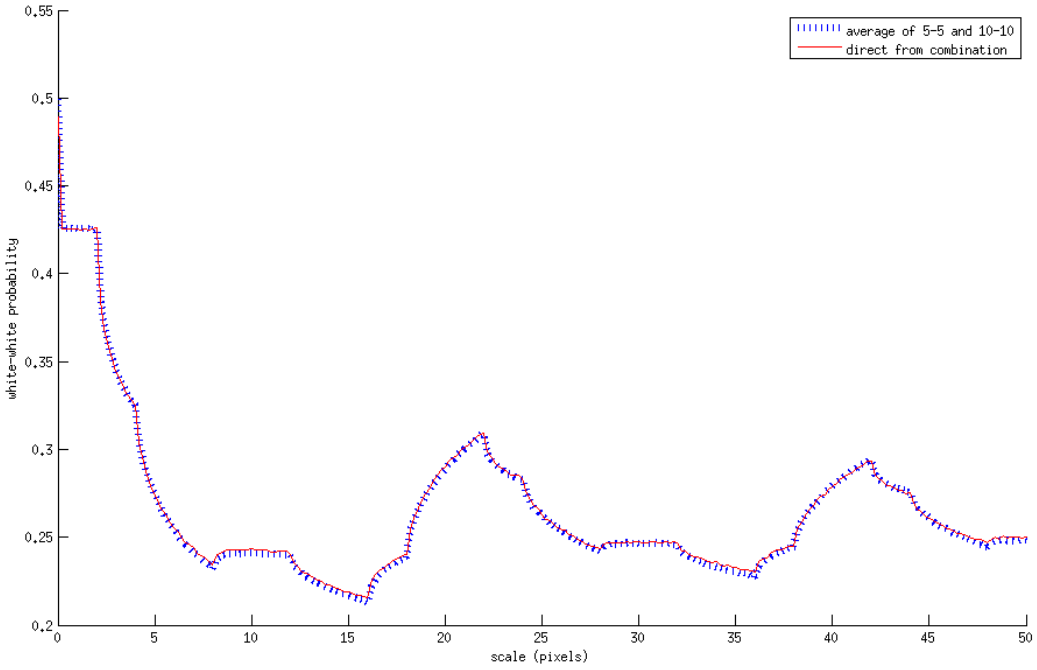

Figure 4-18: White-white average curve and curve directly from combined pattern image space

As expected, the two curves lie directly on top of each other (with a slight variance resulting from the interface). This means that any complicated pattern can be thought of as a collection (grouping) of smaller, simpler patterns (such as the vertical bar pattern). Also, the scale spectrum from any complicated pattern can be thought of as an average of all the scale spectra from the pattern's components. By knowing the scale spectrum of a particular image feature (say a human nose), the effects of that feature on the resulting over-all scale spectra can be removed from the spectrum.

This process of using the weighted averages can be used on any single probability curve of the scale spectrum or the normalized determinants. The normalized determinants for the 5-5 and 10-10 patterns are given in Figure 4-19. Again, the average of the two along with the normalized determinant from the combined pattern has been given in Figure 4-20.

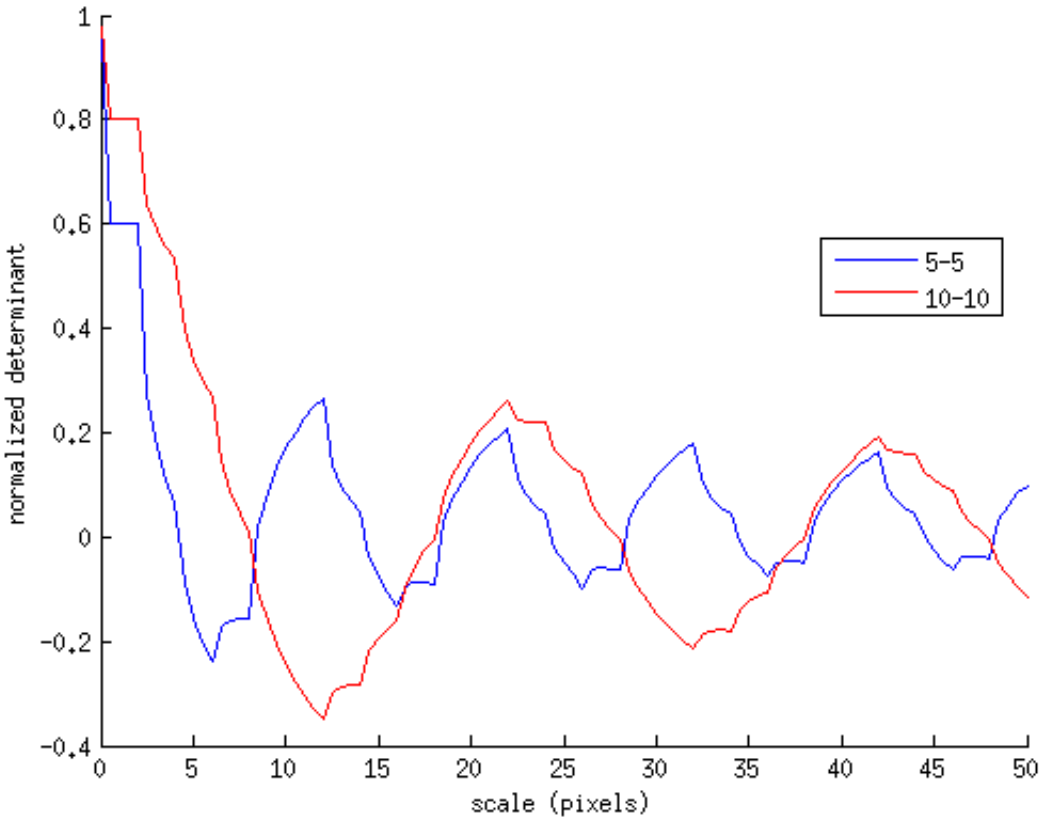

Figure 4-19: Normalized determinants for the 5-5 and 10-10 patterns

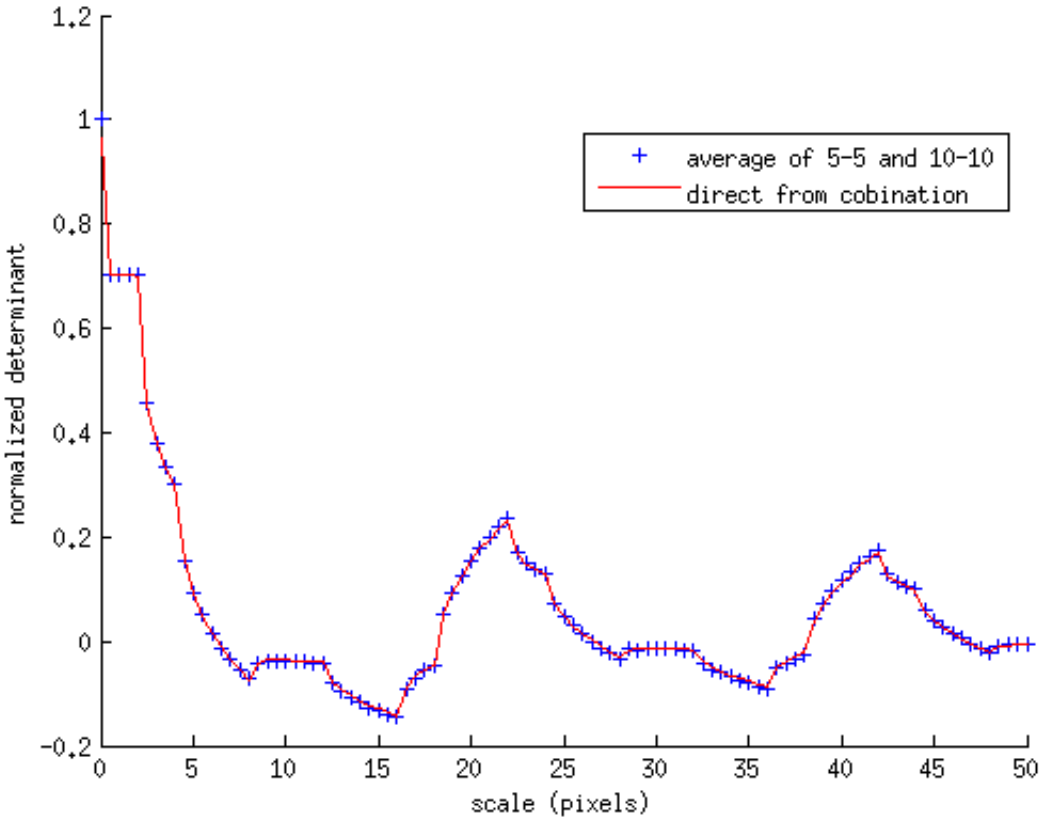

Figure 4-20: Normalized Determinant average curve and curve directly from combined pattern image space

An alternative use is removing the image space regions which have a negative impact on the analysis (say features common among a set of images where delineation is important). Essentially, if the scale spectrum for a region in image space was known prior to random walk analysis, it could be "subtracted" from the resulting scale spectra. The remaining spectra would then be comparable to an image space which didn't contain the particular feature.

In the case of human faces this is particularly useful because all human faces have approximately the same shape. The resulting scale spectrum (and consequently normalized determinant) contains information derived from the specific facial features (ex. the exact curvature of the nose bridge) and information from the basic face makeup of all humans (say the

approximate location of the mouth). In short, by computing the scale spectrum of an "average face," the features which are common among all humans can be removed from the scale spectrum for a particular face. This, in turn, aids in delineation among a database of scale spectra.

4.5. Summary

This chapter has shown the development of a completely rigorous method, the Arc Fraction Circle method for generating the scale spectra. Through this method, many of the features of the spectra curves have been explained. For the simplified case of the uniform vertical bar pattern, the scale spectra becomes tractable for hand calculations and simple algebraic formulations were presented for creating a scale spectra.

Image spaces are now understood to be simple (area-based) linear combinations of a series of scale spectra from smaller (simpler) image spaces. The extension of this logic is important for face comparisons. It allows for faces to be thought of as a data set of features unique to a particular subject (human being) and a dataset of features which are common to all humans. As such, the commonalities can be removed from the scale spectra (normalized determinant) to heighten delineation.

5. Experimental Parameterization

In this chapter, the explanation of how the algorithm will be specifically applied to faces will be detailed along with a parameterization study (goal 4) of the specifics of face comparison. These parameters fall into several possible configurations, all of which will be explored to find the most accurate method. The chapter will conclude with a setup of the computational experiments which will be used to determine the optimum configuration of the resulting GUI.

5.1. Database Construction

Before any analysis can begin, a database of digital photographs was selected for uniform comparison among different versions of the algorithm. As already mentioned, a sub-set of the FERET database was used. Specifically, 400 photographs containing only full-frontal views (no side-ways facing subjects) with limited facial expression change between subjects were selected from the FERET database. Each subject was represented by at least two different photographs in the database for comparison, though greater multiples were included (167 unique subjects). The full list of images selected from the FERET database is provided in Appendix A.

5.2. Pre-processing

As mentioned previously, the photographs in the FERET database contain much more than just a face—usually great amounts of background, torso, and other extraneous items (Figure 2-4). The first step of the pre-processing algorithm was selecting a face from a digital photograph.

In order to avoid the complicated issues regarding the boundary of the domain of the face in the digital photograph, the portion of the digital photograph to be used was arbitrarily limited to a definable portion of the image. In other words, the region of the face to be considered for random walk was easily definable in simple geometric terms. The portion selected included enough information such that one face photograph could be distinguished from another.

The portion used was ideally be independent of facial expression (like smiling) and changes in hairline. Since a human face is similar from side to side, only one half of the face was sufficient for identification. The region of the face which surrounds the eye is the area where the greatest of change in curvature of the occurs (Figure 5-1). As can be seen in Figure

5-1, there still are sections of the face which vary little from person to person (mostly the nostril(s), hairline, and photograph background) and as such are pointless to include. So, the final "shape" of the domain was chosen as an ellipse centered on the subject's left eye.

Figure 5-1: Portion of the face to be used

In order to construct such an ellipse the location of the subject's left eye must be known. As established by the third goal, the user selected both of the subject's pupils during the preprocessing. Knowing the Cartesian coordinates of both eyes (pupils) in image space allowed for several things. First, the angle of rotation (θ) of the face relative to the horizontal of image space was calculated by the slope of a line connecting the eyes (Equation 5.1, where *L* and *R* denote the left and right pupils and *x* and *y* are the coordinates of the eyes in image space). While the random walk is irrespective of face rotation (because the random walk is modeled by creating a circle), the selection of the facial region for analysis was dependent on face rotation (i.e. the ellipse was tilted to match the tilt of the subject's head).

$$\theta = \operatorname{atan}\frac{y_L - y_R}{x_L - x_R} \qquad \textbf{Equation 5.1}$$

Second, the locations of the eyes also defined a distance (specifically, the inter-pupil distance, d_p) in pixels which was relative to a particular image (Equation 5.2). This distance was

used in both creating the ellipse (ensuring the ellipse only contains the relevant pieces of the face) and defining the series of scale (l) used during the random walk.

$$d_p = \sqrt{(y_L - y_R)^2 + (x_L - x_R)^2} \qquad \text{Equation 5.2}$$

Finally, by knowing the inter-pupil distance and the angle of rotation, the equation of the ellipse which encompass the left eye was found from the general form (Equation 5.3), which did not account for any tilt of the subject's head. In Equation 5.3, the semi-major axis was oriented vertically (denominator of y term is larger than x term) and both the semi-major and minor axes were dependent on the inter-pupil distance. The axes were calculated based on percentages of tne inter-pupil distance (75% and 50%) to force the ellipse to "land" on the bridge of the nose and encompass portions of the forehead and cheek.

$$\frac{(y - y_L)^2}{\left(\frac{3}{4}d_p\right)^2} + \frac{(x - x_L)^2}{\left(\frac{1}{2}d_p\right)^2} = 1 \qquad \text{Equation 5.3}$$

To account for the rotation of the subject's head, the standard rotation matrix (Equation 5.4) was used by shifting the ellipse to the origin, multiplying by the rotation matrix, and then shifting back to the original location (Equation 5.5). This results in the coordinates of an ellipse which was rotated to match the rotation of the subject's face. These coordinates then were treated as the boundary of the region around the eye to be kept; any points outside of this region were removed from analysis (masked by pure green because no portion of a human face is likely to be pure green). Figure 5-2 gives a sample of a masked and trimmed image space for the same subject as Figure 5-1. Note the subtle rotation of the ellipse as well.

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad \text{Equation 5.4}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x - x_L \\ y - y_L \end{bmatrix} + \begin{bmatrix} x_L \\ y_L \end{bmatrix} \quad \text{Equation 5.5}$$

Figure 5-2: Sample masked and trimmed face ready for grey-scale conversion

The next step in the pre-processing then was to convert the image to grey-scale (and the series of binary images suitable for random walk analysis). The conversion to grey-scale was handled directly by the MATLAB Image Processing Toolbox according to Equation 5.6 where *R* is the influence of red color band, *B,* blue, *G*, green (33). Figure 5-3 gives a sample grey-scale conversion for the same image from above. Because the actual region of interest was only a subsection of the entire image space (Figure 5-2), the portion of interest will be referred to as Ω_S (*S* for subject's face), and any binary image which results from Ω_S will be referred to as Ω_I^k, which is similar to the notation used in the Algorithm Development chapter, except a superscript *k* has been added to denote that Ω_I^k belongs to a series of images.

$$\Omega_S(x,y) = 0.2989\,R(x,y) + 0.5870\,G(x,y) + 0.1140\,B(x,y)$$ Equation 5.6

Figure 5-3: Sample grey-scale conversion with masked background

Finally, the Ω_S region was converted to the series of binary images. The series of images were created as described before by selecting p contrast values uniformly between the average pixel value of Ω_S (Equation 5.7) minus two standard deviations (Equation 5.8) and plus two standard deviations. Equation 5.9 below gives the exact formulation for a particular c_k. Here, N is the number of pixels which are inside of Ω_S and i denotes a particular grey-scale value of a particular pixel in Ω_S. The reader is referred to Figure 3-15 for a pictorial example because Figure 3-15 was created in exactly this manner, except using an entire facial image. As p becomes larger, the resulting analysis also becomes better; however, the analysis for a particular image will take longer (more p, more iterations). The effects of varying p were measured during the research to determine an optimal value.

$$\overline{\Omega_S} = \frac{\sum_{i=1}^{N} \Omega_S(i)}{N} \qquad \text{Equation 5.7}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\Omega_S(i) - \overline{\Omega_S})} \qquad \text{Equation 5.8}$$

$$c_k = \overline{\Omega_S} + 2\sigma\left(\frac{k-1}{p-1} - 1\right) \quad k = 1, 2, 3, \ldots, p \qquad \text{Equation 5.9}$$

The preprocessing of an image was now complete. The image had been transformed from a multicolor electronic image file containing much extraneous information to a series of binary images containing mostly the facial region of interest which were suitable for random walk analysis (or processing). The images now contained a masked region which was to be ignored during processing. The next section will cover the specifics of the random walk implementation not yet covered by the generic discussions previously.

5.3. Processing (Random Walk)

The random walk portion of analysis was conducted as described in Algorithm Development, however, the specific values of scale have, thus far, been described in terms of pixel values only. In a similar manner to preprocessing, the maximum scale was defined by the inter-pupil distance to remove the effects of differeing numbers of pixels between the subjects' eyes. Here, the number of scales is *m* and all l_j will be defined by Equation 5.10. Effectively then, the scale values were evenly distributed between zero and 25% of the distance between the eyes. The maximum scale value was selected to provide adequate coverage of the random walk without creating circles which were larger than facial features (remember that the ellipse masked

region was only as wide as the inter-pupil distance). Again, (as with the number of binary images), the optimal number of scales was determined by an experiment.

$$l_j = \frac{j-1}{4\,(m-1)}\left(d_p\right) \quad j = 1, 2, 3, \ldots, m \qquad \text{Equation 5.10}$$

Once the image space was converted into the series of binary images, with p binary images and the scale was defined by an array of m values, the resulting normalized determinant was a function of both scale and binary image, or simply a function of two variables, $N(l, c)$. Creating N for all scales and binary images completed the analysis (processing) phase of facial recognition. The next section will detail the options tested for post-processing (or preparing the N for comparison to a different face).

5.4. Post-Processing

For a particular N, several options existed for post-processing to expedite comparison. The first was to fit an n^{th} order polynomial to the curve generated by varying scale for each binary image (essentially, treating N as only a function of scale). The second option was to calculate the slope in the scale and binary image directions for each point in N. The final option was to do nothing to N, and compare it directly to another.

The case of polynomial fitting was handled directly by MATLAB, which uses a least squares method (33). In least squares, a polynomial, P, of degree, b, is defined by Equation 5.11. The number of x-y paired data points (n) to which the curve will be fit must be at least one more than m. Using the fixed number of discrete points, a curve can be approximated by minimizing the squared difference (hence the name Least Squares) between the actual data points (y) and the approximated polynomial (P), (Equation 5.12).

$$P(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \ldots + c_b x^b \quad \text{Equation 5.11}$$

$$q = \sum_{i=1}^{n} \left(y_i - P(x_i)\right)^2 \quad \text{Equation 5.12}$$

To minimize the difference, the partial derivatives of q with respect to each of the coefficients of P are equated to 0 (Equation 5.13). Once the derivatives are rearranged, they form a system of equations (Equation 5.14) which must be simultaneously solved to determine the optimal coefficients which represent the least amount of residual error between the actual data points and the fit polynomial (34). The resulting coefficients for each contrast level were stored to data files and recalled during comparison.

$$\frac{\partial q}{\partial c_0} = 0, \quad \frac{\partial q}{\partial c_1} = 0, \ldots, \frac{\partial q}{\partial c_b} = 0 \quad \text{Equation 5.13}$$

$$\begin{aligned}
c_0 n + c_1 \sum_{i=1}^{n} x_i + c_2 \sum_{i=1}^{n} x_i^2 + \ldots + c_b \sum_{i=1}^{n} x_i^b &= \sum_{i=1}^{n} y_i \\
c_0 \sum_{i=1}^{n} x_i + c_1 \sum_{i=1}^{n} x_i^2 + c_2 \sum_{i=1}^{n} x_i^3 + \ldots + c_b \sum_{i=1}^{n} x_i^{b+1} &= \sum_{i=1}^{n} x_i y_i \\
c_0 \sum_{i=1}^{n} x_i^2 + c_1 \sum_{i=1}^{n} x_i^3 + c_2 \sum_{i=1}^{n} x_i^4 + \ldots + c_b \sum_{i=1}^{n} x_i^{b+2} &= \sum_{i=1}^{n} x_i^2 y_i \\
&\vdots \\
c_0 \sum_{i=1}^{n} x_i^b + c_1 \sum_{i=1}^{n} x_i^{b+1} + c_2 \sum_{i=1}^{n} x_i^{b+2} + \ldots + c_b \sum_{i=1}^{n} x_i^{b+b} &= \sum_{i=1}^{n} x_i^b y_i
\end{aligned} \quad \text{Equation 5.14}$$

The second option for post-processing was to compute areas of positive and negative slope. The slopes were then compared quickly to other curves through Boolean operations. However, faces have many close similarities, meaning the resulting normalized determinants were also similar. To remove the effects of commonality, the average normalized determinant for all 400 images in the database was found. Once the average had been found, it was

subtracted from all normalized determinant surfaces. The resulting surface was still a discrete function of two variables.

The slopes were found using a simple second order central difference using 2 points. For a square of points (*x* and *y*), two slopes were found. The equations of the slopes used are given in Equation 5.15. Shown pictorially in Figure 5-4 is the assumed grid of normalized determinant used in Equation 5.15. The two sets of slopes (*l* and *c*) were stored independently as either +1 or 0 by truncating the result of Equation 5.15 for slopes greater than 0 or less or equal to 0, respectively. The effect of determining the slopes in this manner was that no slope could be found on the perimeter of *N*. To calculate slopes on the border of *N*, the *i*-1 (or *i*+1 for the opposed border) terms in Equation 5.15 were replaced by *i* and likewise for *j*. The substitution of *i* for *i*±1 resulted in a first order forward (or rearward) difference (35), (36).

$$s^{l}_{i,j} = \frac{N(l_{i+1}, c_j) - N(l_{i-1}, c_j)}{l_{i+1} - l_{i-1}}$$

$$s^{c}_{i,j} = \frac{N(l_i, c_{j+1}) - N(l_i, c_{j-1})}{c_{i+1} - c_{i-1}} \qquad \text{Equation 5.15}$$

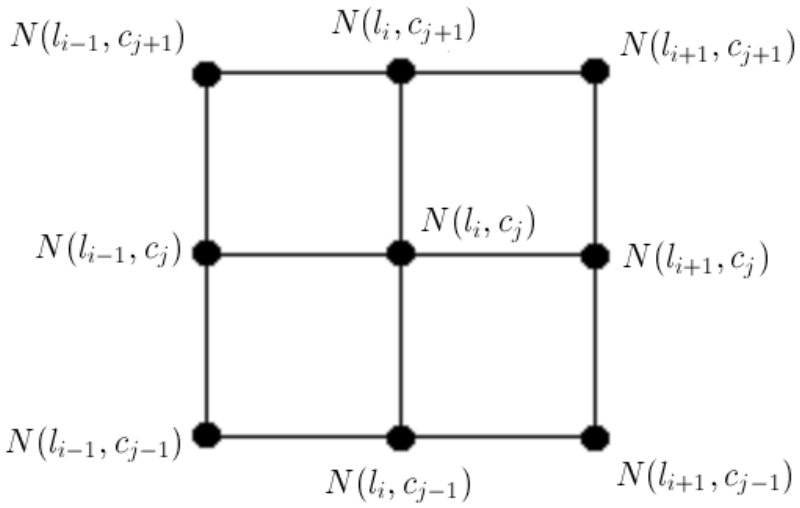

Figure 5-4: Normalized Determinant grid

The final case was simply doing nothing at all. The normalized determinant was stored directly to a data file which was called up during comparison. Once the pre-processing, processing, and post-processing were complete for each of the 400 electronic face images, the database of coefficients which described each face was complete. The next section will describe how faces are compared in the database.

5.5. Comparison of Faces

There were two cases of comparison which must be addressed. The first case results from the coefficient fitting and direct storing methods of post-processing. The second case is the slope calculation. The slope calculation is easier to address, and will be done second. For convention of this section, the test case (image which is being tested against the database) will be treated as an array of coefficients (or slopes), *a*. The database will be treated as a matrix whose columns represent the individual arrays of coefficients for each image in the database, *D*. In

reality, the arrays of image data in the database were loaded and checked individually rather than in one large matrix to save on computational resources.

For the first case (not slopes), the comparison made use of two "tolerance" values which were experimentally optimized. The first tolerance value was the maximum allowable difference between individual coefficients, μ. The second tolerance value was the minimum number of coefficient matches, v, required to match one image (face) to another image in the database. For instance, consider a and one column of D, D^i. The first problem was to determine all of the percent differences (Equation 5.16) between the coefficients of a and D^i. The PD_j was considered successful if it wa less than the pre-established μ.

$$PD_j = \frac{a(j) - D^i(j)}{a(j)} \quad j = 1, 2, 3, \ldots, M \qquad \text{Equation 5.16}$$

Once all of the percent differences between the coefficients of the test image and any image in the database were found, the ratio of successful (less than μ) differences to the total number (M) was calculated. If this ratio was greater than the pre-determined value v the test image and the image from the database were predicted to match (i.e. have the same subject). The pseudo-code is presented in Figure 5-5 to clarify what is being done during this type of comparison. The only remaining step was to check the test image against the remainder of the database and record all predicted matches.

```
success count = 0
for j = 1,2,3,…,M
  diff = (a(j) – D^i(j)) / a(j)
  if diff < μ
    success count = success count + 1
  endif
endfor

if (success count / M) > v
  Match predicted between test image and image i from database
else
  match is not predicted
endif
```

Figure 5-5: Pseudo-code for the comparison algorithm

For the case of the slopes, which only had two possible values (+1, -1), the only match criterion was similar to the second (the ratio) from the previous case. Here, *v* became the minimum count of direct matches (+1 to +1 and similar). If the number of matches was greater than *v*, the test image and image from the database were predicted to match.

5.6. Experimental Design

There are quite a few parameters which were investigated experimentally. However, there are few which have previously been held fixed in the present analysis. The main fixed quantity was the location of the subject's pupils. If this work were to be extended to another "subject" (i.e. something other than faces), the inter-pupil distance could no longer be used to non-dimensionalize the scale values. However, it is expected that in almost any specific application, some representative macro scale should be easily definable.

The other subject to consider (during pre-processing) was the shape of the region around the eye to be considered during analysis. The ellipse was chosen by the researcher simply through observation. Various shapes were tried, and ellipse provided the best qualitative look.

However, a rectangle mask may also work (though it will include more of the background). To prove the effectiveness of the ellipse assumption, 10 images were selected randomly from the dataset and masked both ways. The resulting normalized determinates (once the optimal number of scales and binary images was found using the methodology described later in this section) for each masking were found and plotted.

The number of iterations per scale (and binary image) were fixed at 10^4. As shown previously, the error associated with the random walk (Monte Carlo) for a large sample size such as 10^4 is 0.32%. An error rate of less than 1% seems acceptably low. All of the numerical experiments were carried out on a computer with the following specifications:

Table 5-1: Computer Configuration

CPU	AMD Phenom X4 920
Memory	4.0 Gb. DDR2 1066
Operating System	Fedora Core 14 (64 bit)
Compiler / Language	MATLAB R2009b

5.6.1. Optimal Number of Scales and Contrast Values

There were still several parameters which could be manipulated. First, was the number of scale values m (in the scale array). The second parameter which needed optimized during pre-processing was the number of binary images p to create. The first experiment attempted to determine the optimal number of each by choosing to perform no post-processing (i.e. writing the normalized determinant curves out directly) and fixing the values of $\mu = 0.25$ and $\nu = 0.75$. That is, the maximum allowable percent difference between coefficients was 25% and the minimum coefficient match ratio was 75% for predicting a match.

The entire random walk analysis was performed twice for each image (i.e. two separate databases were created). The first database created was considered the actual database. The

second database was used to test each image against the database. In other words, each of the 400 images selected from the FERET database were completely processed into a database of 400 normalized determinant surfaces. Then the 400 images were ran through the analysis again, and tested against the database. The number of instances of a particular subject in the database was predetermined. Therefore, the number of correct matches (and incorrect matches) for a particular image could be found; and this process was repeated for each image.

For instance, consider that a database has already been created and now image *g* is being tested against the database. Image *g* contains subject 5. When image *g* is compared to each image (normalized determinant) in the database, the predicted matches should only be the images which contain subject 5. However, the predicted matches may include other subjects and may not even include subject 5. In order for the comparison of image *g* to the database to be considered successful (for a particular set of parameters), the predicted matches may only contain subject 5. In addition, all of the instances of subject 5 must also be returned for the comparison to be considered successful.

Because the number of scales and contrast values both directly influence the accuracy of the normalized determinant (by providing more data points for a comparison), they cannot be varied independently. Indeed it is likely that a smaller number of scales may be compensated by a larger number of contrast values. As such, a total of 16 configurations were selected and are presented in Table 5-2. From this list, each configuration was used as described above and the percentage of images which were successfully compared to the database were recorded. In order to determine the best possible configuration, the test values were honed in sections of interest. The configuration which provided the highest success rate was used going forward for subsequent experiments.

Table 5-2: Variation of number of scales and number of contrast values

Case	Number of Scales	Number of Binary Images
1	5	5
2	10	5
3	20	5
4	40	5
5	50	5
6	10	3
7	10	5
8	10	10
9	10	15
10	10	20
11	10	30
12	20	20
13	20	30
14	40	20
15	40	30
16	50	30
17	40	10
18	50	10
19	40	15
20	50	15
21	60	5
22	60	10
23	60	15

In test case 1, only 25 points made up the normalized determinant. It seemed unwise to use fewer than 25 points to delineate among a dataset of 400 individuals. Conversely, case 16 created a normalized determinant of 1,500 floating point numbers. That made an array containing roughly 4 times the number of elements to images in the database in floating point numbers. Cases 17 through 23 were included for completeness to deremine whether the added number of scales affected the accuracy.

It should be noted that increasing the number of scales only affects the run-time of the random walk. That is for every new scale value, another 10^4 iterations of the random walk are performed. However, increasing the number of contrast values increases the number of binary

images. Increasing the number of binary images increases both the processing (more random walks) and the pre-processing time. Now, additional loops with costly conditional statements must be evaluated to create the additional binary image(s). That is why the maximum number of contrast values selected was generally lower than the maximum number of scales.

5.6.2. Comparison of Ellipse and Rectangle Masking

Until now, this document has assumed that the ellipse masking as documented in the previous chapter was equivalent to fitting a rectangle with the same width and height as the ellipse's semi-major and minor axes. Using the optimal configuration of scales and contrast values from the previous experiment, ten images (selected randomly from the database of 400) were masked using the rectangle and difference between the normalized determinant derived from the rectangle masking and ellipse masking was noted.

5.6.3. Optimal Value of v for Slope-based Comparison

Once the optimal number of scales and binary images was determined, the optimal configuration of two different comparison methods was determined for each of the post-processing routines. Concentrating first on the slope post-processing routine, there was only one parameter to consider, the required percentage, v, of slope (coefficient) matches to predict an image to image match. Recall that the normalized determinant was converted to a series of zeros and ones relating to the slope at a given point.

The logical experiment was to vary v between 0 and 1 according to Table 5-3. Obviously, when v is 0, the algorithm will fail completely as every image would be predicted to match every image. However, Case 1 was included for completeness. As before, the cases were honed to determine the best possible match rate. To improve the accuracy of this method, the

average normalized determinant was computed for the entire database of 400 images. This average was then subtracted from each normalized determinant (test and database alike). The experiment was re-run by varying *v* between 70 and 100 to determine the effects of this modification.

Table 5-3: Variation of *v* for slope matching

Case	*v* (%)
1	0
2	10
3	20
4	30
5	40
6	50
7	60
8	70
9	80
10	90
11	91
12	92
13	93
14	94
15	95
16	96
17	97
18	98
19	99
20	100

5.6.4. Number of Coefficients in Curve Fit.

For the curve fitting post-processing method, the optimal order of the polynomial was established before comparisons. As previously mentioned, MATLAB was used to fit the polynomials by invoking built-in functions which have a maximum order of 9. Using lower orders than 5 would likely not accurately describe the shape of the curves. However, it was possible that 9 would be too high an order. The third experiment used the cases presented in

Table 5-4 to determine the best polynomial order. Again, the case which provided the highest percentage of successful matches to the database was noted to deremine which post-processing method produced the highest match rate.

Table 5-4: Possible polynomial orders

Case	Order
1	5
2	6
3	7
4	8
5	9

The experiment was performed using the number of scales and binary images already established. In a similar manner to the first experiment, the values of μ and v were fixed to 0.25 and 0.75, respectively. Once the optimal number of coefficients in the curve fit had been found, the final experiment determined the optimal configuration of μ and v for the post-processing schemes.

5.6.5. Optimal Values of μ and v

Using the number of scales and binary images found previously, the optimal values of the coefficient percent difference tolerance μ and required match rate of coefficients v between two normalized determinant (or polynomial) coefficients was experimentally determined. Here, the values of μ and v were varied independently until the highest percentage of successful comparisons was found. During the testing of v, μ was fixed to 0.25 and during the testing of μ, v was fixed to the optimal value (v will be tested first). This experiment was performed twice. Once for the curve-based post-processing, and once for using the raw normalized determinant data as both post-processing schemes required two variables to be fixed for comparison.

For the direct utilization of the normalized determinant, ν was varied between 0.75 and 1; 0.65 and 1 for the polynomial fit. For both types of comparison, μ was varied between 0.1 and 0.35.

5.6.6. Final Algorithm Selection

Once all of the parameters associated with each post-processing and comparison method were established, the method which produced the highest percentage of successful matches to the database was selected for GUI development. The GUI allows a user then to select the locations of the pupils by hand. The automation takes over and the remainder of the algorithm steps are performed automatically. The user then has the option to add the results to a pre-existing database (or create a new one). The user is able to compare the selected image to an existing database to determine whether the selected image contains a subject already stored in the database.

5.7. Summary

This section has shown the process by which the algorithm was parameterized. The next section will document the results of the various experiments documented in this section.

6. Results and Discussion

All source codes developed for this section are available in Appendix B and Appendix C.

6.1. Determining the Optional Number of Scales and Binary Images

After all 400 images had been run against the database, the percentage which were successful was recorded (recall that an image to database match is only successful when then correct image subject is returned with no extra subjects listed). The results are presented in Table 6-1, which includes the updated case configuration and the success rate results. Case 18

presented the optimal configuration of 50 scales and 10 binary images, which had the greatest success rate of 13.25% (or 53 or 400 images where successfully matched).

Table 6-1: Results of Experiment 1

Case	Number of Scales	Number of Binary Images	Percent Successful
1	5	5	6.250%
2	10	5	8.651%
3	20	5	11.500%
4	40	5	12.500%
5	50	5	12.500%
6	10	3	5.000%
7	10	5	9.250%
8	10	10	8.750%
9	10	15	9.000%
10	10	20	9.250%
11	10	30	8.750%
12	20	20	10.500%
13	20	30	11.250%
14	40	20	11.500%
15	40	30	12.000%
16	50	30	12.000%
17	40	10	12.750%
18	50	10	13.250%
19	40	15	12.000%
20	50	15	12.250%
21	60	5	11.750%
22	60	10	12.750%
23	60	15	12.250%

One might think that the best success rate of 13.25% might be unacceptably low. However, several critical components of the analysis algorithm have yet to be "calibrated." For instance, the values of the matching criteria were chosen artificially and no attempt has yet been made to distinguish or quantify the effectiveness of any of the comparison techniques.

It is also interesting to note that increasing neither the quantity of scales nor binary images (beyond the optimal number) increased the performance of the analysis (i.e. the accuracy of the algorithm cannot be increased indefinitely by adding more scales or binary images). The

cause of this phenomenon is readily explainable. The scales and contrast values were selected (uniformly) between a minimum and maximum (say, *a* and *b*, respectively). On the real number line between *a* and *b*, infinitely many fractional numbers exist. However, the number of integers is finite. Since image space is discrete, a limit to the number of unique scales and binary images which can be created exists.

Indeed, it should be noted that the images selected for the dataset of 400 had an average inter-pupil distance of 127.71 pixels. The scales were varied to a maximum of 25% inter-pupil distance, or an average of 31.92 pixels. This means that the 60 unique scales were uniformly spaced at approximately 0.5 pixel intervals. One-half pixel intervals make little sense because a pixel is, by definition, the smallest addressable element. There was a fundamental limit to the number of unique scales which could be created for this dataset and this limit was reached a 60 scales. This explains why the percentage successful became lower as the number of scales was increased.

Case 18 represented 500 discrete points in the Normalized Determinant curve which are then use to compare images. The images themselves had approximately 10,000 to 15,000 pixels which have not been masked. The Normalized Determinant then represented a compression of approximately 95% from the source image—a substantial decrease in required memory (both long and short term) and computational requirement for comparisons among datasets. Also, the average time to create the normalized determinant data for one image was 62 seconds on the experimental computer. This time does not include the user-selection of the two pupil locations as the locations of the pupils were pre-selected and stored in an accessible data file for simplicity.

6.2. Comparison of Ellipse and Rectangle Masking

10 images were selected randomly from the database of 400 and masked with both ellipses and rectangles and have been rotated to match any tilt in the subject's head. Table 6-2 gives the list of images which were selected.

Table 6-2: Images Selected

Case	Image Name
1	00244_940128_fa.tif
2	00200_940128_fa.tif
3	00324_940422_fb.tif
4	00561_940519_fa.tif
5	00595_940928_fb.tif
6	00529_940519_fb.tif
7	00472_960627_fb.tif
8	00519_940519_fa.tif
9	00020_930831_fb.tif
10	00579_941031_fa.tif

Figure 6-1 through Figure 6-10 are surface plots of the normalized determinants for each of the 10 images. On the left in each image is the determinant for the ellipse mask, and on the right is the rectangle mask. In most of the images, by qualitative inspection, it does appear that each of the determinants is at least similar while the plots for different images differ mch more dramatically. This confirms the assumption that rectangle and ellipse cutouts were comparable. Because the ellipse region requires less memory to store (and use), the algorithm continued with ellipse masking.

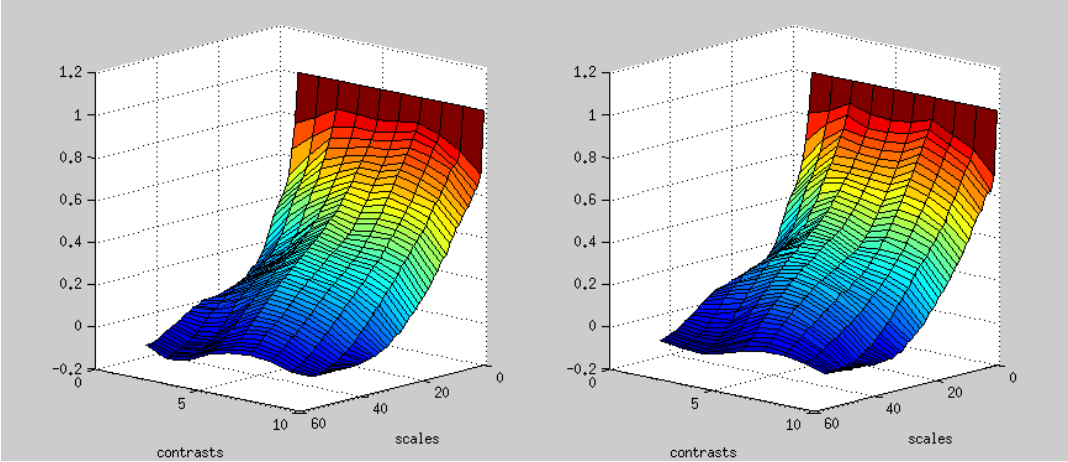

Figure 6-1: Normalized Determinants for Image 1

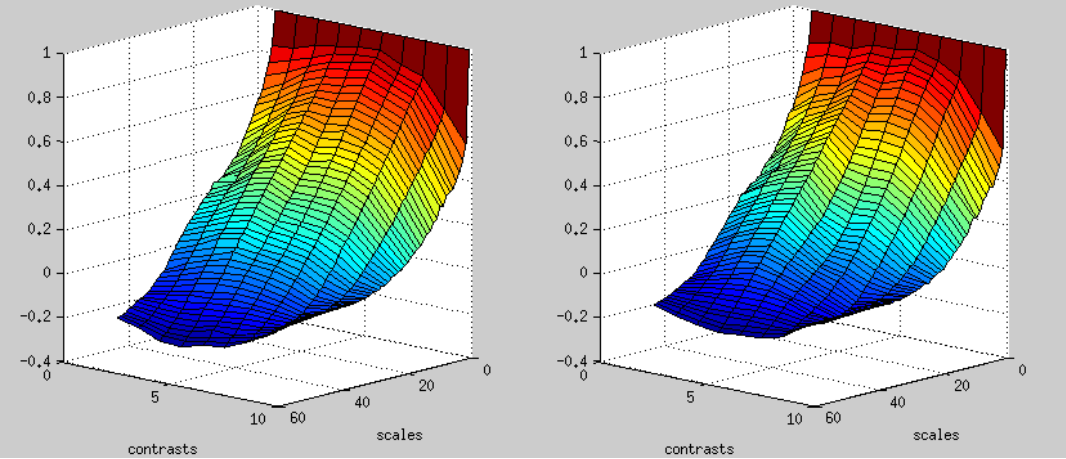

Figure 6-2: Normalized Determinants for Image 2

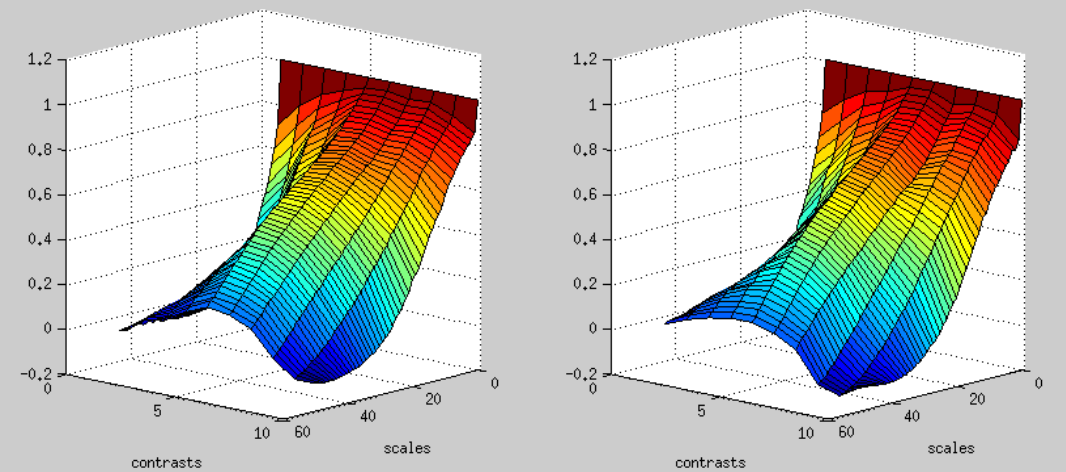

Figure 6-3: Normalized Determinants for Image 3

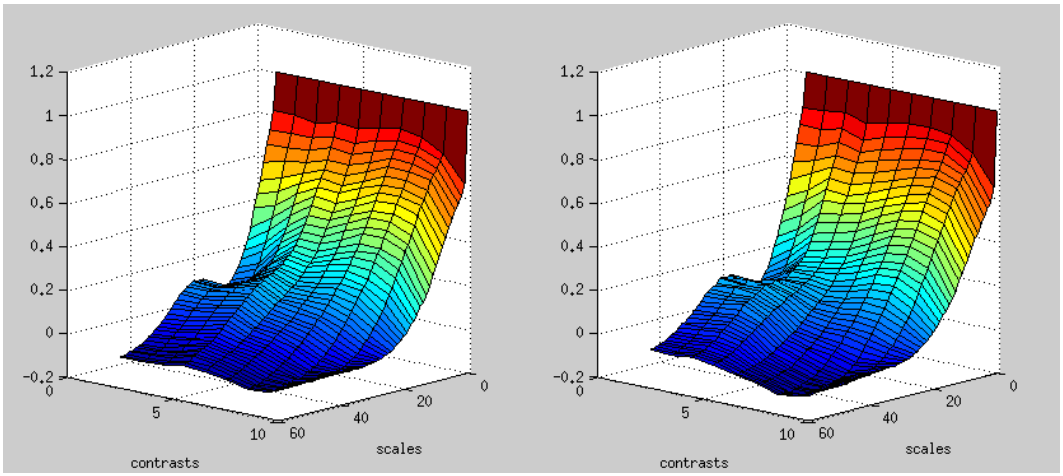

Figure 6-4: Normalized Determinants for Image 4

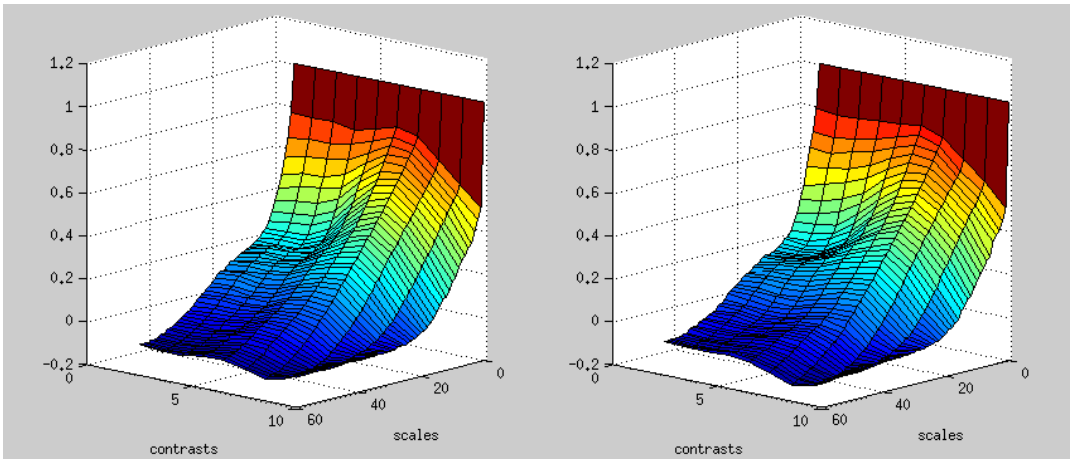

Figure 6-5: Normalized Determinants for Image 5

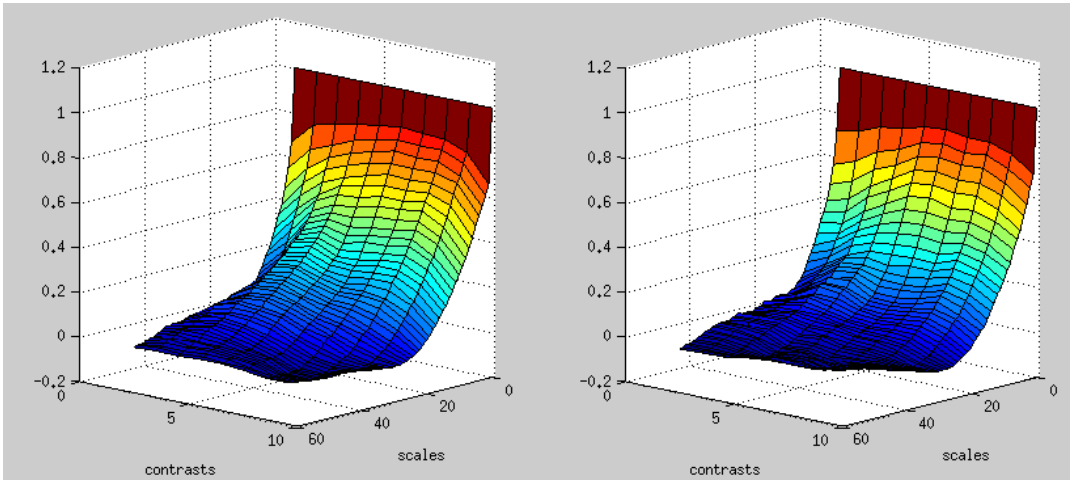

Figure 6-6: Normalized Determinants for Image 6

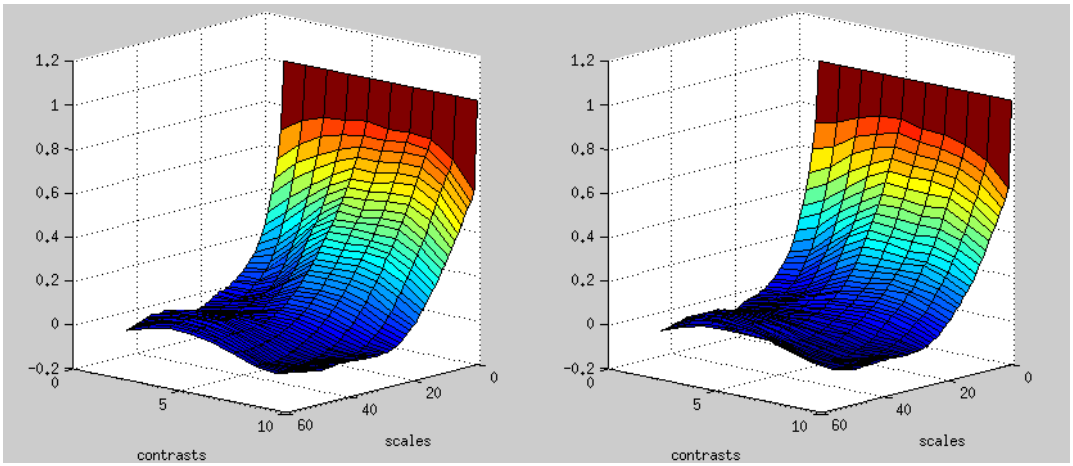

Figure 6-7: Normalized Determinants for Image 7

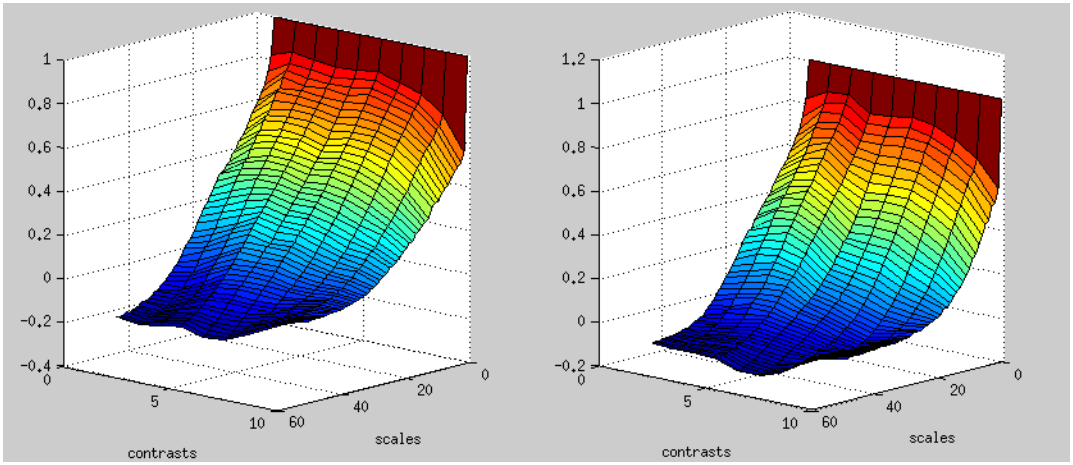

Figure 6-8: Normalized Determinants for Image 8

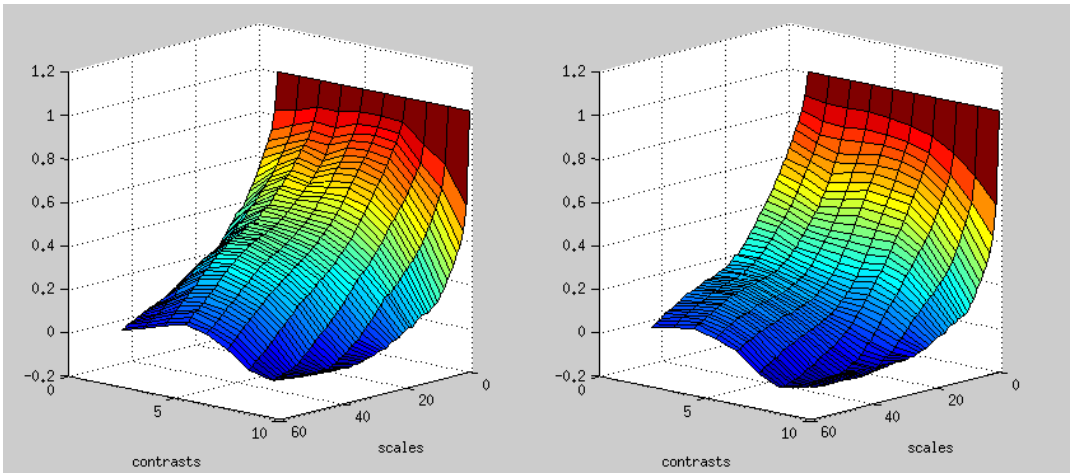

Figure 6-9: Normalized Determinants for Image 9

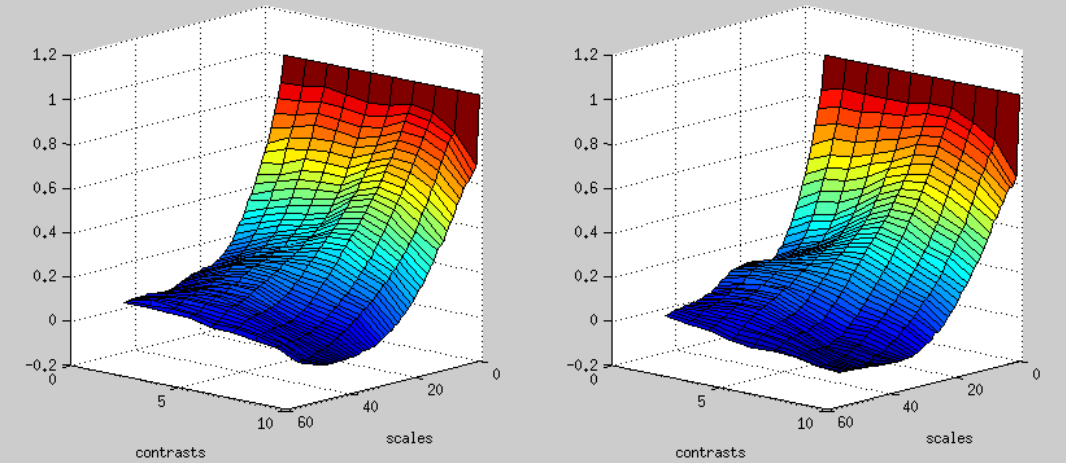

Figure 6-10: Normalized Determinants for Image 10

6.3. Slope Based Comparisons

As described in section 5.6.3 the slopes of each of the 400 normalized determinants were calculated for both the "scale" and "binary image" direction. This resulted in two sets of 0s and 1s for each of the 500 discrete points in the normalized determinant, or 1,000 binary numbers for each face image. To determine if a particular direction was detrimental to the comparison, each was run independently first. The results were quite poor, but are listed in Table 6-3.

Table 6-3: Results of *v* for each set of slopes

Case	*v* (%)	Contrast Percent Successful	Scale Percent Successful
1	0	0.000%	0.000%
2	10	0.000%	0.000%
3	20	0.000%	0.000%
4	30	0.000%	0.000%
5	40	0.000%	0.000%
6	50	0.000%	0.000%
7	60	0.000%	0.000%
8	70	0.250%	0.000%
9	80	1.750%	0.000%
10	90	17.750%	0.500%
11	91	22.250%	0.250%
12	92	26.750%	0.250%
13	93	28.750%	0.750%
14	94	27.500%	2.500%
15	95	22.750%	2.250%
16	96	12.000%	2.750%
17	97	7.000%	3.000%
18	98	2.000%	0.750%
19	99	0.500%	0.250%
20	100	0.000%	0.000%

As can be seen in Table 6-3, case 14 held the most promise for a high success rate when the slopes from contrast (binary image) and scale were combined. Table 6-4 gives the results of combing both sets of slope data, including the average time per image to run that image against the entire database. This time does not include the enrollment (pre-processing and processing); it was only the time required to check a particular image against the 400 sets of slopes for a particular value of *v*. It is interesting to note that simply adding the values of case 14 (27.5% and 2.5%) linearly results in 30%; however, when running with both sets of slopes combined, case 14 yielded 35.5%. There may have been some higher order coupling.

Table 6-4: Combined Slopes Results

Case	v (%)	Combination Percent Successful	Average Run Time (s)
1	0	0.000%	0.019
2	10	0.000%	0.019
3	20	0.000%	0.019
4	30	0.000%	0.019
5	40	0.000%	0.019
6	50	0.000%	0.019
7	60	0.000%	0.019
8	70	0.000%	0.019
9	80	0.500%	0.019
10	90	9.250%	0.018
11	91	14.500%	0.018
12	92	21.000%	0.018
13	93	27.750%	0.018
14	94	35.500%	0.018
15	95	26.000%	0.018
16	96	19.000%	0.018
17	97	5.500%	0.018
18	98	1.500%	0.018
19	99	0.000%	0.018
20	100	0.000%	0.018

During the creation of the average normalized determinant, the average determinant was plotted after each time 50 more face images had been added and averaged. Figure 6-11 through Figure 6-15 show the progression (with repeated images left out). After the first 200 images have been averaged together (Figure 6-14), little appreciable change occurs while averaging in the remaining 200. Images were used in the order presented in Appendix A.

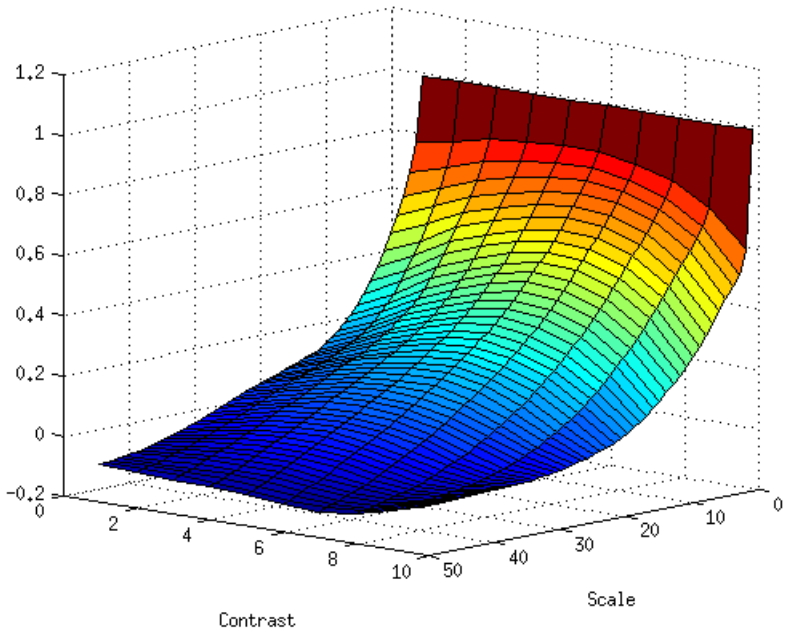

Figure 6-11: Average of first 50 images

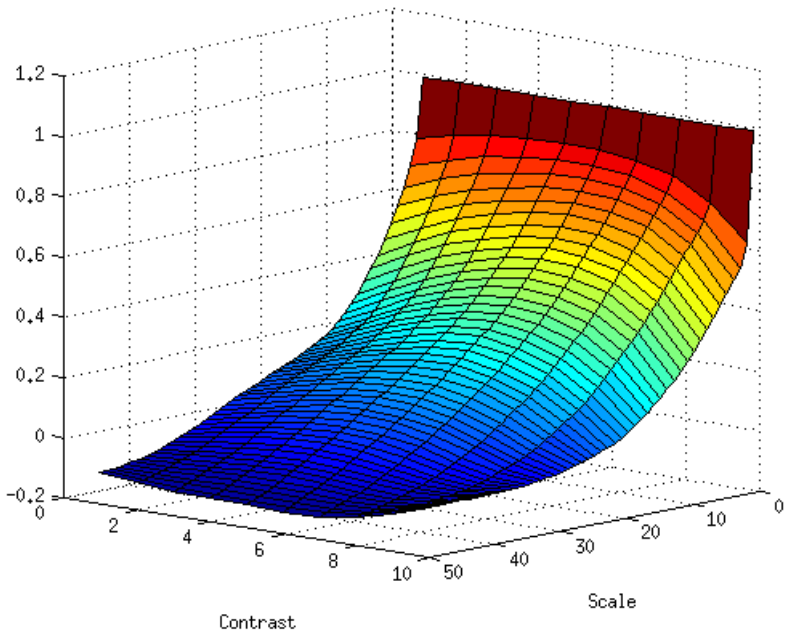

Figure 6-12: Average of first 100 images

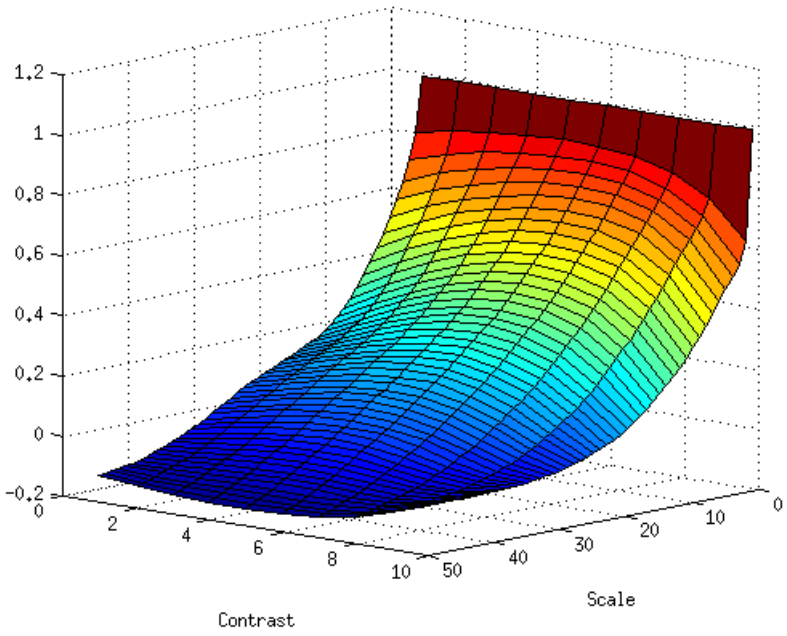

Figure 6-13: Average of first 150 images

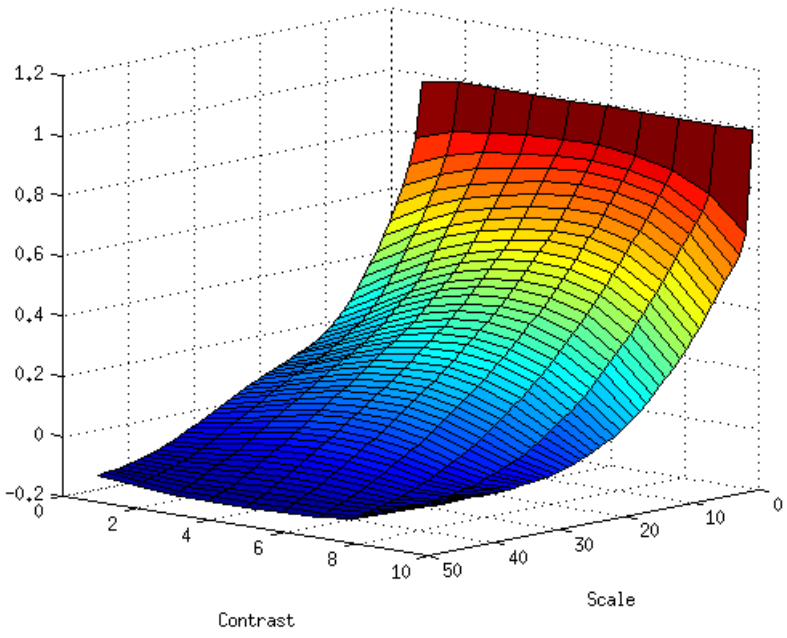

Figure 6-14: Average of first 200 images

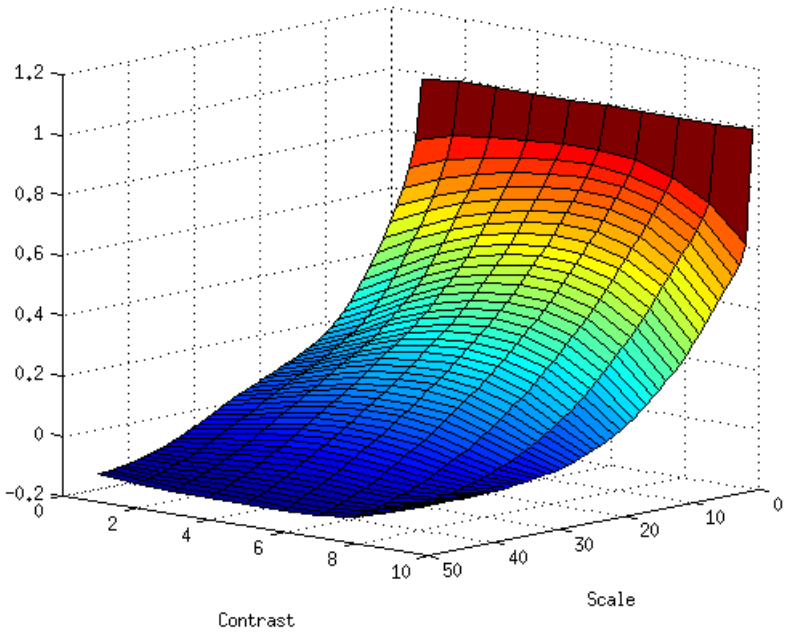

Figure 6-15: Average after all 400 images

After the average normalized determinant was subtracted for each of the 400 in both datasets, the experiment was re-run to determine a new optimal *v*. Again, each slope direction was tested independently, before using both simultaneously.

The results have been tabulated in Table 6-5. Here, case 9 yielded the highest success rate of 54.75%. However, case 15 predicted a combination success rate of 49% + 18.75%, or 67.75%. Such a high match rate did not occur though. While 54.75% seems low, the best result of experiment 1 was only 13.25%. This represents a substantial improvement. Now, 219 images out of 400 have been successfully matched (returned image subject only from database).

Table 6-5: Results of slope-based matching for modified determinants

Case	v (%)	Contrast Percent Successful	Scale Percent Successful	Combination Percent Successful
1	70	0.250%	0.250%	9.000%
2	71	0.750%	0.750%	11.750%
3	72	1.500%	1.500%	18.000%
4	73	2.250%	2.250%	26.000%
5	74	4.250%	4.250%	31.750%
6	75	7.000%	7.000%	39.500%
7	76	10.750%	10.750%	43.500%
8	77	13.000%	13.000%	49.500%
9	78	17.500%	17.500%	54.750%
10	79	22.250%	22.250%	53.500%
11	80	28.000%	28.000%	54.000%
12	81	34.500%	21.000%	51.250%
13	82	39.250%	16.000%	46.750%
14	83	43.750%	20.500%	44.000%
15	84	49.000%	18.750%	38.750%
16	85	47.000%	15.250%	33.250%
17	86	48.000%	12.750%	27.750%
18	87	46.000%	10.000%	21.500%
19	88	39.500%	7.000%	16.000%
20	89	35.250%	4.000%	9.500%
21	90	29.000%	1.500%	5.000%
22	91	24.750%	0.000%	3.500%
23	92	17.250%	0.000%	2.500%
24	93	11.750%	0.000%	1.000%
25	94	9.500%	0.000%	0.000%
26	95	5.250%	0.000%	0.000%
27	96	2.500%	0.000%	0.000%
28	97	1.000%	0.000%	0.000%
29	98	0.500%	0.000%	0.000%
30	99	0.000%	0.000%	0.000%
31	100	0.000%	0.000%	0.000%

While the results of the accuracy of this experiment have been less than stellar, the computational requirements do provide some opportunity in computational time reduction. The time required on the experimental machine to create two sets of slope data for one image was 0.0954 seconds, on average. As this section has already shown, the time required to run one

image against a database of 400 images was less than 0.02 seconds per image. The average time to create the normalized determinant was 62 seconds per image, which remains the majority of computation time for start to finish analysis.

6.4. Optimal Order of Polynomial Fit

The order of the polynomial fit which produced the highest accuracy in the comparisons was 7, as is shown in Table 6-6. The polynomials were fit by removing a "slice" of the normalized determinant along the scale axis. So, each polynomial was a function of scale and 10 polynomials (because there were 10 binary images) described the face. To allow for comparison from one image to the next (to "non-dimensionalize?"), scale was set to an integer array from 1 to 50 (there were 50 scales) instead of the pixel based values were which found during the random walk analysis. This means that the x spacing used in Equation 5.11 was uniform from one normalized determinant to the next.

Also note that Table 6-6 gives the average time (in seconds) to fit a curve of a particular order to a given normalized determinant. For order 7, the time required, per image, was approximately 0.167 seconds. Compared to the 0.1 seconds for generating the slopes, the curve fitting method presented similar computational time requirements. Remember that the time required to generate a normalized determine was over 60 seconds per image. So, generating the polynomial fit data also presents no significant increase to run-time.

Table 6-6: Results of differing orders of polynomial curve fit

Case	Poly Order	Curve Fit Time per Image (s)	Percent Successful
1	5	0.053	22.25%
2	6	0.160	18.00%
3	7	0.167	24.75%
4	8	0.174	14.50%
5	9	0.181	12.00%

6.5. Optimal Values of μ and v

Finally, the optimal values of v and μ were determined for both the curve-fitting method and for working with the normalized determinant directly.

Table 6-7 and Table 6-8 give the results for the direct utilization of the normalized determinant. Recall that once an optimal value of v was found, it was used to determine μ. Using the normalized determinant directly resulted in a best case of 42.25% of the 400 images being matched correctly (only proper subjects returned from database).

Table 6-7: Determining v for directly using the normalized determinant

Case	v	Percent Successful
1	0.75	37.75%
2	0.76	38.25%
3	0.77	39.25%
4	0.78	39.50%
5	0.79	41.50%
6	0.8	42.25%
7	0.81	39.75%
8	0.82	38.00%
9	0.83	36.00%
10	0.84	33.00%
11	0.85	28.75%
12	0.9	11.75%
13	0.95	1.00%
14	1	0.00%

Table 6-8: Using $\nu = 0.80$, and determining μ

Case	μ	Percent Successful
1	0.1	4.00%
2	0.15	15.00%
3	0.2	33.50%
4	0.21	37.25%
5	0.22	39.25%
6	0.23	39.25%
7	0.24	39.25%
8	0.25	42.25%
9	0.26	41.75%
10	0.27	40.25%
11	0.28	39.50%
12	0.29	40.25%
13	0.3	39.00%
14	0.35	36.25%

Table 6-9 and Table 6-10 give the results for the fitted curve coefficient matching. As can be seen, using the normalized determinant directly was more accurate than the curve fitting. With all of the curve fitting parameters honed, the best success rate achieved was only 25.75% (or 103 of the 400 images).

Table 6-9: Determining v for the curve coefficients

Case	v	Percent Successful
1	0.65	17.00%
2	0.7	18.25%
3	0.71	19.00%
4	0.72	21.00%
5	0.73	22.00%
6	0.74	22.00%
7	0.75	24.75%
8	0.76	23.25%
9	0.77	23.25%
10	0.78	23.25%
11	0.79	22.50%
12	0.8	23.75%
13	0.85	19.50%
14	0.9	19.00%
15	0.95	11.00%
16	1	6.00%

Table 6-10: Using $v = 0.75$, and determining μ

Case	μ	Percent Successful
1	0.1	2.25%
2	0.15	12.50%
3	0.2	24.25%
4	0.21	24.50%
5	0.22	24.50%
6	0.23	24.75%
7	0.24	25.75%
8	0.25	24.75%
9	0.26	21.75%
10	0.27	19.50%
11	0.28	19.50%
12	0.29	18.00%
13	0.3	17.75%
14	0.35	12.25%

Because the curve fitting reduced the accuracy substantially (25.75% vs. 42.25%), the final GUI will not use curve fitting for post-processing. Directly using the normalized determinant and creating the slopes were not comparable in accuracy, so the GUI uses the slope

method after subtracting the average normalized determinant. The time required to compare one test image to the database of 400 for the direct utilization was approximately 0.014 seconds. Again, the comparison time was much smaller than the time to create the normalized determinant. The computational time required for a start to finish (pre-processing/enrollment to test against an established database) was almost entirely controlled by the time required to create a normalized determinant.

7. Conclusions

- The fingerprint algorithm has been adapted to face recognition. The adaptation did employ a series of binary images to compensate for the third dimension.
- The resulting scale spectra were normalized to remove the effects of differing numbers of black and white pixels.
- The Arc-Fraction Circle method explained several of the assumptions regarding the use of the random walk method. The features of the scale spectra were understood (and quantitatively predicted) by the AFC method.
- The resulting normalized determinants where then further studied to determine the best way to compare a new "image" to a database of established images. These parameters were studied in detail and refined through a parameterization study. An optimal configuration was found for the resulting GUI.
- As stated above, the run-time was greatly dominated by the computational time required to generate the normalized determinant from the random walk process. The enrollment times used in recent commercial studies were suggested to be less than 5 minutes (5). The algorithm developed here had a demonstrated run-time of approximately 62 seconds.

This means, while the algorithm needs strong accuracy improvements, it was very quick. However, it should be noted that the computer used for development and testing surpassed the equipment which existed in 2006 when the 5 minute benchmark was set.

- Ultimately, the GUI uses the slope matching for two reasons.
    - First, the slope matching was about 10 percentage points more accurate on the database than direct utilization (and has obvious methods of improvements which will not dramatically increase computational time or file size).
    - Second, the file size of the stored data for the slope matching scheme is much smaller because the slopes are stored as integer instead of the floating point numbers of the normalized determinants.
    - For example a typical slope data set is 2,020 bytes; whereas a typical normalized determinant data set is 4,560 bytes. The slope data represents a large reduction in disc and memory requirements for similarly sized (quantity of images) databases.

7.1. Contributions to the Field

- A novel, quick algorithm for analysis of non-binary electronic images to create a fractal image (specifically for face images) was developed.
- Scale-based fractal from source image was converted to a series of probilities which describe the image (instead of directly using the fractal) for faces.
- A normalization technique which accounts for differeing quantities of black/white pixels in binary images was implanted and tested.
- A fully analytic solution was demonstrated for a control source image (vertical bars), and a rigorous solution for all source images was found.

In summary, a new method for facial recognition has been developed. It was much quicker than currently available systems. While the accuracy demonstrated to date was low, it is expected that the accuracy can be improved with additional work. The algorithm was simple enough to be reduced to a GUI which any lay-person should be able to use with little training. The methods developed for fingerprint recognition have been modified to account for the unique complexities of the human face. The methodology used to drive the algorithm has been explained by a rigorous example. And, the 2-D scale spectrum curve features have been explained through the rigorous solution.

8. GUI Implementation

A GUI has been compiled based on the source codes developed for the first and third experiments. The GUI allows for the user to select a digital photograph (presumably of a face) and create a set of slope data. Then the user can compare the newly created slope data to a directory which contains a dataset of pre-created slope data. Any predicted matches are displayed to the user for visual confirmation.

The user is required to select the locations of the pupils of both eyes, which have been labeled in reverse in the GUI intentionally to create a logical flow and prevent the user from improperly selecting the pupils. The values for the number of binary images and slopes are pre-programmed directly into the source codes. However, the required percentage of slope point matches is not hard coded. It was left variable (although the default has been set to 80% as required) to the user in case the user would like the program to generate false positive matches or confirm whether it is matching correctly.

Figure 8-1 gives the Main Menu screen, which greets the user when the program is executed. Here, the user can select a digital photograph, save an existing pre-processed image, and directly load a set of slope data for comparison. Most of the control buttons are unavailable until the user has either loaded an image or dataset to prevent improper execution of sub-steps. In Figure 8-2, an image has been loaded into the main menu.

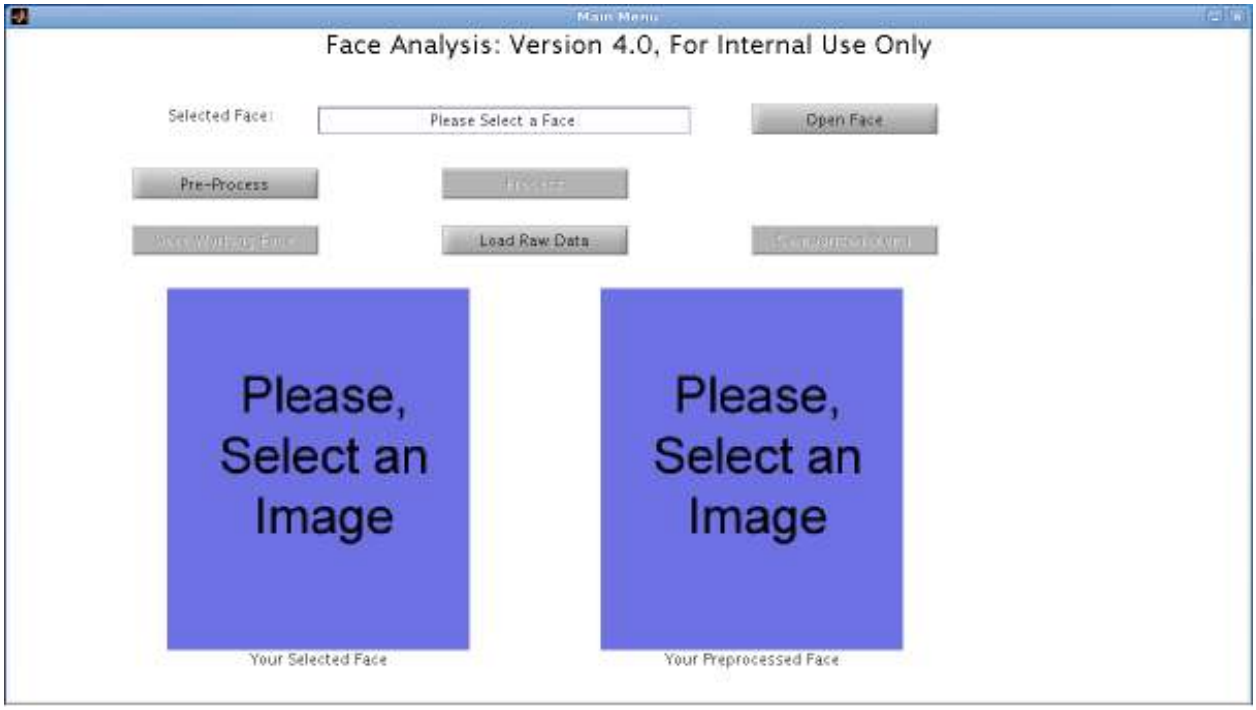

Figure 8-1: Main Menu after the program has been executed

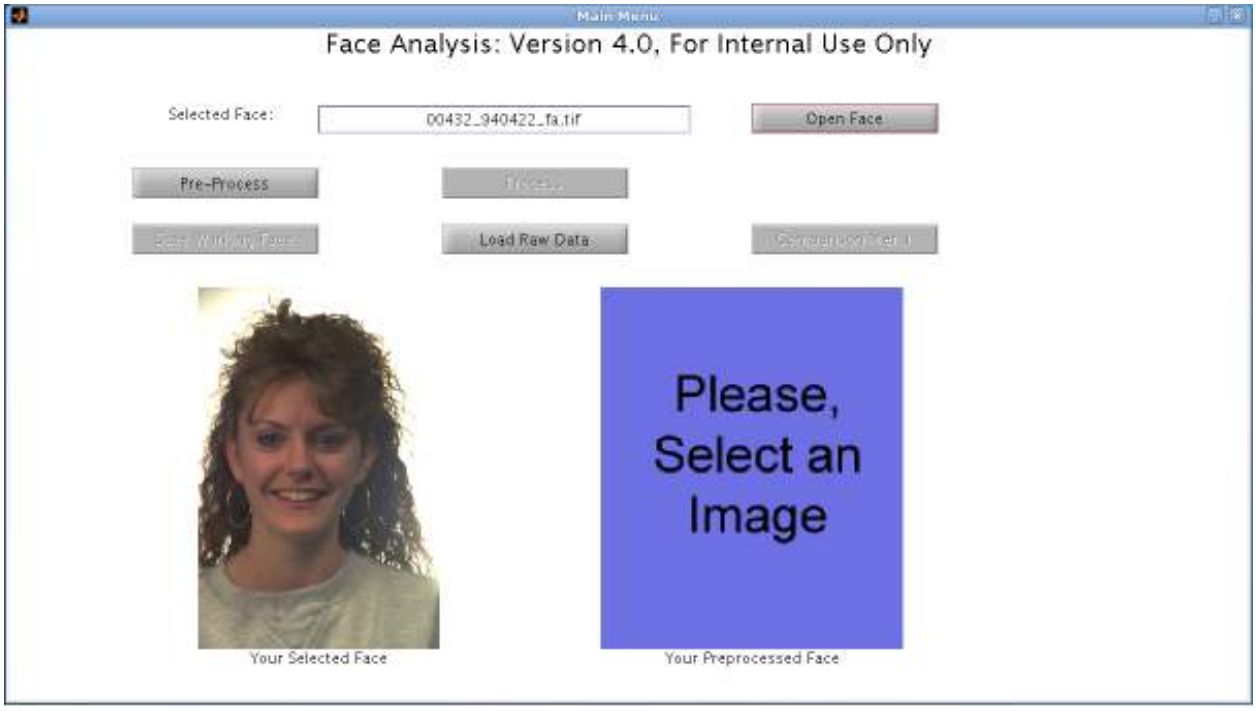

Figure 8-2: Main Menu after a digital photograph has been loaded

Once the digital photograph has successfully been loaded, the user must pre-process the image by selected the locations of the centers of the eyes, which is done in the pre-processing window. The pre-processing window is shown in Figure 8-3. It includes buttons for zooming and panning the image. The user must first select the "left eye," which is the eye which appears on the left-side of the image. Then, the user must select the "right eye." Finally, the user may (though it is not required) preview the masking that will occur. In Figure 8-4, the locations of the pupils have been selected and the image is ready for masking. The right eye is still marked from selection and the image remains zoomed in. Figure 8-5 shows the Main Menu screen again after the Pre-Processing window has handed back the masked image.

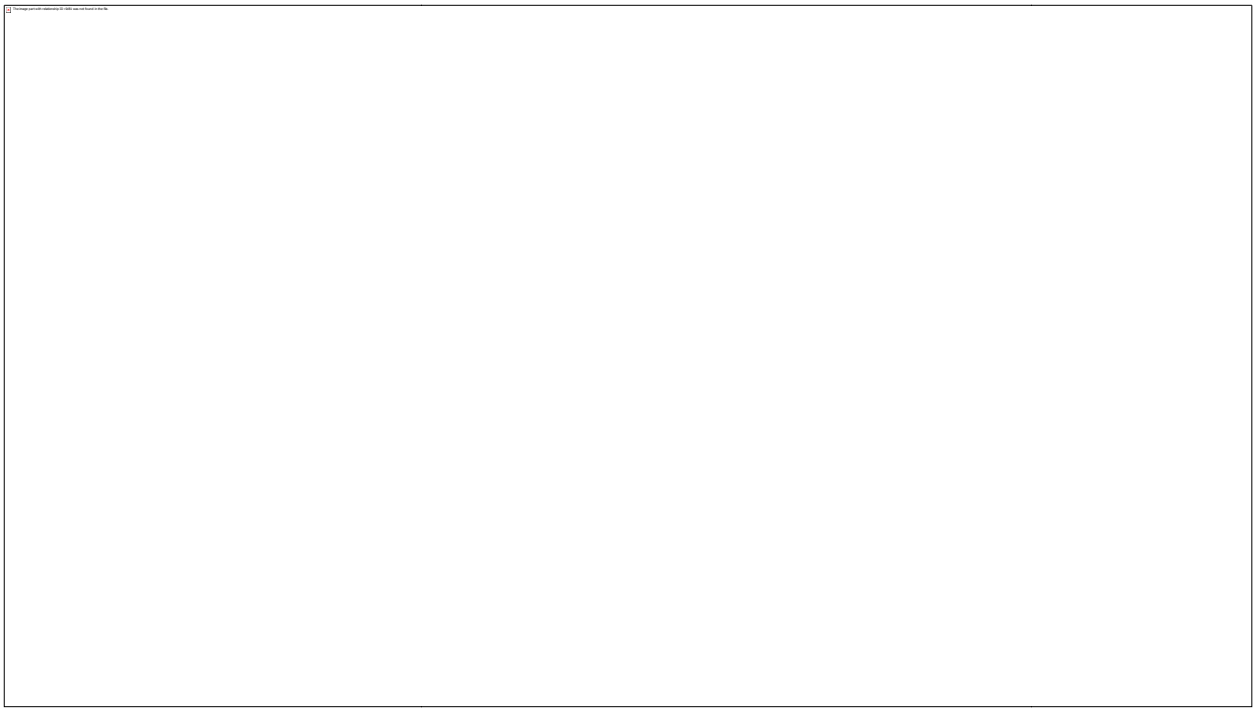

Figure 8-3: Pre-Processing window after execution

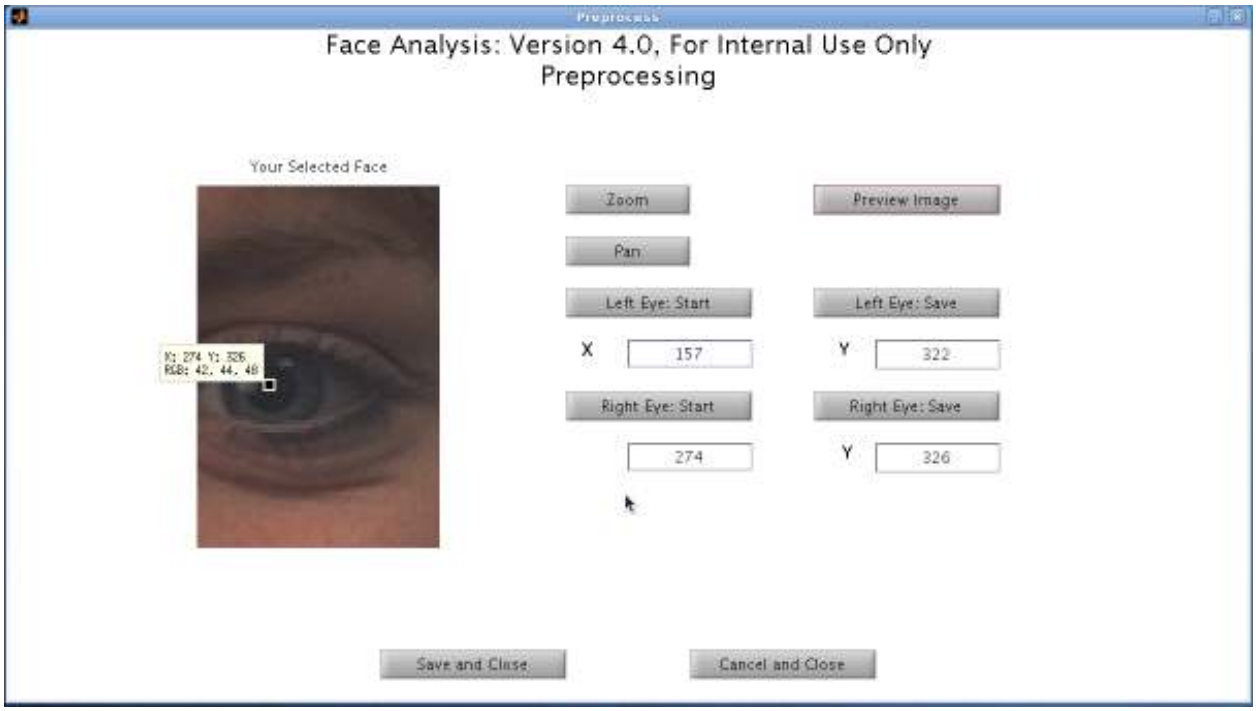

Figure 8-4: Pre-Processing window after both eye locations have been selected

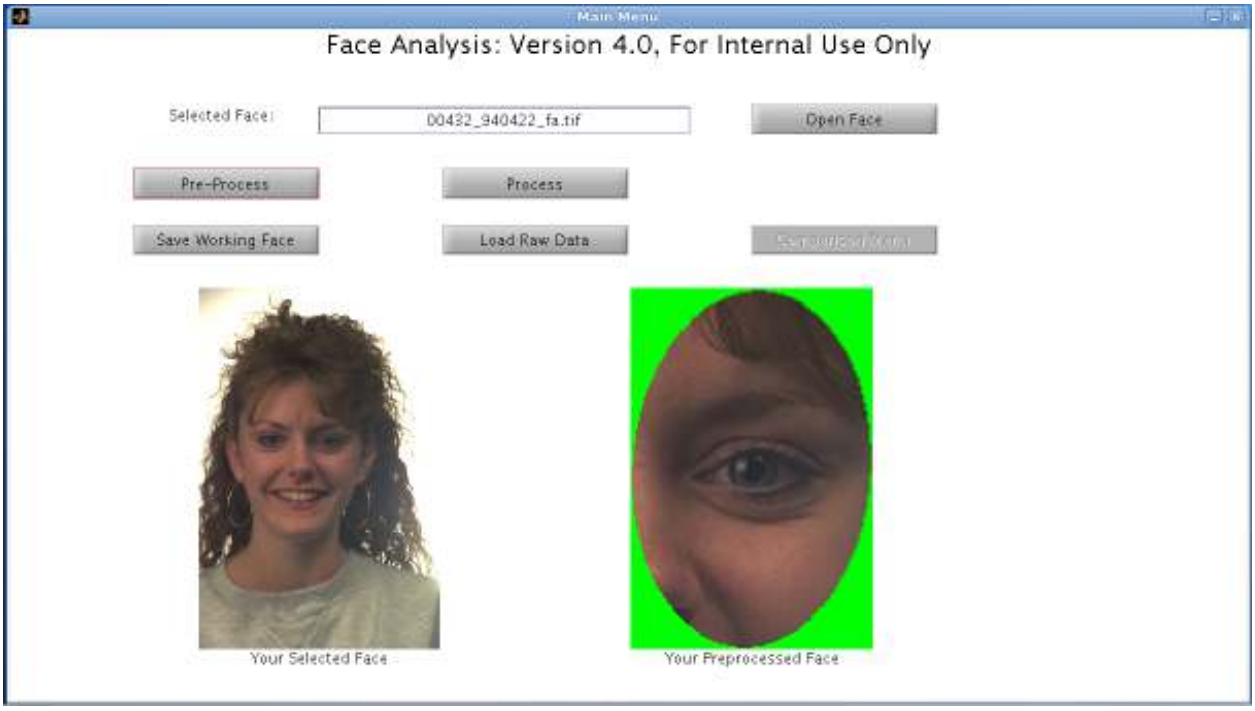

Figure 8-5: Main Menu after pre-processing is complete

From here, the processing window can now be executed. It is shown in Figure 8-6 after analysis is completed. The plot of normalized determinant is provided to the user for inspection prior to saving the slope dataset. This is so that an experienced user might quickly identify characteristics of a certain individual. This plot has not been subtracted from the average, which is done during the creation of the slope data. The slope data, however, may be stored to the file should the user desire.

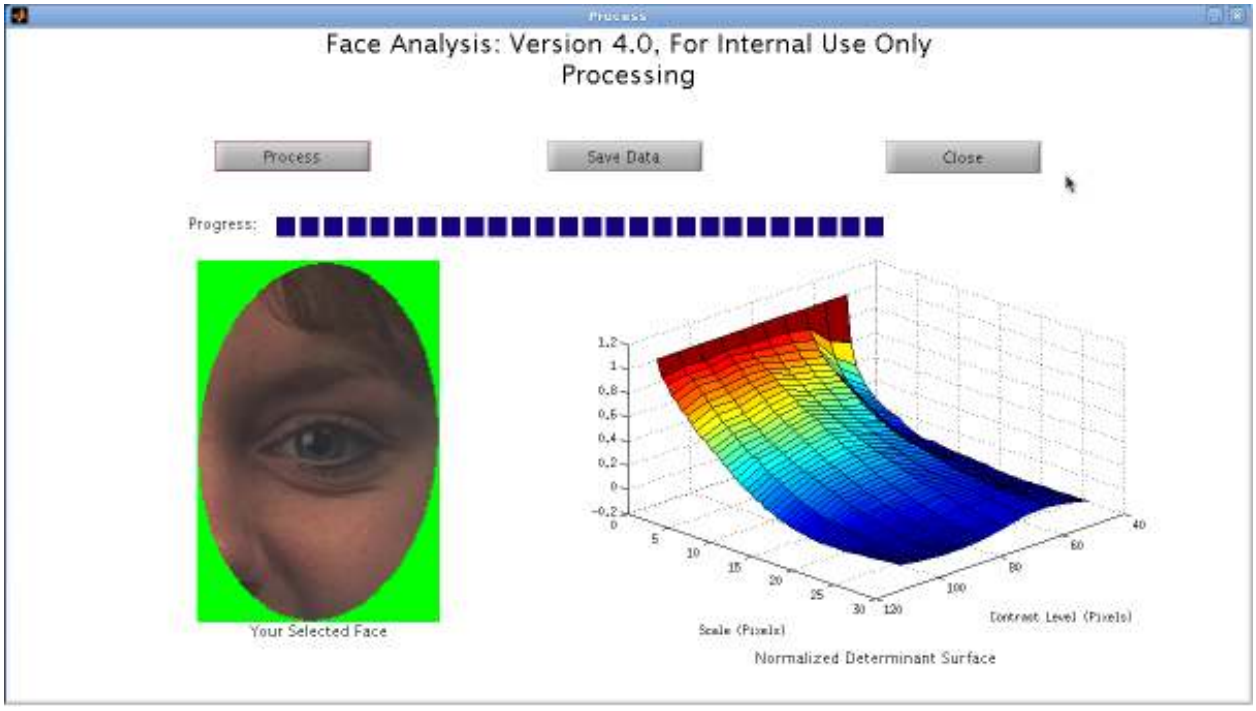

Figure 8-6: Processing complete

Once Processing is completed, the user is returned to the Main Window for the final time. The user may now open the comparison window (Figure 8-3). Here the user can compare the processed image to any dataset. Figure 8-4 shows the comparison window after a particular dataset has been loaded. Note that the list on the left is now populated so that the user may scroll through the list of images. After the comparison has been made to the database, any predicted matches populate the list on the right-hand side (Figure 8-5). The user may select any predicted match and click on the "preview" button to see a side-by-side visual of each original electronic image. (Note, this assumes that locations of each image are known and the images still are named correctly, otherwise an error message is displayed.)

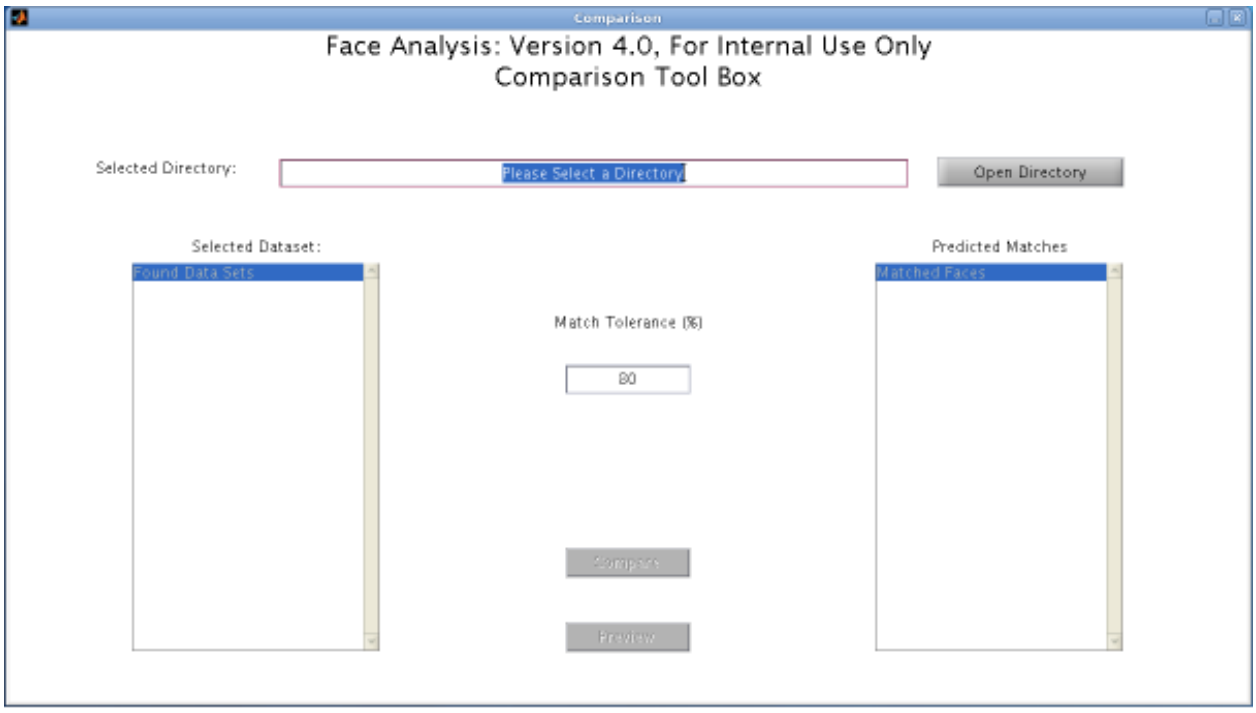

Figure 8-7: Comparison window after execution

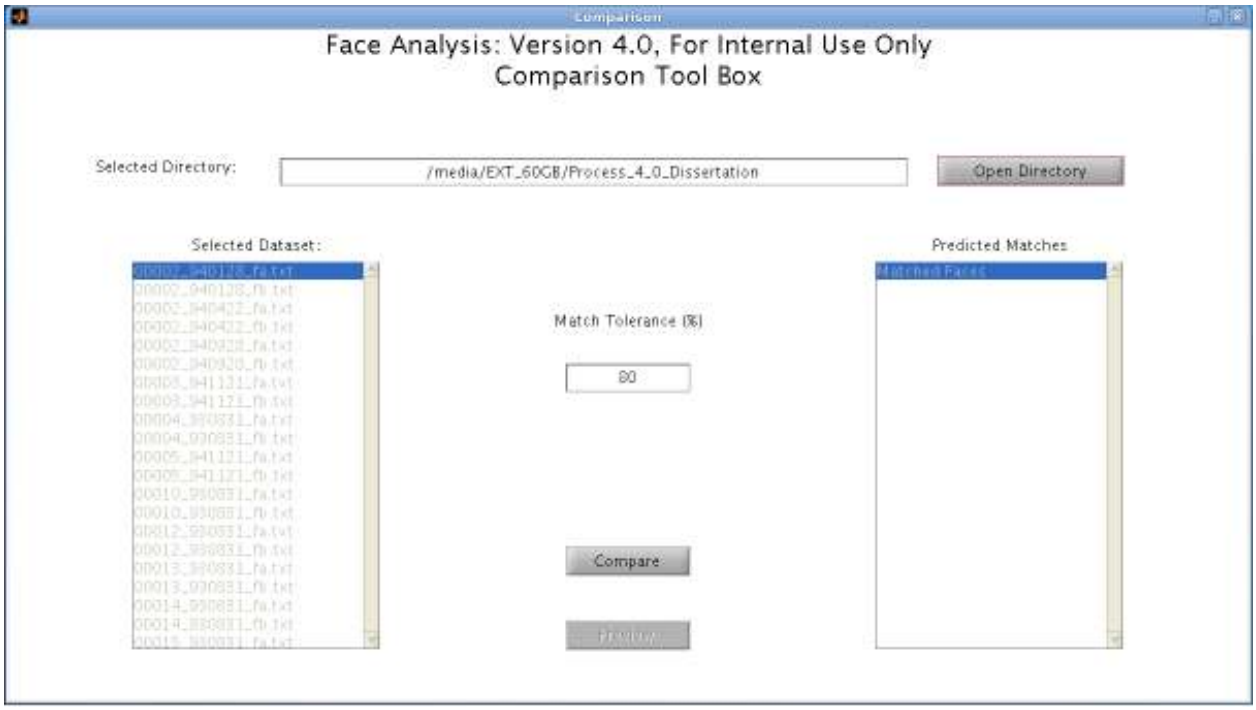

Figure 8-8: Comparison window after data set has been loaded

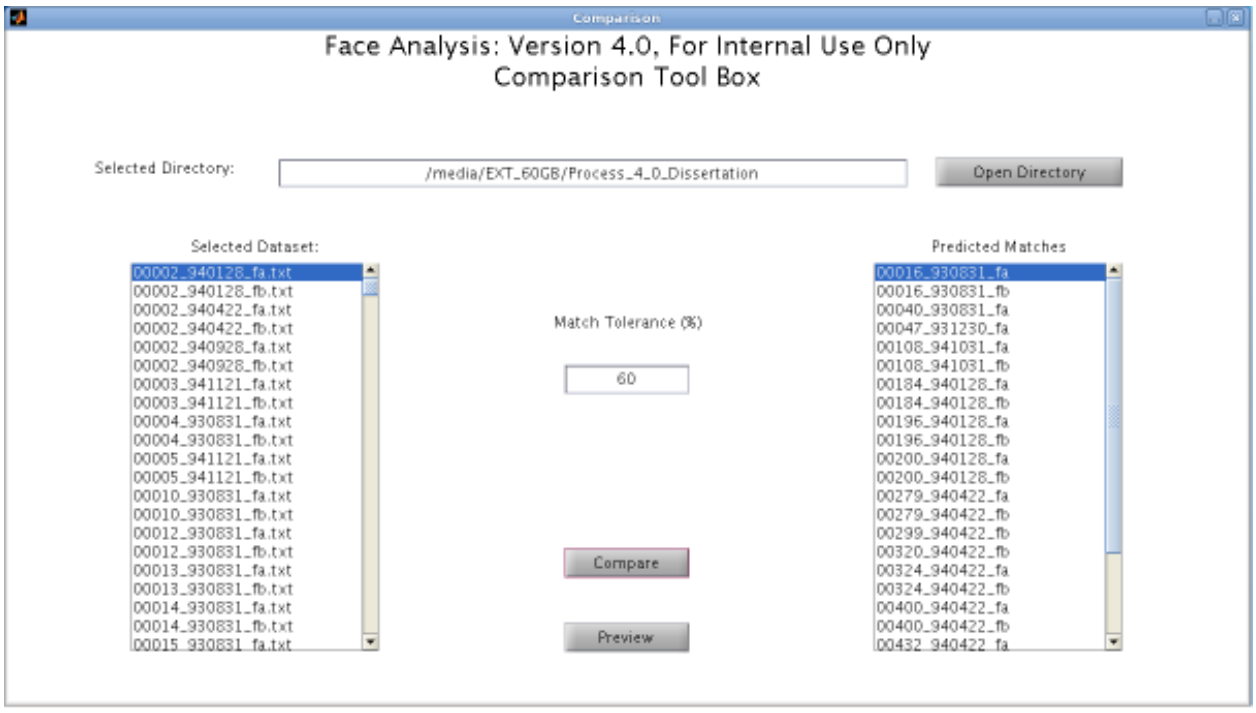

Figure 8-9: Comparison window after comparison has been made to database (note the tolerance was reduced to display a large list of predicted matches)

Once the comparison is complete, the user can return to the Main Menu and start the process over. The GUI has been completed using the best configuration possible from the various methods attempted. The entire algorithm in simple enough to be implemented in a fairly compact program which is in turn implemented in the GUI documented above.

9. Suggestions for Further Work

Of primary concern is the relatively low accuracy of the method selected. Ideally, any method would be at least 99% accurate on such a small database. The numbers of scales and binary images used created a normalized determinant consisting of 500 points. This generated 1,000 slope points which were used for comparison. The database in question consisted of 400 images. The number of markers (slopes) used to describe the database is quite similar to the database. The number of unique binary images and scales must be increased then.

It seems obvious that a high resolution image will increase the accuracy of the method. As previously stated, the FERET database was collected using first generation digital camera equipment in the early 90s. This only provided, on average, 127 pixels between the pupils. The digital camera technologies have gone from the sub-megapixel images in the FERET database to 10+ megapixel cameras which are readily available.

The greater number of pixels would mean that greater numbers of scales and binary images can be used without the overlapping experienced in the FERET Database images. Second, the number of pixels (discrete points) describing a particular feature in the image would also increase, which allows for more information to be conveyed through the normalized determinant curve.

Another way to improve the source images is to move from the 8 bit images (256 unique pixel values) stored in the FERET database to a modern digital image of 16 or 24 million unique pixel values. The substantial gain in the discrete number of possible pixel values would obviously increase the number of binary images which can be created without over lapping.

The other problem with the source images is the collection technique. The variance among the area of pixels associated with the subject's head and the remainder of the image is great for the FERET. If a database with a much stronger control on this variable were created, the quantity of pixels selected for analysis would be less deviant among images. While the algorithm used here does "non-dimensionalize" the effects of varying pixel distances between eyes, the fact remains that some images used have very low resolution faces. In other words, some digital images have a higher numbers of pixels in the face region than others.

The generation of the slope data and polling of the database for a match represent almost no increase in computational time to "test" an image after the normalized determinant has been

found. This very quick comparison scheme, then, warrants further study in more depth to refine the operating parameters. The number of scales and binary images was fixed prior to this experiment through a means which may have detracted from the full possibility of slope matching. Knowing now that slope matching presents the most probable means of greater matching ability, the parameter study could be adjusted to better determine the number of scales and binary images.

The number of possible slope values was chosen to be 2 (0 and 1). However, this number could easily be increased to 3 or 4 or more. The comparison methods which result from the slope data set would be virtually unaltered. Further, the number of random points per scale-binary image was selected to be 500,000 because of the work of Stoffa. It is quite possible this number can be reduced without great impact on the accuracy of the resulting data. Any reduction in the number of random points selected would reduce the over-all run time.

10. Works Cited

1. **Deal, John C.** Fractal Analysis of Fingerprints (Thesis). Morgantown WV : s.n., 2007.

2. **Stoffa, Joseph M.** Pattern Recognition via a Novel Markov Chain Monte Carlo Method Employing a First-Order Approximation of a Random Walk (Dissertation). Morgantown, WV : s.n., 2010.

3. The Facial Recognition Technology (FERET) Database. *NIST.gov.* [Online] [Cited: July 19, 2010.] http://www.itl.nist.gov/iad/humanid/feret/feret_master.html.

4. *The FERET Evaluation Methodology for Face-Recognition Algorithms.* **P. Jonathon Phillips, Hyeonjoon Moon, Syed A. Rizvi, Patrick J. Rauss.** 10, s.l. : IEEE Transcations of Pattern Analysis and Machine Intelligence, October 2000, Vol. 22.

5. **Phillips, P. Jonathon, et al.** *FRVT 2006 and ICE 2006 Large-Scale Results.* s.l. : National Institute for Standards and Technology, March 2007.

6. *An Introduction to Biometric Recognition.* **Anil K. Jain, Arun Ross, Salil Prabhakar.** 1, s.l. : IEEE Transactions on Circuits and Systems for Video Technology, January 2004, Vol. 14.

7. **Lander, Mark.** Where Little is Left Outside the Camera's Eye. *New York Times.* [Online] July 8, 2007. [Cited: July 19, 2010.] http://www.nytimes.com/2007/07/08/weekinreview/08landler.html.

8. TSA Contact Center - Frenquently Asked Questions. *Travel Security Agency (Contact Center).* [Online] [Cited: 1 16, 2011.] http://www.tsa.gov/travelers/customer/editorial_1029.shtm#2.

9. Growing Backlash Agaisnt TSA Pat-downs and Body Scanners. *www.cnn.com.* [Online] CNN, 11 20, 2010. [Cited: 1 16, 2011.] http://articles.cnn.com/2010-11-12/travel/travel.screening_1_body-scanners-pat-downs-travel-companies?_s=PM:TRAVEL.

10. **Krause, Mike.** Is Face Recognition Just High-tech Snake Oil? *Enter Stage Right.* [Online] January 14, 2002. [Cited: January 27, 2010.] http://www.enterstageright.com/archive/articles/0102/0102facerecog.htm.

11. **Graf, Rudolf f.** *Modern Dictionary of Electronics.* Woodburn MA : Butterworth-Heinemann, 1999. 0-7506-9866-7.

12. **James D. Foley, Andries Van Dam, John F. Hughes, Steven K. Feiner.** *Computer Graphics: Principles and Practice (2nd ed.).* s.l. : Addison-Wesley, June 1990. 0201121107.

13. **Blackburn, Duane M., Bone, Mike and Phillips, P. Jonathon.** Facial Recognition Vendor Test 2000: Evaluation Report. 2001.

14. **Phillips, P. Jonathon, et al.** Face Recognition Vendor Test 2002. March 2003.

15. *Identification of Human Faces.* **Goldstien, A. Jay, Harmon, Leon D. and Lesk, Ann B.** No. 5, s.l. : IEEE, May1971, Vol. Proceedings of the IEEE Vol. 59, pp. 748 - 760.

16. *Low-Dimensional Procedure for the Characterization of Human Faces.* **Sirovich, L. and Kirby, M.** 3, s.l. : Optical Society of America, March 1987, Vol. 4.

17. *A Random Walk through Eigenspace.* **Turk, Matthew.** 12, s.l. : IEICE Trans. Inf. & System., December 2001, Vols. E84-D.

18. *Recognizing Faces with PCA and ICA.* **Draper, Burce A., et al.** 1-2, s.l. : Computer Vision and Image Understanding, August 2003, Vol. 91.

19. **Delac, Kresimir, Grgic, Mislav and Liatsis, Panos.** Appearance-based Statistical Methods for Face Recognition. *47th International Symposium ELMAR-2005.* Zadar, Croatia : s.n., June 2005.

20. **Barlett, Marian Stewart, Movellan, Javier R. and Sejnowski, Terrence J.** Face Recognition by Independent Component Analysis. *IEEE Transactions on Neural Networks.* November 2002. Vol. 13, No 6.

21. **Bell, Anthony J. and Sejnowski, Terrance J.** An Information-Maximization Approach to Blind Seperation and Blind Deconvolution. *Neural Computation.* 1995. Vol. 7, 6.

22. **Hyvärinen, Aapo.** The Pixed-Point Algorithm and Maximum Likelihood Estimation for Independent Component Analysis. *Neural Processing Letters.* 1999, Vol. 10, 1-5.

23. **Cichocki, A. and Unbehauen, r. Rummert, E.** Robust Learning Algorithm for Blind Separation of Signals . *Electronic Leters.* August 1994. Vol. 30, 17. ISBN 0013-5194.

24. **Comon, Pierre.** Independent Component Analysis, A New Concept? *Signal Processing.* April 1994. Vol. 36, 3.

25. **Navarrete, Pablo and Ruiz-Del-Solar, Javier.** Analysis and Comparison of Eigenspace-Based Face Recognition Approaches. *International Journal of Pattern Recognition and Artificial Intelligence.* 2002. Vol. 16, 7.

26. **Wenyi Zhao, Arvindh Krishnaswamy, Rama Chellappa Daniel L. Swets, and John Weng.** Discriminant Analysis of Principal Components for Face Recognition. *Face Recognition.org.* [Online] April 1998. [Cited: 1 17, 2011.] http://dx.doi.org/10.1109/AFGR.1998.670971.

27. **Wiksott, Laurenz, et al.** Face Recognition by Elastic Bunch Graphing. *Intelligent Biometric Techniques in Fingerprint and Face Recognition.* s.l. : CRC Press, 1999. ISBN 0-8493-2055-0.

28. **Feder, Jens.** *Fractals.* s.l. : Plenum Press, 1989. 0-306-42851-2.

29. **Pearson, Karl.** The Problem of the Random Walk. *Nature.* 1905, Vol. 72.

30. **Charles M. Grinstead, J. Laurie Snell.** *Introduction to Probability.* s.l. : American Mathematical Society, July 1997.

31. **Reese, George.** Buffon's Needle: An Analysis and Simulation. *Office of Mathematics, Science, and Technology Education, University of Illinois.* [Online] [Cited: 01 21, 2011.] http://mste.illinois.edu/reese/buffon/buffon.html.

32. *The Monte Carlo Method.* **Nicholas Metropolis, S. Ulam.** 247, s.l. : Journal of the American Statiscal Association, 1949, Vol. 44.

33. *MATLAB R2009b User Documentation.* [Electronic Document] s.l. : Mathworks, 2009.

34. **Kreyszig, Erwin.** *Advance Engineering Mathematics.* s.l. : Wiley and Sons, 2006. 0-471-48885-2.

35. **John C, Strikwerda.** *Finite Difference Schemes and Partial Differential Equations, 2nd ed.* s.l. : Society for Industrial and Applied Mathematics, 2004. 978-0-8989716-39-9.

36. **John D. Anderson, Jr.** *Computational Fluid Dynamics.* s.l. : McGraw-Hill, 1995. 007-001685-2.

37. **Lu, Juwe, Plataniotis, K. N. and Venetsanopoulos, A. N.** Regularized Discriminant Analysis for the Small Sample Size Problem in Face Recognition. *Pattern Recognition Letters.* July 2003. Vol. 24.

38. **Lu, Juwei, Plataniotis, K. N. and Venetsanopoulos, A. N.** Boosting Linear Discriminant Analysis for Face Recognition. *Image Processing.* s.l. : ICIP 2003, 2003. Vol. 1.

39. **Barnsley, Michael, Hutchinson, John E. and Stenflot, Organ.** V-variable Fractals and Superfractrals. *Austrailan National University Mathematical Sciences Institute.* [Online] [Cited: July 21, 2010.] http://www.maths.anu.edu.au/~barnsley/pdfs/V-var_super_fractals.pdf.

Appendix A: List of Images Selected from the FERET Database

Number	Image Name
1	00002_940128_fa.tif
2	00002_940128_fb.tif
3	00002_940422_fa.tif
4	00002_940422_fb.tif
5	00002_940928_fa.tif
6	00002_940928_fb.tif
7	00003_941121_fa.tif
8	00003_941121_fb.tif
9	00004_930831_fa.tif
10	00004_930831_fb.tif
11	00005_941121_fa.tif
12	00005_941121_fb.tif
13	00010_930831_fa.tif
14	00010_930831_fb.tif
15	00012_930831_fa.tif
16	00012_930831_fb.tif
17	00013_930831_fa.tif
18	00013_930831_fb.tif
19	00014_930831_fa.tif
20	00014_930831_fb.tif
21	00015_930831_fa.tif
22	00015_930831_fb.tif
23	00016_930831_fa.tif
24	00016_930831_fb.tif
25	00019_930831_fa.tif
26	00019_930831_fb.tif
27	00019_940128_fa.tif
28	00019_940128_fb.tif
29	00019_940307_fa.tif
30	00019_940307_fb.tif
31	00020_930831_fa.tif
32	00020_930831_fb.tif
33	00023_930831_fa.tif
34	00023_930831_fb.tif
35	00026_930831_fa.tif
36	00026_930831_fb.tif
37	00028_940128_fa.tif
38	00028_940128_fb.tif
39	00029_930831_fa.tif
40	00029_930831_fb.tif
41	00029_940422_fa.tif
42	00029_940422_fb.tif
43	00029_940928_fa.tif
44	00029_940928_fb.tif
45	00029_941031_fa.tif
46	00029_941031_fb.tif
47	00029_960627_fa.tif
48	00029_960627_fb.tif
49	00038_930831_fa.tif
50	00038_930831_fb.tif
51	00038_940128_fa.tif
52	00038_940128_fb.tif
53	00039_930831_fa.tif
54	00039_930831_fb.tif
55	00040_930831_fa.tif
56	00040_930831_fb.tif
57	00042_930831_fa.tif
58	00042_930831_fb.tif
59	00047_931230_fa.tif
60	00047_931230_fb.tif
61	00050_931230_fa.tif
62	00050_931230_fb.tif
63	00093_941121_fa.tif
64	00093_941121_fb.tif
65	00095_940128_fa.tif
66	00095_940128_fb.tif
67	00107_941121_fa.tif
68	00107_941121_fb.tif
69	00108_941031_fa.tif
70	00108_941031_fb.tif
71	00140_941121_fa.tif
72	00140_941121_fb.tif
73	00146_941201_fa.tif
74	00146_941201_fb.tif
75	00146_960620_fa.tif
76	00146_960620_fb.tif
77	00157_940928_fa.tif
78	00157_940928_fb.tif
79	00162_940128_fa.tif
80	00162_940128_fb.tif
81	00166_931230_fa.tif
82	00166_931230_fb.tif
83	00183_940128_fa.tif
84	00183_940128_fb.tif
85	00184_940128_fa.tif
86	00184_940128_fb.tif
87	00188_940307_fa.tif
88	00188_940307_fb.tif
89	00191_940128_fa.tif
90	00191_940128_fb.tif
91	00193_940928_fa.tif
92	00193_940928_fb.tif
93	00195_940128_fa.tif

94	00195_940128_fb.tif
95	00196_940128_fa.tif
96	00196_940128_fb.tif
97	00199_940128_fa.tif
98	00199_940128_fb.tif
99	00200_940128_fa.tif
100	00200_940128_fb.tif
101	00203_940128_fa.tif
102	00203_940128_fb.tif
103	00214_940128_fa.tif
104	00214_940128_fb.tif
105	00218_940128_fa.tif
106	00218_940128_fb.tif
107	00220_940128_fa.tif
108	00220_940128_fb.tif
109	00221_940128_fa.tif
110	00221_940128_fb.tif
111	00223_940128_fa.tif
112	00223_940128_fb.tif
113	00229_940128_fa.tif
114	00229_940128_fb.tif
115	00234_940128_fa.tif
116	00234_940128_fb.tif
117	00237_940128_fa.tif
118	00237_940128_fb.tif
119	00238_940128_fa.tif
120	00238_940128_fb.tif
121	00242_940128_fa.tif
122	00242_940128_fb.tif
123	00244_940128_fa.tif
124	00244_940128_fb.tif
125	00246_940128_fa.tif
126	00246_940128_fb.tif
127	00247_941121_fa.tif
128	00247_941121_fb.tif
129	00250_940128_fa.tif
130	00250_940128_fb.tif
131	00253_940128_fa.tif
132	00253_940128_fb.tif
133	00254_940128_fa.tif
134	00254_940128_fb.tif
135	00256_940928_fa.tif
136	00256_940928_fb.tif
137	00258_940128_fa.tif
138	00258_940128_fb.tif
139	00259_940128_fa.tif
140	00259_940128_fb.tif

141	00264_940128_fa.tif
142	00264_940128_fb.tif
143	00268_940307_fa.tif
144	00268_940307_fb.tif
145	00268_960530_fa.tif
146	00268_960530_fb.tif
147	00272_940422_fa.tif
148	00272_940422_fb.tif
149	00278_940422_fa.tif
150	00278_940422_fb.tif
151	00279_940422_fa.tif
152	00279_940422_fb.tif
153	00282_940422_fa.tif
154	00282_940422_fb.tif
155	00285_940422_fa.tif
156	00285_940422_fb.tif
157	00291_940422_fa.tif
158	00291_940422_fb.tif
159	00297_940422_fa.tif
160	00297_940422_fb.tif
161	00299_940422_fa.tif
162	00299_940422_fb.tif
163	00301_940422_fa.tif
164	00301_940422_fb.tif
165	00304_940422_fa.tif
166	00304_940422_fb.tif
167	00313_940422_fa.tif
168	00313_940422_fb.tif
169	00320_940422_fa.tif
170	00320_940422_fb.tif
171	00321_940422_fa.tif
172	00321_940422_fb.tif
173	00324_940422_fa.tif
174	00324_940422_fb.tif
175	00327_940422_fa.tif
176	00327_940422_fb.tif
177	00334_940422_fa.tif
178	00334_940422_fb.tif
179	00336_940422_fa.tif
180	00336_940422_fb.tif
181	00337_940422_fa.tif
182	00337_940422_fb.tif
183	00346_940422_fa.tif
184	00346_940422_fb.tif
185	00347_940422_fa.tif
186	00347_940422_fb.tif
187	00348_940422_fa.tif

188	00348_940422_fb.tif
189	00350_940422_fa.tif
190	00350_940422_fb.tif
191	00351_940422_fa.tif
192	00351_940422_fb.tif
193	00361_940422_fa.tif
194	00361_940422_fb.tif
195	00362_940422_fa.tif
196	00362_940422_fb.tif
197	00365_940422_fa.tif
198	00365_940422_fb.tif
199	00381_940422_fa.tif
200	00381_940422_fb.tif
201	00383_940928_fa.tif
202	00383_940928_fb.tif
203	00388_940422_fa.tif
204	00388_940422_fb.tif
205	00398_940422_fa.tif
206	00398_940422_fb.tif
207	00400_940422_fa.tif
208	00400_940422_fb.tif
209	00408_940422_fa.tif
210	00408_940422_fb.tif
211	00430_940422_fa.tif
212	00430_940422_fb.tif
213	00432_940422_fa.tif
214	00432_940422_fb.tif
215	00436_940422_fa.tif
216	00436_940422_fb.tif
217	00443_940422_fa.tif
218	00443_940422_fb.tif
219	00445_940422_fa.tif
220	00445_940422_fb.tif
221	00454_940422_fa.tif
222	00454_940422_fb.tif
223	00467_940519_fa.tif
224	00467_940519_fb.tif
225	00468_940519_fa.tif
226	00468_940519_fb.tif
227	00468_941201_fa.tif
228	00468_941201_fb.tif
229	00468_960530_fa.tif
230	00468_960530_fb.tif
231	00468_960620_fa.tif
232	00468_960620_fb.tif
233	00469_940519_fa.tif
234	00469_940519_fb.tif
235	00469_941201_fa.tif
236	00469_941201_fb.tif
237	00469_960530_fa.tif
238	00469_960530_fb.tif
239	00469_960620_fa.tif
240	00469_960620_fb.tif
241	00470_940519_fa.tif
242	00470_940519_fb.tif
243	00472_960627_fa.tif
244	00472_960627_fb.tif
245	00474_940519_fa.tif
246	00474_940519_fb.tif
247	00478_940519_fa.tif
248	00478_940519_fb.tif
249	00479_940519_fa.tif
250	00479_940519_fb.tif
251	00483_940519_fa.tif
252	00483_940519_fb.tif
253	00490_940519_fa.tif
254	00490_940519_fb.tif
255	00493_940519_fa.tif
256	00493_940519_fb.tif
257	00494_940519_fa.tif
258	00494_940519_fb.tif
259	00496_940519_fa.tif
260	00496_940519_fb.tif
261	00499_940519_fa.tif
262	00499_940519_fb.tif
263	00500_940519_fa.tif
264	00500_940519_fb.tif
265	00500_960627_fa.tif
266	00500_960627_fb.tif
267	00501_940519_fa.tif
268	00501_940519_fb.tif
269	00505_940519_fa.tif
270	00505_940519_fb.tif
271	00508_960627_fa.tif
272	00508_960627_fb.tif
273	00510_940519_fa.tif
274	00510_940519_fb.tif
275	00511_940519_fa.tif
276	00511_940519_fb.tif
277	00515_940519_fa.tif
278	00515_940519_fb.tif
279	00518_940519_fa.tif
280	00518_940519_fb.tif
281	00519_940519_fa.tif

282	00519_940519_fb.tif
283	00520_940519_fa.tif
284	00520_940519_fb.tif
285	00521_940519_fa.tif
286	00521_940519_fb.tif
287	00523_940519_fa.tif
288	00523_940519_fb.tif
289	00526_940519_fa.tif
290	00526_940519_fb.tif
291	00527_940519_fa.tif
292	00527_940519_fb.tif
293	00529_940519_fa.tif
294	00529_940519_fb.tif
295	00530_940519_fa.tif
296	00530_940519_fb.tif
297	00531_960627_fa.tif
298	00531_960627_fb.tif
299	00532_940519_fa.tif
300	00532_940519_fb.tif
301	00533_940519_fa.tif
302	00533_940519_fb.tif
303	00536_940519_fa.tif
304	00536_940519_fb.tif
305	00538_940519_fa.tif
306	00538_940519_fb.tif
307	00542_940519_fa.tif
308	00542_940519_fb.tif
309	00543_940519_fa.tif
310	00543_940519_fb.tif
311	00543_960627_fa.tif
312	00543_960627_fb.tif
313	00545_940519_fa.tif
314	00545_940519_fb.tif
315	00546_940519_fa.tif
316	00546_940519_fb.tif
317	00547_940519_fa.tif
318	00547_940519_fb.tif
319	00548_940519_fa.tif
320	00548_940519_fb.tif
321	00549_940519_fa.tif
322	00549_940519_fb.tif
323	00554_940519_fa.tif
324	00554_940519_fb.tif
325	00556_940519_fa.tif
326	00556_940519_fb.tif
327	00557_940519_fa.tif
328	00557_940519_fb.tif

329	00558_940519_fa.tif
330	00558_940519_fb.tif
331	00560_940519_fa.tif
332	00560_940519_fb.tif
333	00561_940519_fa.tif
334	00561_940519_fb.tif
335	00565_940307_fa.tif
336	00565_940307_fb.tif
337	00565_940928_fa.tif
338	00565_940928_fb.tif
339	00565_941121_fa.tif
340	00565_941121_fb.tif
341	00566_940928_fa.tif
342	00566_940928_fb.tif
343	00566_941121_fa.tif
344	00566_941121_fb.tif
345	00567_940928_fa.tif
346	00567_940928_fb.tif
347	00568_940928_fa.tif
348	00568_940928_fb.tif
349	00568_941031_fa.tif
350	00568_941031_fb.tif
351	00569_940928_fa.tif
352	00569_940928_fb.tif
353	00570_940928_fa.tif
354	00570_940928_fb.tif
355	00571_940928_fa.tif
356	00571_940928_fb.tif
357	00577_940928_fa.tif
358	00577_940928_fb.tif
359	00578_940928_fa.tif
360	00578_940928_fb.tif
361	00579_941031_fa.tif
362	00579_941031_fb.tif
363	00581_940928_fa.tif
364	00581_940928_fb.tif
365	00582_940928_fa.tif
366	00582_940928_fb.tif
367	00583_940928_fa.tif
368	00583_940928_fb.tif
369	00584_940928_fa.tif
370	00584_940928_fb.tif
371	00585_940928_fa.tif
372	00585_940928_fb.tif
373	00586_940928_fa.tif
374	00586_940928_fb.tif
375	00587_940928_fa.tif

376	00587_940928_fb.tif
377	00588_940928_fa.tif
378	00588_940928_fb.tif
379	00588_941031_fa.tif
380	00588_941031_fb.tif
381	00588_941121_fa.tif
382	00588_941121_fb.tif
383	00590_940928_fa.tif
384	00590_940928_fb.tif
385	00592_940928_fa.tif
386	00592_940928_fb.tif
387	00593_941031_fa.tif
388	00593_941031_fb.tif
389	00594_940928_fa.tif
390	00594_940928_fb.tif
391	00594_941031_fa.tif
392	00594_941031_fb.tif
393	00594_941121_fa.tif
394	00594_941121_fb.tif
395	00595_940928_fa.tif
396	00595_940928_fb.tif
397	00596_940928_fa.tif
398	00596_940928_fb.tif
399	00596_941121_fa.tif
400	00596_941121_fb.tif

Appendix B: Source Codes (Batch Codes) Used for the Experiments in Section 5

Image Read-in and Cropping Algorithm
(makes use of GUI windows shown later in Appendix D)

```
% Batch Mode based on Version 2.0
clear
clc

tic;
preProcess=input('Do you need to pre process images (1 = yes, 0 = no): ');

if preProcess == 1
    [file_list path, index] = uigetfile({'*.jpg;*.tif;*.png;*.gif','All Image
Files';'*.*',...
        'All Files' },'Please Select a File','MultiSelect','On');

    %fix the fact that matlab seems to think that a single file doens't
    %need to be pulled in as a cell.  Hopefully they'll fix this in figure
    %versions.
    if ischar(file_list) == 1
        file_list = {file_list};
    end

    %find out how many elements there are
    [nil, num_files] = size(file_list);

    %iniatlize the cell array for the global file names
    full_name = cell(1,num_files);

    %loop over images
    for i = 1:num_files
        %pull in the image
        full_name(i) = strcat(path,file_list(i));
        image_io = imread(cell2mat(full_name(i)));

        %pass the image to the Pre-Process window and capture the output
        [image{i}, distance{i}] = Pre_Process(image_io);

        %clear the temp variable to avoid loop over conflicts.
        %clear image_io
    end

    % I'm going to save the image files according to the following
    % convention: file_name_XXX_YYYY.ext, where file_name is the
    % orriginal name of the file, ext is the three letter file
    % extension, XXX is the three whole digits of the eye center distance
    % and YYYY are the first four digits of the decimal of the ECD

    fprintf('Please select a directory to store the images in:\n');
    kbd = input('Paused, Press Enter to continue ');
    output_dir = uigetdir('./','Please select an output directory');

    output_global = cell(1,num_files);

    %write the results:
```

```
    for i = 1:num_files

        fprintf('Writing file %d of %d files\n', i, num_files);
        drawnow
        file = cell2mat(file_list(i));

        %strip out the file extensions
        j = 1;
        while strcmp(file(j),'.') ~= 1
            j = j+1;
        end
%
        %keep the period
        ext = file(j:end);
        file = file(1:j-1);
%
%         %get the ECD (eye center distance)
        ECD = cell2mat(distance(i));
        %-----New Way-----%
        % File Information regarding original image
        output_global{i} = strcat(output_dir,'/',file,'.txt');
        file = fopen(cell2mat(output_global(i)),'w');
        fprintf(file,'distance = %f\n', ECD);

        %--Store PreProcessed Image--%

        imDummy = cell2mat(image(i));
        [Y,X,RGB] = size(imDummy);

        Red = imDummy(:,:,1);
        Green = imDummy(:,:,2);
        Blue = imDummy(:,:,3);


        fprintf(file,'Y_size = %d, X_size = %d\n', Y, X);

        %--Red--%
        for j = 1:Y
            for k = 1:X
                fprintf(file,'%d,',Red(j,k));
            end
            fprintf(file,'\n');
        end

        %--Green--%
        for j = 1:Y
            for k = 1:X
                fprintf(file,'%d,',Green(j,k));
            end
            fprintf(file,'\n');
        end

        %--Blue--%
        for j = 1:Y
            for k = 1:X
```

```
                fprintf(file,'%d,',Blue(j,k));
            end
            fprintf(file,'\n');
        end

        fprintf(file,'Image Global = %s\n',cell2mat(full_name(i)));
        fclose(file);


        %cleanup for loopback
        clear file ext ECD ECD_whole ECD_4dec XXX YYYY
    end

end
```

Normalized Determinant Dataset Source Code
(Used in experiment 1 and subsequent experiments)

```
clc;
clear

tic;

time_old = 0;
iter = 0;

input_dir = '/media/EXT_60GB/PreProcess_4_0/';
%input_dir = './Raw_Data/';
%output_dir = '/media/EXT_60GB/Process_4_0_b/';

%assume this is  linux, I can fix later
[contents, num_images] = linux_list(input_dir);

num_scale = [5 10 20 40 50 10 10 10 10 10 10 20 20 40 40 50 40 50 40 50 60 60 60 ];
num_contrast = [5 5 5 5 5 3 5 10 15 20 30 20 30 20 30 30 10 10 15 15 5 10 15];

for k = 1:num_images
    try
        dummy = fopen(strcat(input_dir,'/',contents(k,:)),'r');
        fclose(dummy);
        clear dummy
    catch
        %remove bad file
        contents(k,:) = [];
    end
end

[num_images, nil] = size(contents);
total = num_images*50*11;

imGlobal = cell(1,num_images);

% for i = 1:numlabs
%     file_lab(i) = fopen(strcat('./Parallel_Run_Logs/lab_',num2str(labindex)...
%         ,'.txt'),'w');
% end

%create outter loop for managing the different cases

for count = 21:23
    %create the name of output directory
    output_dir = strcat('./Experiment 1/Case_',num2str(count),'_a/');
    mkdir(output_dir);

    time_old = toc;
    time_new = 0;
    dtime = 0;
    for k = 1:num_images
```

```
%new way to read in image:
file = fopen(strcat(input_dir,'/',contents(k,:)),'r');
trash = fscanf(file,'%s',[1,2]);
handles.distance = str2double(fscanf(file,'%s',[1,1]));
trash = fscanf(file,'%s',[1,2]);
Y_stop = str2double(fscanf(file,'%s',[1,1]));
trash = fscanf(file,'%s',[1,2]);
X_stop = str2double(fscanf(file,'%s',[1,1]));

%Get the Red,Green, and Blue values out of file:
R = zeros(Y_stop,X_stop);
for i = 1:Y_stop
    for j = 1:X_stop
        R(i,j) = fscanf(file,'%d,',[1,1]);
    end
end

R = uint8(R);

G = zeros(Y_stop,X_stop);
for i = 1:Y_stop
    for j = 1:X_stop
        G(i,j) = fscanf(file,'%d,',[1,1]);
    end
end

G = uint8(G);

B = zeros(Y_stop,X_stop);
for i = 1:Y_stop
    for j = 1:X_stop
        B(i,j) = fscanf(file,'%d,',[1,1]);
    end
end

B = uint8(B);

%store the image:
handles.image = uint8(zeros(Y_stop,X_stop,3));
handles.image(:,:,1) = R;
handles.image(:,:,2) = G;
handles.image(:,:,3) = B;

%and finally get the image global reference:
trash = fscanf(file,'%s',[1,3]);

imTemp = fscanf(file,'%c');
%remove the garbage at the front:
imTemp(1) = [];
imGlobal(k) = {imTemp};

% close out the file now that we're done reading
fclose(file);

num_pixels = 0;
```

```
pixel_total = 0;
for i =1:Y_stop
    for j = 1:X_stop
        if handles.image(i,j,2) == 255 && handles.image(i,j,1) == 0 && ...
                handles.image(i,j,3) == 0
            %don't want to do anything this is a green pixel
        else
            num_pixels = num_pixels + 1;
            pixel_total = pixel_total + double(rgb2gray(handles.image(i,j,:)));
        end
    end
end

average = pixel_total / num_pixels;

%calulate the standard deviation
stand_dev = 0;
for i =1:Y_stop
    for j = 1:X_stop
        if handles.image(i,j,2) == 255 && handles.image(i,j,1) == 0 && ...
                handles.image(i,j,3) == 0
            %don't want to do anything this is a green pixel
        else
            stand_dev = stand_dev+(double(rgb2gray(handles.image(i,j,:))...
                -average))^2;
        end
    end
end

stand_dev = sqrt(1/num_pixels*stand_dev);

%numbers = [-2.0 -1.6 -1.2 -0.8 -0.4 0 0.4 0.8 1.2 1.6 2.0];

%now 31 numbers
numbers = linspace(-2.0,2.0,num_contrast(count));
for i = 1:length(numbers)
    contrast(i) = average + numbers(i)*stand_dev;
end


for i = 1:length(contrast)
    if contrast(i) < 0
        contrast(i) = 0;
    elseif contrast(i) > 255
        contrast(i) = 255;
    end
end
%ok, need to non-dimen this:

%display(handles.distance);
scale = linspace(0,(handles.distance/4),num_scale(count));

%display(scale)

%num_contrast = length(contrast);
```

```
%num_scale = length(scale);
alpha_1 = zeros(num_contrast(count), num_scale(count));
alpha_2 = alpha_1;
beta = alpha_1;
scale_actual = alpha_1;
norm_det = alpha_1;
progress_percent = 0;

for i = 1:num_contrast(count)

    %open level out the image:
    image_temp = Contrast_Level(handles.image, contrast(i));
    for j = 1:num_scale(count)
        iter = iter + 1;
        %invoke the solver:
        [alpha_1(i,j), alpha_2(i,j), beta(i,j), scale_actual(i,j)] = ...
            outlined_RW(image_temp, scale(j));

        %find the normalized determinant
        norm_det(i,j) = (alpha_1(i,j)*alpha_2(i,j) - 0.25*beta(i,j)^2) / ...
            (alpha_1(i,1)*alpha_2(i,1));

    end
end
%     clear alpha_1 alpha_2 beta scale num_contrast numbers X_stop Y_stop
%     clear num_scale contrast average stand_dev handles.image pixel_total
%     clear num_pixels XXX YYYY

%fit some nice equations:
scale = (scale_actual) / (handles.distance/4);
%[num_contrast, num_scale] = size(norm_det);

for i = 1:num_contrast(count)
    for j = 1:num_scale(count)
        if isnan(norm_det(i,j)) == 1 || isinf(norm_det(i,j)) == 1
            norm_det(i,j) = 0;
        end
    end
end

temp = cell2mat(imGlobal(k));
for i = length(temp):-1:1
    if strcmp(temp(i),'/') == 1 || strcmp(temp(i),'\') == 1
        break
    end
end
handles.base_image.path = temp(1:i);
handles.base_image.name = temp(i+1:end);

%and now the output directory stuff:
pathname = output_dir;
%concotenate the file name
%find the distance:
j = 1;
while strcmp(contents(k,j),'.') ~= 1
```

```
            j = j + 1;
        end

        filename = contents(k,:);
        %     %         %save the findings
        %    data = [P1',P2',P3',P4',P5',P6',P7',P8',P9',r_square'];
        %   data = [alpha_1, alpha_2, beta, norm_det];
        data = norm_det;
        test = Faces_Write(data,handles.base_image.path,handles.base_image.name,...
            pathname,filename);

        %  Use this option to dump all raw data
        %    test =
Faces_Write_Process([],handles.base_image.path,handles.base_image.name,...
        %        pathname,filename,alpha_1,alpha_2,beta,norm_det, scale_actual);

        time_new = toc;
        dtime = time_new - time_old;

        fprintf('Case = %d image %d / %d complete with estimated %.2f hours remaining for
Case\n',count, k,...
            num_images,(num_images - k)*dtime/3600);

        time_old = time_new;

    end

end
% for i = 1:numlabs
%     fclose(file_lab(i));
% end


toc;
```

Experiment 2 Source Codes
(verifying ellipse and rectangle masking)

```
clear;
close all;
clc;

%results of experiment #1
num_scale = 50;
num_contrast = 10;

%previous generated list of images:
file_names = ['00244_940128_fa.tif'
    '00200_940128_fa.tif'
    '00324_940422_fb.tif'
    '00561_940519_fa.tif'
    '00595_940928_fb.tif'
    '00529_940519_fb.tif'
    '00472_960627_fb.tif'
    '00519_940519_fa.tif'
    '00020_930831_fb.tif'
    '00579_941031_fa.tif'];

num_files = 10;

out_dir = './Experiment 2';

%in this experiment, we will generate the normalized determinant surface
%for each fo the 10 images for both teh ellipse cutout ("standard") and
%square cutout, prooving there is little difference between the two.

%loop goes here
for k = 1:10

    %this will only work on my machine!

    %pre-done masked images:
    in_dir_a = '/media/EXT_60GB/PreProcess_4_0/'; %ellipse
    in_dir_b = '/media/EXT_60GB/PreProcess_3_0/'; %square

    out_file_a = strcat(file_names(k,1:end-4),'_ellipse.txt');
    out_file_b = strcat(file_names(k,1:end-4),'_square.txt');

    %read in the image data for the ellipse:
    fprintf('Analyzing image %d for ellipse\n', k);
    filename = strcat(in_dir_a,file_names(k,1:end-4),'.txt');
    file = fopen(filename,'r');

    %from experiment 1
    trash = fscanf(file,'%s',[1,2]);
    handles.distance = str2double(fscanf(file,'%s',[1,1]));
    trash = fscanf(file,'%s',[1,2]);
    Y_stop = str2double(fscanf(file,'%s',[1,1]));
    trash = fscanf(file,'%s',[1,2]);
```

```
X_stop = str2double(fscanf(file,'%s',[1,1]));

%Get the Red,Green, and Blue values out of file:
R = zeros(Y_stop,X_stop);
for i = 1:Y_stop
    for j = 1:X_stop
        R(i,j) = fscanf(file,'%d,',[1,1]);
    end
end

R = uint8(R);

G = zeros(Y_stop,X_stop);
for i = 1:Y_stop
    for j = 1:X_stop
        G(i,j) = fscanf(file,'%d,',[1,1]);
    end
end

G = uint8(G);

B = zeros(Y_stop,X_stop);
for i = 1:Y_stop
    for j = 1:X_stop
        B(i,j) = fscanf(file,'%d,',[1,1]);
    end
end

B = uint8(B);

%store the image:
handles.image = uint8(zeros(Y_stop,X_stop,3));
handles.image(:,:,1) = R;
handles.image(:,:,2) = G;
handles.image(:,:,3) = B;

%and finally get the image global reference:
trash = fscanf(file,'%s',[1,3]);

imTemp = fscanf(file,'%c');
%remove the garbage at the front:
imTemp(1) = [];
imGlobal(k) = {imTemp};

% close out the file now that we're done reading
fclose(file);

num_pixels = 0;
pixel_total = 0;
for i =1:Y_stop
    for j = 1:X_stop
        if handles.image(i,j,2) == 255 && handles.image(i,j,1) == 0 && ...
                handles.image(i,j,3) == 0
            %don't want to do anything this is a green pixel
        else
```

```
            num_pixels = num_pixels + 1;
            pixel_total = pixel_total + double(rgb2gray(handles.image(i,j,:)));
        end
    end
end

average = pixel_total / num_pixels;

%calulate the standard deviation
stand_dev = 0;
for i =1:Y_stop
    for j = 1:X_stop
        if handles.image(i,j,2) == 255 && handles.image(i,j,1) == 0 && ...
                handles.image(i,j,3) == 0
            %don't want to do anything this is a green pixel
        else
            stand_dev = stand_dev+(double(rgb2gray(handles.image(i,j,:))...
                -average))^2;
        end
    end
end

stand_dev = sqrt(1/num_pixels*stand_dev);

numbers = linspace(-2.0,2.0,num_contrast);
for i = 1:length(numbers)
    contrast(i) = average + numbers(i)*stand_dev;
end

for i = 1:length(contrast)
    if contrast(i) < 0
        contrast(i) = 0;
    elseif contrast(i) > 255
        contrast(i) = 255;
    end
end

scale = linspace(0,(handles.distance/4),num_scale);

alpha_1 = zeros(num_contrast, num_scale);
alpha_2 = alpha_1;
beta = alpha_1;
scale_actual = alpha_1;
norm_det = alpha_1;
progress_percent = 0;

for i = 1:num_contrast

    %open level out the image:
    image_temp = Contrast_Level(handles.image, contrast(i));
    for j = 1:num_scale
        %         iter = iter + 1;
        %invoke the solver:
        [alpha_1(i,j), alpha_2(i,j), beta(i,j), scale_actual(i,j)] = ...
            outlined_RW(image_temp, scale(j));
```

```
        %find the normalized determinant
        norm_det(i,j) = (alpha_1(i,j)*alpha_2(i,j) - 0.25*beta(i,j)^2) / ...
            (alpha_1(i,1)*alpha_2(i,1));

    end
end

for i = 1:num_contrast
    for j = 1:num_scale
        if isnan(norm_det(i,j)) == 1 || isinf(norm_det(i,j)) == 1
            norm_det(i,j) = 0;
        end
    end
end

%store the data for the ellipse
success = Faces_Write(norm_det,'','',out_dir,out_file_a);

%save the data to make a figure window
ellipse = norm_det;

%and repeat for the square cutouts...




%read in the image data for the square:
fprintf('Analyzing image %d for square\n', k);
filename = strcat(in_dir_b,file_names(k,1:end-4),'.txt');
file = fopen(filename,'r');

%from experiment 1
trash = fscanf(file,'%s',[1,2]);
handles.distance = str2double(fscanf(file,'%s',[1,1]));
trash = fscanf(file,'%s',[1,2]);
Y_stop = str2double(fscanf(file,'%s',[1,1]));
trash = fscanf(file,'%s',[1,2]);
X_stop = str2double(fscanf(file,'%s',[1,1]));

%Get the Red,Green, and Blue values out of file:
R = zeros(Y_stop,X_stop);
for i = 1:Y_stop
    for j = 1:X_stop
        R(i,j) = fscanf(file,'%d,',[1,1]);
    end
end

R = uint8(R);

G = zeros(Y_stop,X_stop);
for i = 1:Y_stop
    for j = 1:X_stop
```

```
        G(i,j) = fscanf(file,'%d,',[1,1]);
    end
end

G = uint8(G);

B = zeros(Y_stop,X_stop);
for i = 1:Y_stop
    for j = 1:X_stop
        B(i,j) = fscanf(file,'%d,',[1,1]);
    end
end

B = uint8(B);

%store the image:
handles.image = uint8(zeros(Y_stop,X_stop,3));
handles.image(:,:,1) = R;
handles.image(:,:,2) = G;
handles.image(:,:,3) = B;

%and finally get the image global reference:
trash = fscanf(file,'%s',[1,3]);

imTemp = fscanf(file,'%c');
%remove the garbage at the front:
imTemp(1) = [];
imGlobal(k) = {imTemp};

% close out the file now that we're done reading
fclose(file);

num_pixels = 0;
pixel_total = 0;
for i =1:Y_stop
    for j = 1:X_stop
        if handles.image(i,j,2) == 255 && handles.image(i,j,1) == 0 && ...
                handles.image(i,j,3) == 0
            %don't want to do anything this is a green pixel
        else
            num_pixels = num_pixels + 1;
            pixel_total = pixel_total + double(rgb2gray(handles.image(i,j,:)));
        end
    end
end

average = pixel_total / num_pixels;

%calulate the standard deviation
stand_dev = 0;
for i =1:Y_stop
    for j = 1:X_stop
        if handles.image(i,j,2) == 255 && handles.image(i,j,1) == 0 && ...
                handles.image(i,j,3) == 0
            %don't want to do anything this is a green pixel
```

```
        else
            stand_dev = stand_dev+(double(rgb2gray(handles.image(i,j,:)))...
                -average))^2;
        end
    end
end

stand_dev = sqrt(1/num_pixels*stand_dev);

numbers = linspace(-2.0,2.0,num_contrast);
for i = 1:length(numbers)
    contrast(i) = average + numbers(i)*stand_dev;
end

for i = 1:length(contrast)
    if contrast(i) < 0
        contrast(i) = 0;
    elseif contrast(i) > 255
        contrast(i) = 255;
    end
end

scale = linspace(0,(handles.distance/4),num_scale);

alpha_1 = zeros(num_contrast, num_scale);
alpha_2 = alpha_1;
beta = alpha_1;
scale_actual = alpha_1;
norm_det = alpha_1;
progress_percent = 0;

for i = 1:num_contrast

    %open level out the image:
    image_temp = Contrast_Level(handles.image, contrast(i));
    for j = 1:num_scale
        %         iter = iter + 1;
        %invoke the solver:
        [alpha_1(i,j), alpha_2(i,j), beta(i,j), scale_actual(i,j)] = ...
            outlined_RW(image_temp, scale(j));

        %find the normalized determinant
        norm_det(i,j) = (alpha_1(i,j)*alpha_2(i,j) - 0.25*beta(i,j)^2) / ...
            (alpha_1(i,1)*alpha_2(i,1));

    end
end

for i = 1:num_contrast
    for j = 1:num_scale
        if isnan(norm_det(i,j)) == 1 || isinf(norm_det(i,j)) == 1
            norm_det(i,j) = 0;
        end
    end
end
```

```
    %store the data for the ellipse
    success = Faces_Write(norm_det,'','',out_dir,out_file_b);

    %save the data to make a figure window
    square = norm_det;


    %make some pretty figure windows
    figure
    %set the size ahead of time:
    h = gcf;
    set(h,'position',[522,716,946,382]);
    subplot(1,2,1)
    [X,Y]=meshgrid(scale,contrast);
    surf(ellipse);
    xlabel('scales'); ylabel('contrasts');
    view([132,12]);
    subplot(1,2,2);
    surf(square);
    view([132,12]);
    xlabel('scales'); ylabel('contrasts');
end
```

Experiment 3 Source Codes
(calculating slopes and finding the optimal value of percent match among coefficients)

```
%program to calculate the average normalized det.

clear
close all
clc

in_dir = './Raw Norm Det/Process_a/';

%get the list of file names:
[file_names, num_images] = linux_list(in_dir);

%from experiment 1
num_contrast = 10;
num_scale = 50;

avg_norm_det = zeros(num_contrast,num_scale);

for k=1:num_images

    %load the dataset
    data = load([in_dir,file_names(k,:)]);

    avg_norm_det = avg_norm_det + data;

    %let's see the surface every 50 images
    if rem(k,50) == 0
        figure
        surf(avg_norm_det ./ k);
        view([132,12]);
        ylabel('Contrast'); xlabel('Scale');
    end

    fprintf('finished scanning %d of %d\n', k, num_images);
end

avg_norm_det = avg_norm_det ./ num_images;

%save './Raw Norm Det/avg_norm_det.mat' avg_norm_det;
file = './Raw Norm Det/avg_norm_det.csv';
output = fopen(file,'w');
for i = 1:num_contrast
    for j = 1:num_scale
        fprintf(output,'%f,', avg_norm_det(i,j));
    end
    fprintf(output,'\n');
end
```

```
%Dissertation Experiment 3, loads raw data from norm. det. and creates the
%slopes afte subtracting the average data.

tic;

clear;
close all;
clc;

%get the average data
avg_norm_det = load('./Raw Norm Det/avg_norm_det.csv');

%list all of the files from both perivously created data sets;
in_dir_a = './Raw Norm Det/Process_a/';
in_dir_b = './Raw Norm Det/Process_b/';

out_dir_a = './Experiment 3/slopes_a/';
out_dir_b = './Experiment 3/slopes_b/';

%get the lists
[file_names_a, num_images] = linux_list(in_dir_a);
[file_names_b, num_images] = linux_list(in_dir_b);

%from experiment1
num_contrast = 10;
num_scale = 50;

for k=1:num_images
    %start with set a
    norm_det = load([in_dir_a,file_names_a(k,:)]);

    %subtract the average
    norm_det = norm_det - avg_norm_det;

    %create the slopes using sam algorithm as first experiment 3

    out_file_a = strcat(out_dir_a,'cons/',file_names_a(k,1:end-4),'_con.csv');
    out_file_b = strcat(out_dir_a,'scales/',file_names_a(k,1:end-4),'_scale.csv');

    %determine the sloped data for normalized determinant:
    s_con = zeros(size(norm_det));
    s_scale = zeros(size(norm_det));

    %create "fake" boundary conditions so that conditionals aren't
    %necessary

    norm_use = zeros(num_contrast+2, num_scale+2);
    norm_use(2:end-1, 2:end-1) = norm_det;

    %and duplicate the boundary condition so
    %left hand side
    norm_use(2:end-1,1) = norm_det(:,1);

    %right hand side
    norm_use(2:end-1,end) = norm_det(:,end);
```

```
%top
norm_use(1,2:end-1) = norm_det(1,:);

%bottom
norm_use(end,2:end-1) = norm_det(end,:);

%denominators are constant due to setup
%it doesn't matter what these are.  We're only concerned with the sign
%and these are inherently always positive.  The reason this crops up is
%pre-processing the normalized determinants removes this information
%from this program.
%del_con = contrast(3) - contrast(1);
%del_scale = scale(4) - scale(2);

temp = 0.;

for i = 2:num_contrast+1
    for j = 2:num_scale+1
        temp = (norm_use(i+1,j)-norm_use(i-1,j));
        if temp > 0
            s_con(i-1,j-1) = 1;
        else
            s_con(i-1,j-1) = 0;
        end

        temp = (norm_use(i,j+1) - norm_use(i,j-1));
        if temp > 0
            s_scale(i-1,j-1) = 1;
        else
            s_scale(i-1,j-1) = 0;
        end
    end
end

%write out the slope data
%open the contrast file for writing
file = fopen(out_file_a,'w');
for i = 1:num_contrast
    for j = 1:num_scale
        fprintf(file,'%d,', s_con(i,j));
    end
    fprintf(file,'\n');
end
fclose(file);

file = fopen(out_file_b,'w');
for i = 1:num_contrast
    for j = 1:num_scale
        fprintf(file,'%d,', s_scale(i,j));
    end
    fprintf(file,'\n');
end
fclose(file);
```

```
%repeat with set b
norm_det = load([in_dir_a,file_names_b(k,:)]);

%subtract the average
norm_det = norm_det - avg_norm_det;

%create the slopes using sam algorithm as first experiment 3

out_file_a = strcat(out_dir_b,'cons/',file_names_b(k,1:end-4),'_con.csv');
out_file_b = strcat(out_dir_b,'scales/',file_names_b(k,1:end-4),'_scale.csv');

%determine the sloped data for normalized determinant:
s_con = zeros(size(norm_det));
s_scale = zeros(size(norm_det));

%create "fake" boundary conditions so that conditionals aren't
%necessary

norm_use = zeros(num_contrast+2, num_scale+2);
norm_use(2:end-1, 2:end-1) = norm_det;

%and duplicate the boundary condition so
%left hand side
norm_use(2:end-1,1) = norm_det(:,1);

%right hand side
norm_use(2:end-1,end) = norm_det(:,end);

%top
norm_use(1,2:end-1) = norm_det(1,:);

%bottom
norm_use(end,2:end-1) = norm_det(end,:);

%denominators are constant due to setup
%it doesn't matter what these are.  We're only concerned with the sign
%and these are inherently always positive.  The reason this crops up is
%pre-processing the normalized determinants removes this information
%from this program.
%del_con = contrast(3) - contrast(1);
%del_scale = scale(4) - scale(2);

temp = 0.;

for i = 2:num_contrast+1
    for j = 2:num_scale+1
        temp = (norm_use(i+1,j)-norm_use(i-1,j));
        if temp > 0
            s_con(i-1,j-1) = 1;
```

```
                else
                    s_con(i-1,j-1) = 0;
                end

                temp = (norm_use(i,j+1) - norm_use(i,j-1));
                if temp > 0
                    s_scale(i-1,j-1) = 1;
                else
                    s_scale(i-1,j-1) = 0;
                end
            end
        end

        %write out the slope data
        %open the contrast file for writing
        file = fopen(out_file_a,'w');
        for i = 1:num_contrast
            for j = 1:num_scale
                fprintf(file,'%d,', s_con(i,j));
            end
            fprintf(file,'\n');
        end
        fclose(file);

        file = fopen(out_file_b,'w');
        for i = 1:num_contrast
            for j = 1:num_scale
                fprintf(file,'%d,', s_scale(i,j));
            end
            fprintf(file,'\n');
        end
        fclose(file);

        fprintf('Finished %d / %d\n', k, num_images);
    end

    toc;
```

```
%Dissertation experiment #3, optimal value of nu for slope matching.

clear;
close all;
clc;

%start the runtime clock
tic;

%setup the list of thresholds
thresh = [70:100] ./ 100;

max_count = length(thresh);
```

```
%recall from exp. 1
num_scale = 50;
num_contrast = 10;


%setup the directories:

%remember the slopes of scale and contrast were seperated.  I did that so I
%could test each individually (through the process) and then combine them
%for a third run through the list of possible values
in_dir_a_con = './Experiment 3/slopes_a/cons/';
in_dir_a_scale = './Experiment 3/slopes_a/scales/';


%pop the lists of each directory:
[file_names_a_con, num_images_a_con] = linux_list(in_dir_a_con);
[file_names_a_scale, num_images_a_scale] = linux_list(in_dir_a_scale);

%make sure they are the same
if num_images_a_con ~= num_images_a_scale
    fprintf('Errory, group a directories not compatible\n');
end

in_dir_b_con = './Experiment 3/slopes_b/cons/';
in_dir_b_scale = './Experiment 3/slopes_b/scales/';


%pop the lists of each directory:
[file_names_b_con, num_images_b_con] = linux_list(in_dir_b_con);
[file_names_b_scale, num_images_b_scale] = linux_list(in_dir_b_scale);

%make sure they are the same
if num_images_b_con ~= num_images_b_scale
    fprintf('Errory, group b directories not compatible\n');
end

num_images_a = num_images_a_con;
num_images_b = num_images_b_con;

if num_images_a ~= num_images_b
   fprintf('Errory, directories not compatible\n');
end

num_images = num_images_a;

data_a_con = zeros(num_images,num_contrast,num_scale);
data_a_scale = data_a_con;
data_b_con = data_a_con;
data_b_scale = data_a_con;

%read in all of the data ahead of time:
for i=1:num_images
    data_a_con(i,:,:) = load([in_dir_a_con,file_names_a_con(i,:)]);
    data_a_scale(i,:,:) = load([in_dir_a_scale,file_names_a_scale(i,:)]);
```

```
    data_b_con(i,:,:) = load([in_dir_b_con,file_names_b_con(i,:)]);
    data_b_scale(i,:,:) = load([in_dir_b_scale,file_names_b_scale(i,:)]);

end

%start wiht the slopes from the contrasts first:
for count = 1:max_count
    time_a = toc;
    fprintf('Beginning comparison for contrast case %d\n', count);

    logfile = ['./Experiment 3/outputs/case_',num2str(count),'_contrast.csv'];
    file = fopen(logfile,'w');

    %prep the output file
    fprintf(file,'File_name,success\n');

    %now we compare them:
    tol = thresh(count)*num_scale*num_contrast;

    %perform the comparisons
    for i = 1:num_images %running over all "test" images (group a)

        %keep track of the number of possible matches
        num_match = 0;

        for j = 1:num_images %running over all "database" images (group b)

            %matlab allows us to test equality (boolean operations) through
            %entire arrays, so we just select the data which corresponds to
            %the test or database image.  If the match is true (1 to 1 or 0
            %to 0) it returns 1, 0 otherwise.  So then we just add up all
            %the ones (twice because of the matrix structure).
            matches = sum(sum(data_a_con(i,:,:) == data_b_con(j,:,:)));

            if matches >= tol %image i is predicted to match images j
                num_match = num_match + 1;
                %build the list of possible matches, strip out the fa and
                %fb
                check_names(num_match,:) = file_names_b_con(j,1:12);
            end

        end

        %now how many matches do we expect?  Only two for success, same as
        %before
        test_image = file_names_a_con(i,1:12);

        if num_match == 2
            %check to make sure we've only the same family
            if ( (strcmp(test_image,check_names(1,:)) == 1)...
                    && (strcmp(test_image,check_names(2,:)) == 1) )
                %success
                fprintf(file,'%s,%d\n',file_names_a_con(i,1:15),1);
            else %must have had erroneous matches
                fprintf(file,'%s,%d\n',file_names_a_con(i,1:15),0);
```

```
            end

        else %it failed
            fprintf(file,'%s,%d\n',file_names_a_con(i,1:15),0);
        end
    end


    %close the output file
    fclose(file);
    time_b = toc;
    fprintf('Case for Contrast completed in %.2f seconds\n',...
        time_b - time_a);
end




%move onto the scales
for count = 1:max_count
    time_a = toc;
    fprintf('Beginning comparison for slope case %d\n', count);

    logfile = ['./Experiment 3/outputs/case_',num2str(count),'_slope.csv'];
    file = fopen(logfile,'w');

    %prep the output file
    fprintf(file,'File_name,success\n');

    %now we compare them:
    tol = thresh(count)*num_scale*num_contrast;

    %perform the comparisons
    for i = 1:num_images %running over all "test" images (group a)

        %keep track of the number of possible matches
        num_match = 0;

        for j = 1:num_images %running over all "database" images (group b)

            %matlab allows us to test equality (boolean operations) through
            %entire arrays, so we just select the data which corresponds to
            %the test or database image.  If the match is true (1 to 1 or 0
            %to 0) it returns 1, 0 otherwise.  So then we just add up all
            %the ones (twice because of the matrix structure).
            matches = sum(sum(data_a_scale(i,:,:) == data_b_scale(j,:,:)));

            if matches >= tol %image i is predicted to match images j
                num_match = num_match + 1;
                %build the list of possible matches, strip out the fa and
                %fb
                check_names(num_match,:) = file_names_b_scale(j,1:12);
            end

        end
```

```
        %now how many matches do we expect?  Only two for success, same as
        %before
        test_image = file_names_a_scale(i,1:12);

        if num_match == 2
            %check to make sure we've only the same family
            if ( (strcmp(test_image,check_names(1,:)) == 1)...
                    && (strcmp(test_image,check_names(2,:)) == 1) )
                %success
                fprintf(file,'%s,%d\n',file_names_a_scale(i,1:15),1);
            else %must have had erroneous matches
                fprintf(file,'%s,%d\n',file_names_a_scale(i,1:15),0);
            end

        else %it failed
            fprintf(file,'%s,%d\n',file_names_a_scale(i,1:15),0);
        end
    end


    %close the output file
    fclose(file);
    time_b = toc;
    fprintf('Case_1 for Scale completed in %.2f seconds\n',...
        time_b - time_a);
end




%finish with the combniations
for count = 1:max_count
    time_a = toc;
    fprintf('Beginning comparison for combniation case %d\n', count);

    logfile = ['./Experiment 3/outputs/case_',num2str(count),'_combo.csv'];
    file = fopen(logfile,'w');

    %prep the output file
    fprintf(file,'File_name,success\n');

    %now we compare them:
    tol = thresh(count)*num_scale*num_contrast*2;

    %perform the comparisons
    for i = 1:num_images %running over all "test" images (group a)

        %keep track of the number of possible matches
        num_match = 0;

        for j = 1:num_images %running over all "database" images (group b)

            %matlab allows us to test equality (boolean operations) through
            %entire arrays, so we just select the data which corresponds to
```

```
        %the test or database image.  If the match is true (1 to 1 or 0
        %to 0) it returns 1, 0 otherwise.  So then we just add up all
        %the ones (twice because of the matrix structure).
        matches = sum(sum(data_a_scale(i,:,:) == data_b_scale(j,:,:)));
        matches = matches + ...
            sum(sum(data_a_con(i,:,:) == data_b_con(j,:,:)));

        if matches >= tol %image i is predicted to match images j
            num_match = num_match + 1;
            %build the list of possible matches, strip out the fa and
            %fb
            check_names(num_match,:) = file_names_b_scale(j,1:12);
        end

    end

    %now how many matches do we expect?  Only two for success, same as
    %before
    test_image = file_names_a_scale(i,1:12);

    if num_match == 2
        %check to make sure we've only the same family
        if ( (strcmp(test_image,check_names(1,:)) == 1)...
                && (strcmp(test_image,check_names(2,:)) == 1) )
            %success
            fprintf(file,'%s,%d\n',file_names_a_scale(i,1:15),1);
        else %must have had erroneous matches
            fprintf(file,'%s,%d\n',file_names_a_scale(i,1:15),0);
        end

    else %it failed
        fprintf(file,'%s,%d\n',file_names_a_scale(i,1:15),0);
    end
end


%close the output file
fclose(file);
time_b = toc;
fprintf('Case_1 for Combination completed in %.2f seconds\n',...
    time_b - time_a);
end
```

Experiment 4 Source Codes
(optimal polynomial order)

```
%Dissertation Experiment 4, create a set of data which has been fit with
%increasing orders of polynomials from 5 to 9.

tic;

clear;
close all;
clc;

%list all of the files from both perivously created data sets;
in_dir = './Raw Norm Det/Process_a/';
%in_dir_b = './Raw Norm Det/Process_b/';

%get the lists
[file_names_a, num_images] = linux_list(in_dir);
%[file_names_b, num_images] = linux_list(in_dir_b);

%from experiment1
num_contrast = 10;
num_scale = 50;

%note, how do we "plot" normalized determinant?  Are we plotting with the
%raw arrays of contrast and scale or the iterated ones?  In order for these
%coefficients to "non-dimensional," we should only plot with iterated value
%so that the space between discrete points in the norm det curve are
%equidistant in computational space.

%second note, we are ignoring the constant because it is forced to be
%almost one everytime.  So the order+1 constant is this constant.
%in order words if the order were 5, the actual form matlab produces is:
%c_1 x^5 + c_2 x^4 + c_3 x^3 + c_4 x^2 + c_5 x + x_6

scale = 1:1:num_scale;
contrast = 1:1:num_contrast;

%set a

%fit polys
for i = 5:9
    a = toc;
    %create the output directory
    out_dir = ['./Experiment 4/Poly_',num2str(i),'/set_a/'];

    for k = 1:num_images
        norm_det = load([in_dir,file_names_a(k,:)]);

        %poly nomical coefficients
        p = make_fit(scale,norm_det,i,num_contrast);

        %now we need to write them to file:
        output = fopen([out_dir,file_names_a(k,1:end-4),'.csv'],'w');
```

```
        for n = 1:i
            for m = 1:num_contrast
                fprintf(output,'%e,',p(n,m));
            end
            fprintf(output,'\n');
        end
        fclose(output);
    end


    b = toc;
    fprintf('Finished fitting all %d images for order %d in %f seconds\n'...
        ,num_images,i,b-a);
end %outter polyfit loop

fprintf('Finished set a\n');

%set b
in_dir = './Raw Norm Det/Process_b/';
for i = 5:9
    a = toc;
    %create the output directory
    out_dir = ['./Experiment 4/Poly_',num2str(i),'/set_b/'];

    for k = 1:num_images
        norm_det = load([in_dir,file_names_a(k,:)]);

        %poly nomical coefficients
        p = make_fit(scale,norm_det,i,num_contrast);

        %now we need to write them to file:
        output = fopen([out_dir,file_names_a(k,1:end-4),'.csv'],'w');
        for n = 1:i
            for m = 1:num_contrast
                fprintf(output,'%e,',p(n,m));
            end
            fprintf(output,'\n');
        end
        fclose(output);
    end


    b = toc;
    fprintf('Finished fitting all %d images for order %d in %f seconds\n'...
        ,num_images,i,b-a);
end %outter polyfit loop
fprintf('Finished set b\n');
   toc;



%Dissertation Experiment 4, this half of the program will analyze the data
%sets by performing the standard comparisons with the data values preset in
%the disseration.  The order which provides the highest order will then be
%selected.
```

```
tic;

clear;
close all;
clc;

%from experiment1
num_contrast = 10;
num_scale = 50;

nu = 0.75;
tol = 0.25;

for i = 5:9
    a = toc;
    %create the output directory
    in_dir_a = ['./Experiment 4/Poly_',num2str(i),'/set_a/'];
    in_dir_b = ['./Experiment 4/Poly_',num2str(i),'/set_b/'];

    %get the file list:
    [file_names_a, num_images] = linux_list(in_dir_a);
    [file_names_b, num_images] = linux_list(in_dir_b);

    %setup the output file
    outfile = ['./Experiment 4/Poly_',num2str(i),'_Results.csv'];

    %preload the database:
    for k = 1:num_images
        data_b(k,:,:) = load([in_dir_a,file_names_a(k,:)]);
    end

    %open the output file for writing
    file = fopen(outfile,'w');
    for k = 1:num_images
        %load the coefficient data:
        data_a = load([in_dir_a,file_names_a(k,:)]);

        %as always, b is the "database" and a's are the test images.
        %let the comparisons begin

        num_matches = 0;
        [M N] = size (data_a); %I could predict it knowing the value of i
        for j = 1:num_images %run a against all b
            match_count = 0;
            for m = 1:M
                for n = 1:N
                    temp = abs((data_a(m,n) - data_b(j,m,n)) / data_a(m,n));
                    if temp <= tol
                        match_count = match_count + 1;
                    end
                end
            end

            %is image b(k) predicted to match the test image, a?
```

```
            if match_count >= nu*M*N %yes
                %increment the counter of possible matches
                num_matches = num_matches + 1;
                %add the name to the list of possible match file names
                file_list_2(num_matches,:) = file_names_b(k,1:12);
            end

        end %loop to run a image against database

        %do we only have 2 predicted matches, and are they correct?
        %write out the result

        test_image = file_names_a(k,1:12);
        if num_matches == 2 %possible correct match
            %check to make sure it's only matched to itself
              if ( (strcmp(test_image,file_list_2(1,:)) == 1)...
                    && (strcmp(test_image,file_list_2(2,:)) == 1) )
                %success
                fprintf(file,'%s,%d\n',file_names_a(k,1:15),1);
              else
                  fprintf(file,'%s,%d\n',file_names_a(k,1:15),0);
              end

        else %total failure
            fprintf(file,'%s,%d\n',file_names_a(k,1:15),0);
        end

    end %loop over all a images

    %close this output file and prep for the next order of polynomial
    fclose(file);
    clear data_b;
end %outter loop
```

Experiment 5 Source Codes

(Optimal values of coefficient match rate and allowable percent difference for direct normalized determinant, and curve fitting methods)

```
%Dissertation Experiment 5., this half of the program will be used with the
%raw normalized determinant data.  The second half will be used with teh
%curve fit data.  Note that the second half will be almost a direct copy of
%the source used during experiment 4.

tic;

clear;
close all;
clc;

%from experiment1
num_contrast = 10;
num_scale = 50;

%nu_array = [0.75:0.01:0.85 .90 .95 1];
%i_max = length(nu_array);
mu_array = [0.1 0.15 0.2:0.01:0.3 0.35];
i_max = length(mu_array);
%tol = 0.25;

%create the output directory
in_dir_a = './Raw Norm Det/Process_a/';
in_dir_b = './Raw Norm Det/Process_b/';

%get the file list:
[file_names_a, num_images] = linux_list(in_dir_a);
[file_names_b, num_images] = linux_list(in_dir_b);

%preload the database:
for k = 1:num_images
    data_b(k,:,:) = load([in_dir_a,file_names_a(k,:)]);
end

%note this program can be used to find both NU and MU.  Simply swap the
%comments around as needed.

for i = 1:i_max
    a = toc;
    %nu = nu_array(i);

    %from first part of experiment, mu = 0.8 is best
    nu = 0.8;
    tol = mu_array(i);

    %setup the output file
    %done fine mu
    %outfile = ['./Experiment 5/No_Pre/Find_NU_Case_',num2str(i),'.csv'];
    outfile = ['./Experiment 5/No_Pre/Find_MU_Case_',num2str(i),'.csv'];
```

```
%open the output file for writing
file = fopen(outfile,'w');
for k = 1:num_images
    %load the coefficient data:
    data_a = load([in_dir_a,file_names_a(k,:)]);

    %as always, b is the "database" and a's are the test images.
    %let the comparisons begin

    num_matches = 0;
    [M N] = size (data_a); %I could predict it knowing the value of i
    for j = 1:num_images %run a against all b
        match_count = 0;
        for m = 1:M
            for n = 1:N
                temp = abs((data_a(m,n) - data_b(j,m,n)) / data_a(m,n));
                if temp <= tol
                    match_count = match_count + 1;
                end
            end
        end

        %is image b(k) predicted to match the test image, a?
        if match_count >= nu*M*N %yes
            %increment the counter of possible matches
            num_matches = num_matches + 1;
            %add the name to the list of possible match file names
            file_list_2(num_matches,:) = file_names_b(k,1:12);
        end

    end %loop to run a image against database

    %do we only have 2 predicted matches, and are they correct?
    %write out the result

    test_image = file_names_a(k,1:12);
    if num_matches == 2 %possible correct match
        %check to make sure it's only matched to itself
        if ( (strcmp(test_image,file_list_2(1,:)) == 1)...
                && (strcmp(test_image,file_list_2(2,:)) == 1) )
            %success
            fprintf(file,'%s,%d\n',file_names_a(k,1:15),1);
        else
            fprintf(file,'%s,%d\n',file_names_a(k,1:15),0);
        end

    else %total failure
        fprintf(file,'%s,%d\n',file_names_a(k,1:15),0);
    end

end %loop over all a images

%close this output file and prep for the next order of polynomial
```

```
    fclose(file);
    %clear data_b;
    b = toc;
    fprintf('Finished all %d images for case %d in %f seconds\n',...
        num_images, i, b-a);
end %outter loop



%Dissertation Experiment 5., this half of the program will be used with the
%curve fit data from exp. 4.

tic;

clear;
close all;
clc;

%from experiment1
num_contrast = 10;
num_scale = 50;

%nu_array = [0.65 0.7:0.01:0.8 0.85 .90 .95 1];
%i_max = length(nu_array);
mu_array = [0.1 0.15 0.2:0.01:0.3 0.35];
i_max = length(mu_array);
%tol = 0.25;

%create the output directory
in_dir_a = './Experiment 4/Poly_7/set_a/';
in_dir_b = './Experiment 4/Poly_7/set_b/';

%get the file list:
[file_names_a, num_images] = linux_list(in_dir_a);
[file_names_b, num_images] = linux_list(in_dir_b);

%preload the database:
for k = 1:num_images
    data_b(k,:,:) = load([in_dir_a,file_names_a(k,:)]);
end

%note this program can be used to find both NU and MU.  Simply swap the
%comments around as needed.

for i = 1:i_max
    a = toc;
    %nu = nu_array(i);

    %from first part of experiment, nu = 0.75 is best
    nu = 0.75;
    tol = mu_array(i);

    %setup the output file
    %done fine mu
    %outfile = ['./Experiment 5/Curves/Find_NU_Case_',num2str(i),'.csv'];
```

```
outfile = ['./Experiment 5/Curves/Find_MU_Case_',num2str(i),'.csv'];


%open the output file for writing
file = fopen(outfile,'w');
for k = 1:num_images
    %load the coefficient data:
    data_a = load([in_dir_a,file_names_a(k,:)]);

    %as always, b is the "database" and a's are the test images.
    %let the comparisons begin

    num_matches = 0;
    [M N] = size (data_a); %I could predict it knowing the value of i
    for j = 1:num_images %run a against all b
        match_count = 0;
        for m = 1:M
            for n = 1:N
                temp = abs((data_a(m,n) - data_b(j,m,n)) / data_a(m,n));
                if temp <= tol
                    match_count = match_count + 1;
                end
            end
        end

        %is image b(k) predicted to match the test image, a?
        if match_count >= nu*M*N %yes
            %increment the counter of possible matches
            num_matches = num_matches + 1;
            %add the name to the list of possible match file names
            file_list_2(num_matches,:) = file_names_b(k,1:12);
        end

    end %loop to run a image against database

    %do we only have 2 predicted matches, and are they correct?
    %write out the result

    test_image = file_names_a(k,1:12);
    if num_matches == 2 %possible correct match
        %check to make sure it's only matched to itself
        if ( (strcmp(test_image,file_list_2(1,:)) == 1)...
                && (strcmp(test_image,file_list_2(2,:)) == 1) )
            %success
            fprintf(file,'%s,%d\n',file_names_a(k,1:15),1);
        else
            fprintf(file,'%s,%d\n',file_names_a(k,1:15),0);
        end

    else %total failure
        fprintf(file,'%s,%d\n',file_names_a(k,1:15),0);
    end

end %loop over all a images
```

```
    %close this output file and prep for the next order of polynomial
    fclose(file);
    %clear data_b;
    b = toc;
    fprintf('Finished all %d images for case %d in %f seconds\n',...
        num_images, i, b-a);
end %outter loop
```

Appendix C: Ancillary and Support Function Source Codes

Random Walk Function

```
function [alpha_1, alpha_2, beta, scale] = outlined_RW(image, scale)


   %clc
   %Here, alpha_ww, alpha_bb, and beta are the probability counters created
   %during the random walk proces.  The image is assumed to be an actual
   %image.

   num_iter = 100000;
   %j = 1;

   %--been screwing around with this: 6/17/2008 NJH
   %scale = 0:.25:20;
   alpha_1 = 0;
   alpha_2 = 0;
   beta = 0;

   %home_dir = pwd();
   %work_dir = 'C:\Documents and Settings\Nick\My Documents\Nick''s\Stiller\Shape
Experiments\Opposed Cosine';
   %cd(work_dir);

   %image = imread('image_3.tiff');
   %
   % try
   % image = rgb2gray(image);
   %figure
   %imshow(image)
   % catch
   %     fprintf('Image already grayscale \n');
   % end
   image = double(image)./255;
   %
   % %--------remove this code once we get a good preprocessor---------------
   % image = round(image);  %incidentally, this completely defeats the outlining


   %imshow(image)
   %pause()
   [y_stop x_stop] = size(image);

   %Q = scale;

   if scale > (y_stop)/2 || scale > (x_stop/2)
       fprintf('Warning, largest scale greater than image allows, resetting\n');
       scale = max([y_stop,x_stop])/2;
   end


   %now we find the probabilities

   % for i=1:length(scale)
```

```
    j = 1;
    while (j < num_iter+1)
        center_y = rand()*(y_stop-1)+1; %pick our center, now a floating point
        center_x = rand()*(x_stop-1)+1; %pick our center, now a floating point
        theta = rand()*pi; %pick our angle
        x_fore = round(scale*cos(theta) + center_x); %has to become interger now I guess,
doesn't make sense otherwise
        x_back = round(center_x - scale*cos(theta));

        y_fore = round(scale*sin(theta) + center_y);
        y_back = round(center_y - scale*sin(theta));

        %check to makre sure the chosen points are valid
        if (x_fore > x_stop) || (y_fore > y_stop) || (x_back < 1) || (y_back < 1)
            %ignore this step and repeat it:
            j = j-1;
        elseif (x_back > x_stop) || (y_back > y_stop) || (x_fore < 1) || (y_fore < 1)
            %ignore this step and repeat it:
            j = j-1;
        else
            point_1 = image(y_fore,x_fore);
            point_2 = image(y_back,x_back);

            if point_1 == 0 && point_2 == 0
                alpha_1 = alpha_1+1;
            elseif point_1 == 1 && point_2 == 1
                alpha_2 = alpha_2+1;
            elseif (point_1 == 1 && point_2 == 0) || (point_2 == 1 && point_1 == 0)
                beta = beta+1;
            else
                %ignore this step because we picked up the gray area
                j = j-1;
            end
        end

        j = j+1;
    end
    %end
```

Faces Read and Write Utility Functions

```
function varargout = Faces_Read(varargin)

%-------------------------------------------------------------------------
%Faces Analysis Program File Read Function
%Nicholas Hansford 11-10-2008
%           Inputs  (Listed in order)
%           filePath   Path to output the file in
%           fileName   Name of output file
%           Ouputs
%           data       data table to read from file
%           imGlobal   image path (global reference on creation computer)
%           success    0 for sucess, 1 for failure
%-------------------------------------------------------------------------
%
%
%     Function Protoype:
%     [data imGlobal success] = Faces_Read(filePath, fileName)

% initialize success:
success = 0; %assume it worked, change for failure
data = -1;
imGlobal = [];
if nargin ~= 2
    fprintf('Error: invalid function call\n');
    success = 1;
    varargout{1} = data;
    varargout{2} = imGlobal;
    varargout{3} = success;
    return;
else
    filePath = varargin{1};
    fileName = varargin{2};
end
fileGlobal = [filePath,'/',fileName];

file = fopen(fileGlobal,'r');
if file < 0
    fprintf('Error opening file\n');
    success = 1;
    varargout{1} = data;
    varargout{2} = imGlobal;
    varargout{3} = success;
    return;
end
trash = fscanf(file,'%s',[1,3]);
num_contrast = str2double(fscanf(file,'%s',[1,1]));
trash = fscanf(file,'%s',[1,3]);
num_scale = str2double(fscanf(file,'%s',[1,1]));

data = zeros(num_contrast,num_scale);
for i = 1:num_contrast
```

```
    for j = 1:num_scale
        data(i,j) = str2double(fscanf(file,'%s',[1,1]));
    end
end

trash = fscanf(file,'%s',[1,2]);
imGlobal = fscanf(file,'%c');
imGlobal(1:2) = [];%had some file trash in 1 and 2
fclose(file);
varargout{1} = data;
varargout{2} = imGlobal; %strange extra character at end??? (sometimes keep an eye on this)
varargout{3} = success;
```

```
function success = Faces_Write(varargin)

%-------------------------------------------------------------------------
%Faces Analysis Program File Write Function
%Nicholas Hansford 06-06-2009
%           Inputs  (Listed in order)
%           data       data table to write to file
%           imPath     image path (global reference on creation computer)
%           imName     image name on creation computer
%           filePath   Path to output the file in
%           fileName   Name of output file
%           Ouput
%           success    0 for sucess, 1 for failure
%-------------------------------------------------------------------------
%
%
%     Function Protoype:
%     success = Faces_Write(data, imPath, imName, filePath, fileName)


% initialize success:
success = 0; %assume it worked, change for failure

if nargin ~= 5
    fprintf('Error: data call to function wrong\n');
    success = 1;

    %terminate
    return;
else

    % parse out the inputs
    data = varargin{1};
    imPath = varargin{2};
    imName = varargin{3};
    filePath = varargin{4};
```

```
    fileName = varargin{5};
end

[num_con,num_scale]=size(data);

%gather the file information together
fileGlobal = [filePath,'/',fileName];
file = fopen(fileGlobal,'w');
if file < 0
    fprintf('Invalide file name or error opening file\n');
    success = 1;
    return;
end

fprintf(file,'Number Contrast = %d\n',num_con);
fprintf(file,'Number Scale = %d\n', num_scale);

for i = 1:num_con
    for j = 1:num_scale
        fprintf(file,'%f\t', data(i,j));
    end
    fprintf(file,'\n');
end

imGlobal = [imPath,'/',imName];
fprintf(file,'Path = \n');
fprintf(file,'%s',imGlobal);
fclose(file);
```

```
function [file_list, num_file] = linux_list(Directory)

%the input is a Directory contianing the files to be listed in the same
%format that a Windows list produces.  The outputs at the list of files and
%the number of files in the directory
%Syntax:
%    [file_list, num_file] = linux_list(Directory)



file_list = ls(Directory, '-m');

%let's test this to get ride of that damned space:
file_list = file_list(1:length(file_list)-1);

%initializing this messes with strings later:
%file_list_2 = [];
flag_back = 1;
flag_forward = 1;
num_file_2 = 0;
for i = 1:length(file_list)
    if file_list(i) == ','
        num_file_2 = num_file_2 + 1;
        flag_forward = i;
        file_list_2(num_file_2,1:(flag_forward-flag_back)) =
file_list(flag_back:flag_forward-1);
        flag_back = flag_forward+2;
    end
end
%now we need to catch the end of the string
%file_list(flag_back:length(file_list))
num_file_2 = num_file_2 + 1;
file_list_2(num_file_2,1:(length(file_list)-flag_back+1)) =...
    file_list(flag_back:length(file_list));

%there now we finally got our shit straight for linux, grrr
[num_file, max_length] = size(file_list_2);

%test to make sure the prase didn't screw something up:
if (num_file ~= num_file_2)
    %opps
    fprintf('ERROR, file parse mistake\n');
end

%dump the old file_list
clear file_list

%create the correct one
file_list = file_list_2;
```

```
function p = make_fit(X, norm_det, order, num_contrast)
%this function will be used during experiment 4 to clean up the body of
%code and allow the curve fitting order to be in a loop.  Here a library of
%pre-defined conditions will exist.  Order must be between 5 and 9 for this
%work and we'll continue to ignore the constant term since it always near
%one.

p = zeros(order,num_contrast);

if order == 5
    for i = 1:num_contrast
        Y = norm_det(i,:);
        [curve, goodness] = fit(X',Y','poly5');
        p(1,i) = curve.p1;
        p(2,i) = curve.p2;
        p(3,i) = curve.p3;
        p(4,i) = curve.p4;
        p(5,i) = curve.p5;
    end
elseif order == 6
    for i = 1:num_contrast
        Y = norm_det(i,:);
        [curve, goodness] = fit(X',Y','poly6');
        p(1,i) = curve.p1;
        p(2,i) = curve.p2;
        p(3,i) = curve.p3;
        p(4,i) = curve.p4;
        p(5,i) = curve.p5;
        p(6,i) = curve.p6;
    end
elseif order == 7
    for i = 1:num_contrast
        Y = norm_det(i,:);
        [curve, goodness] = fit(X',Y','poly7');
        p(1,i) = curve.p1;
        p(2,i) = curve.p2;
        p(3,i) = curve.p3;
        p(4,i) = curve.p4;
        p(5,i) = curve.p5;
        p(6,i) = curve.p6;
        p(7,i) = curve.p7;
    end
elseif order == 8
    for i = 1:num_contrast
        Y = norm_det(i,:);
        [curve, goodness] = fit(X',Y','poly8');
        p(1,i) = curve.p1;
        p(2,i) = curve.p2;
        p(3,i) = curve.p3;
        p(4,i) = curve.p4;
        p(5,i) = curve.p5;
        p(6,i) = curve.p6;
        p(7,i) = curve.p7;
```

```
        p(8,i) = curve.p8;
    end
elseif order == 9
    for i = 1:num_contrast
        Y = norm_det(i,:);
        [curve, goodness] = fit(X',Y','poly9');
        p(1,i) = curve.p1;
        p(2,i) = curve.p2;
        p(3,i) = curve.p3;
        p(4,i) = curve.p4;
        p(5,i) = curve.p5;
        p(6,i) = curve.p6;
        p(7,i) = curve.p7;
        p(8,i) = curve.p8;
        p(9,i) = curve.p9;
    end

end
```

Appendix D: GUI Source Codes

Note: GUI support functions are not replicated and are available in Appendix C.

Main Menu Window

```
function varargout = Main_Menu(varargin)
% MAIN MENU M-file for Main Menu.fig
%      MAIN_MENU, by itself, creates a new MAIN_MENU or raises the existing
%      singleton*.
%
%      H = MAIN_MENU returns the handle to a new MAIN_MENU or the handle to
%      the existing singleton*.
%
%      MAIN_MENU('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MAIN_MENU.M with the given input arguments.
%
%      MAIN_MENU('Property','Value',...) creates a new MAIN_MENU or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Main_Menu_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Main_Menu_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Main_Menu

% Last Modified by GUIDE v2.5 15-Nov-2008 12:58:05

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Main_Menu_OpeningFcn, ...
                   'gui_OutputFcn',  @Main_Menu_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Main_Menu is made visible.
function Main_Menu_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Main_Menu (see VARARGIN)

% Choose default command line output for Main_Menu
handles.output = hObject;

clc;
%set up the initial axes

%----------Variable Stucture-------------------------------------------------%
%    base_image                    original image                             %
%    base_image.path               global path for real image                 %
%                                  local path for start up defaults           %
%   base_image.name                name of image in .path                     %
%   base_image.data                RGB or grayscale values of base image      %
%   image_use.data                 RGB values of the cropped image            %
%   image_use.distance             1/2 the distance between the eyers         %
%   data.alpha_1                   alpha1 counter                             %
%   data.alpha_2                   alpha2 counter                             %
%   data.beta                      beta counter                               %
%   data.scale                     scale matrix                               %
%   data.norm_det                  normalized determinant                     %

handles.base_image.path = './Images/';
handles.base_image.name = 'default_image.jpg';
handles.base_image.data = imread([handles.base_image.path...
    handles.base_image.name]);

axes(handles.axes1);
imshow(handles.base_image.data);

axes(handles.axes2);
imshow(handles.base_image.data);

handles.distance = 0;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Main_Menu wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Main_Menu_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

[filename,pathname,index]=uigetfile({'*.jpg;*.tif;*.png;*.gif','All Image Files';'*.*',...
    'All Files' },'Please Select a File','MultiSelect','Off');

if index == 0
    set(handles.edit1,'string','Error, try again');
else
    location = [pathname, filename];
    try image = imread(location);
        axes(handles.axes1)
        image = imread(location);
        imshow(image); %update the figure
        %handles.valid_image = 0;
        set(handles.edit1,'string',filename);
    catch
        %error opening the image
        axes(handles.axes1)
        pathname = 'Images/';
        filename = 'error_image.jpg';
        location = [pathname, filename];
        image = imread(location);
        imshow(image);
        handles.valid_image = 1;
    end

    handles.base_image.path = pathname;
    handles.base_image.name = filename;
```

```
    handles.base_image.data = image;
    % update the GUI data handle
    guidata(hObject, handles);
end


guidata(hObject, handles);



% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

axes(handles.axes1);
zoom;


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

axes(handles.axes1);
pan;




% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[handles.image_use.data, handles.image_use.distance] = ...
    Pre_Process(handles.base_image.data);

%display(handles.distance)
%upon assumed successful completion, turn on the save image button
set(handles.pushbutton6,'Enable','on');
set(handles.pushbutton7,'Enable','on');

axes(handles.axes2)
imshow(handles.image_use.data);
guidata(hObject, handles);




% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

try handles.image_use(1,1);
    %ok, it actually exists:
    [filename,pathname]=uiputfile({'*.tiff','TIFF Image File'});
    filename = [pathname,filename]; %,'.tiff']

    try
        %imshow (handles.image_use
        imwrite(handles.image_use.data, filename);
        %disp(handles.image_use.data);
    catch
        fprintf('Error occured during image write, try again\n');
    end
catch
    fprintf('Error, please pre-process an image first.\n');
end




% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%and call the Process menu, I don't know what the outputs will be just yet:
[alpha_1, alpha_2, beta, scale, norm_det]=...
    Process(handles.image_use.data, handles.image_use.distance,...
    handles.base_image);
handles.data.alpha_1 = alpha_1;
handles.data.alpha_2 = alpha_2;
handles.data.beta = beta;
handles.data.scale = scale;
handles.data.norm_det = norm_det;
handles.base_image.imGlobal = [handles.base_image.path,...
    handles.base_image.name];

set(handles.pushbutton11,'Enable','on');
guidata(hObject, handles);



% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename,pathname,index]=uigetfile({'*.txt',...
    'Face Analysis Data Files';'*.*',...
    'All Files' },'Please Select a File','MultiSelect','Off');

try
```

```
%
    [data imGlobal test] = Faces_Read(pathname,filename);
    handles.data.norm_det = data;
    handles.base_image.imGlobal = imGlobal;

    set(handles.pushbutton11,'Enable','on');


catch
    fprintf('Error opening file, setting all values to NULL');
    contrast_temp = -1;
    scale_temp = -1;
    alpha_1_temp = -1;
    alpha_2_temp = -1;
    beta_temp = -1;
    norm_det = -1;
end

guidata(hObject, handles);

% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

data = handles.data.norm_det;
try
    base_image.data = imread(handles.base_image.imGlobal);
catch
    base_image.data = imread('./Images/error_image.jpg');
end
Comparison(data,base_image);
```

Pre-processing Window

```
function varargout = Pre_Process(varargin)
% PRE_PROCESS M-file for Pre_Process.fig
%      PRE_PROCESS, by itself, creates a new PRE_PROCESS or raises the existing
%      singleton*.
%
%      H = PRE_PROCESS returns the handle to a new PRE_PROCESS or the handle to
%      the existing singleton*.
%
%      PRE_PROCESS('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PRE_PROCESS.M with the given input arguments.
%
%      PRE_PROCESS('Property','Value',...) creates a new PRE_PROCESS or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Pre_Process_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Pre_Process_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Pre_Process

% Last Modified by GUIDE v2.5 24-Jun-2008 22:00:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Pre_Process_OpeningFcn, ...
                   'gui_OutputFcn',  @Pre_Process_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Pre_Process is made visible.
function Pre_Process_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Pre_Process (see VARARGIN)

% Choose default command line output for Pre_Process
handles.output = hObject;

%gather the inputs:
%varargin{1};
axes(handles.axes1);
imshow(varargin{1});

handles.image = varargin{1};
handles.distance = 0;
% disp(handles.image);

handles.propper_select = 0;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Pre_Process wait for user response (see UIRESUME)
%turn this on so it doesn't output immediately the outputs
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Pre_Process_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.image_out;
varargout{2} = handles.distance;
%this is where the outputs go:

delete(hObject);

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
zoom;

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
pan
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
datacursormode on;

% if handles.propper_select > 0
%     handles.propper_select = handles.propper_select-1;
% end %already not set:

% Update handles structure
guidata(hObject, handles);

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close;


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.image_out = handles.image;
guidata(hObject, handles);
uiresume;
% Hint: delete(hObject) closes the figure
%delete(hObject);



% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%figure_handle = get(handles.axes1)
axes(handles.axes1);
dummy = gcf;
dcm_object = datacursormode(dummy);
dcm_data = getCursorInfo(dcm_object);

%[Top_Left_X, Top_Left_Y] = dcm_data.Position
Top_Left_X = dcm_data.Position(1);
Top_Left_Y = dcm_data.Position(2);

set(handles.edit3,'String',num2str(Top_Left_X));
set(handles.edit4,'String',num2str(Top_Left_Y));

handles.Top_Left_X=Top_Left_X;
handles.Top_Left_Y=Top_Left_Y;
%handles.axes1.datacursormode
%pos = get(handles.axes1.datacursormode,'Position')
datacursormode off;

% if handles.propper_select > 0 && handles.propper_select < 2
%     handles.proper_select = handles.propper_select + 1;
```

```
% end
%and finally, update the guidata
guidata(hObject, handles);


% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
datacursormode on;
%
% if handles.propper_select >= 0
%     handles.propper_select = handles.propper_select-1;
% end %already not set:

% Update handles structure
guidata(hObject, handles);
% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
dummy = gcf;
dcm_object = datacursormode(dummy);
dcm_data = getCursorInfo(dcm_object);

%[Top_Left_X, Top_Left_Y] = dcm_data.Position
Bot_Right_X = dcm_data.Position(1);
Bot_Right_Y = dcm_data.Position(2);

set(handles.edit1,'String',num2str(Bot_Right_X));
set(handles.edit2,'String',num2str(Bot_Right_Y));

handles.Bot_Right_X=Bot_Right_X;
handles.Bot_Right_Y=Bot_Right_Y;
%handles.axes1.datacursormode
%pos = get(handles.axes1.datacursormode,'Position')
datacursormode off;

%let's see if we can User proof this a little
% if handles.propper_select >= 0 && handles.propper_select < 2
%     handles.proper_select = handles.propper_select + 1;
% end
%and finally, update the guidata
guidata(hObject, handles);




% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%preview button

handles.propper_select = 0;
warn = 0;

try X1 = handles.Top_Left_X;
    handles.propper_select = handles.propper_select+1;
catch
    warn = 1;
end

try X2 = handles.Bot_Right_X;
    handles.propper_select = handles.propper_select+1;
catch
    warn = 1;
end
if handles.propper_select == 2 %ok to open
    X1 = handles.Top_Left_X;
    X2 = handles.Bot_Right_X;
    Y1 = handles.Top_Left_Y;
    Y2 = handles.Bot_Right_Y;

    image_temp = elipse_creator(X1, X2,Y1, Y2, handles.image);
%     %axes(handles.axes2);
    Image_Preview(image_temp);
    % do what is neccessary to invoke the elipse_creator.m

end

guidata(hObject, handles);


% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%save and close button

handles.propper_select = 0;
warn = 0;

try X1 = handles.Top_Left_X;
    handles.propper_select = handles.propper_select+1;
catch
    warn = 1;
end

try X2 = handles.Bot_Right_X;
    handles.propper_select = handles.propper_select+1;
catch
```

```
    warn = 1;
end
if handles.propper_select == 2 %ok to open
    X1 = handles.Top_Left_X;
    X2 = handles.Bot_Right_X;
    Y1 = handles.Top_Left_Y;
    Y2 = handles.Bot_Right_Y;

    %calculate the distance between eyes for scale scaling...
    distance = sqrt( (X2-X1)^2 + (Y2-Y1)^2 );
    handles.distance = distance;
    image_temp = elipse_creator(X1, X2,Y1, Y2, handles.image);
    handles.image_out = image_temp;
else
    disp('Improper Data Selection')
    %I don't really know if this is necessary:
    handles.image_out = imread('./Images/error_return.jpg');
end

guidata(hObject, handles);
uiresume;
```

Pre-processing Window Support Functions
(ellipse creator and image preview window)

```
function image_use_small_save = elipse_creator(X1,X2,Y1,Y2,image)

% function to create an elipse in an image file at the using the x,y pairs
% corrisponding to the center of the two eyes.

% start by copying the input image to the useput image:

%first check to make sure that iamge is RGB, if not, convert back

%-----used during beta-----
% clear all;
% close all;
% clc;
% image = imread('Images/image0019.tif');
% X1 = 178;
% Y1 = 360;
% X2 = 305;
% Y2 = 362;

Q = length(size(image));

dimensions = size(image);

image_use = uint8(255*ones(dimensions(1),dimensions(2),3));

if Q == 3 %rgb image, nothing is neccessary
    image_use = image;
else %assume it is actually an image file...
    image_use(:,:,1) = image;
    image_use(:,:,2) = image;
    image_use(:,:,3) = image;
end

%imshow(image_use)
%ok, now to place the line
%slope
m = (Y2 - Y1) / (X2 - X1);

%and the intercept:
intcpt = -m*X1 + Y1;


%now lets get the line in the image

% for i = X1:X2
%     x = i;
%     y = round(m*x + intcpt);
%
%     image_use(y,x,1) = 0;
%     image_use(y,x,2) = 255;
%     image_use(y,x,3) = 0;
```

```
% end

%and find the half_length of the line

pupil_bridge = 0.5*sqrt( (Y2 - Y1)^2 + (X2 - X1)^2 );

%find the center of the ellipse (h,k)
theta = asin( ( Y2 - Y1) / pupil_bridge);

delta = pupil_bridge * cos(theta);
nu = pupil_bridge * sin(theta);

%-----Place the Eye as the center and work from that-----%
% h = round(X2 - nu);
% k = round(Y2 + delta);
h = round(X2);
k = round(Y2);

% image_use(k,h,1) = 0;
% image_use(k,h,2) = 255;
% image_use(k,h,3) = 0;

%gather all the X-based X,Y pairs

%gather and sort all integers including X2 and h

% figure
% imshow(image_use)

%semi_Major
semi_Major = 1.5 * pupil_bridge;
%semi_Minor take from using the bridge-eye distance as the semi-latus rectum
%semi_Minor = sqrt( 2*pupil_bridge^2);
semi_Minor = pupil_bridge;
%mark a box around the ellipse area
%I'll use a rotation matrix later to rotate the ellipse by theta, lets just
%get the thing in there first!

a = semi_Major;
b = semi_Minor;
%------First Quadrant-------%
%note that these values have not been adjusted for the rotation yet, I
%intend to that last
x = h:1:round(h+b);
y_up = (k + sqrt(a^2*(1 - (x - h).^2 / b^2)));

y = k:1:round(k+a);
x_up = (h + sqrt(b^2*(1 - (y - k).^2 / a^2)));

x_use_first_quad = sort([x x_up]);
y_use_first_quad = sort([y y_up], 'descend');

%------Second Quadrant-------%
%note that these values have not been adjusted for the rotation yet, I
%intend to that last
```

```
y_use_sec_quad = y_use_first_quad;
x_use_sec_quad = 2*h - x_use_first_quad;

%------Third Quadrant-------%
%note that these values have not been adjusted for the rotation yet, I
%intend to that last
y_use_third_quad = 2*k - y_use_first_quad;
x_use_third_quad = 2*h - x_use_first_quad;

%------Fourth Quadrant-------%
%note that these values have not been adjusted for the rotation yet, I
%intend to that last
y_use_fourth_quad = 2*k - y_use_first_quad;
x_use_fourth_quad = x_use_first_quad;

%just as a proof of concept for the ellipse
% figure
% plot(x_use_sec_quad,y_use_sec_quad,x_use_first_quad,y_use_first_quad,...
%     x_use_third_quad,y_use_third_quad,x_use_fourth_quad,y_use_fourth_quad)

%------now put it all back togeter-------%
clear x y;
x = [x_use_first_quad x_use_sec_quad x_use_third_quad x_use_fourth_quad];
y = [y_use_first_quad y_use_sec_quad y_use_third_quad y_use_fourth_quad];


%Perform the rotations here...
%ok first move the center of the elipse to the origin or rotation
x = x - h;
y = y - k;

%and rotate that stuff:
x_use_temp = cos(theta)*x - sin(theta)*y;
y_use_temp = sin(theta)*x + cos(theta)*y;

x = x_use_temp+ h;
y = y_use_temp + k;

x_int = round(real(x));
y_int = round(real(y));

x_mod_up = []; %might be used to add to the x arrary
y_mod_up = []; %might be used to add to the y array
x_mod_dn = [];
y_mod_dn = [];
num_mod = 0; %number of additional pixels to be masked out

%-----Check to make sure it is continuous-----%
for i = 1:length(y_int)-1
    %see which one changed:
    diff = abs(y_int(i) -  y_int(i+1));
    if diff > 1 && diff < 10
        %y grew by more than, meaning it opened up a hole
        %fill in that number of ys
        for j = 1:diff
```

```
            num_mod = num_mod + 1;
            %average the x pixels together neeed here
            x_mod_up(num_mod) = (x_int(i) + x_int(i+1)) / 2;
            y_mod_up(num_mod) = y_int(i) + j;

            x_mod_dn(num_mod) = (x_int(i) + x_int(i+1)) / 2;
            y_mod_dn(num_mod) = y_int(i) - j;
        end
    end
end

x = [x x_mod_up x_mod_dn]; %add in the new pixels
y = [y y_mod_up y_mod_dn];

x_int = round(real(x));
y_int = round(real(y));
%-----mark the ellipse in the image for beta testing-----%
for i = 1:length(x) % they're all the same anyway

    %let's make some color
    image_use(y_int(i),x_int(i),1) = 0;
    image_use(y_int(i),x_int(i),2) = 255;
    image_use(y_int(i),x_int(i),3) = 0;

end


% figure
% imshow(image_use)

%and now strip the ellipse out of the image:

X_min = round(min(x));% - 10; %just to be safe:
X_max = round(max(x));% + 10;
Y_min = round(min(y));% - 10;
Y_max = round(max(y));% + 10;

image_use_small = image_use(Y_min:Y_max, X_min:X_max,:);
image_use_small_save = image_use(Y_min:Y_max, X_min:X_max,:);

%-----March around image and make sure we have a complete ellipse-----%

% y_start = round(y(1));
% x_start = round(x(1));
%
% is_green = 0; %assume the pixel is green
% stop = 0; %finishes when all pixels have been checked:
%
% while stop == 0


%----Mask out what I don't want to use:-----%

%left side
stop = 0;
```

```
small_dimensions = size(image_use_small);
for i = 1:small_dimensions(1) %march over all Ys

    stop = 0;
    j = 1;
    while stop == 0
        if (image_use_small(i,j,1) == 0 && image_use_small(i,j,3) == 0 ...
                && image_use_small(i,j,2) == 255) %found a grean pixel
            stop = 1; %break the loop
        else
            %mask the pixel
            image_use_small_save(i,j,1) = 0;
            image_use_small_save(i,j,2) = 255;
            image_use_small_save(i,j,3) = 0;
            j = j + 1;
        end
    end
end

%right side
stop = 0;
small_dimensions = size(image_use_small);
for i = 1:small_dimensions(1) %march over all Ys

    stop = 0;
    j = small_dimensions(2);
    while stop == 0
        if (image_use_small(i,j,1) == 0 && image_use_small(i,j,3) == 0 ...
                && image_use_small(i,j,2) == 255) %found a grean pixel
            stop = 1; %break the loop
        else
            %mask the pixel
            image_use_small_save(i,j,1) = 0;
            image_use_small_save(i,j,2) = 255;
            image_use_small_save(i,j,3) = 0;
            j = j - 1;
        end
    end
end
```

```
function varargout = Image_Preview(varargin)
% IMAGE_PREVIEW M-file for Image_Preview.fig
%      IMAGE_PREVIEW, by itself, creates a new IMAGE_PREVIEW or raises the existing
%      singleton*.
%
%      H = IMAGE_PREVIEW returns the handle to a new IMAGE_PREVIEW or the handle to
%      the existing singleton*.
%
%      IMAGE_PREVIEW('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in IMAGE_PREVIEW.M with the given input arguments.
%
```

```
%      IMAGE_PREVIEW('Property','Value',...) creates a new IMAGE_PREVIEW or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Image_Preview_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Image_Preview_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Image_Preview

% Last Modified by GUIDE v2.5 24-Jun-2008 21:50:22

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Image_Preview_OpeningFcn, ...
                   'gui_OutputFcn',  @Image_Preview_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Image_Preview is made visible.
function Image_Preview_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Image_Preview (see VARARGIN)
% Choose default command line output for Pre_Process
handles.output = hObject;

%gather the inputs:
varargin{1};
axes(handles.axes1);
imshow(varargin{1});

handles.image = varargin{1};
% disp(handles.image);

handles.propper_select = 0;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Image_Preview wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Image_Preview_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close
```

Processing Window

```
function varargout = Process(varargin)
% PROCESS M-file for Process.fig
%      PROCESS, by itself, creates a new PROCESS or raises the existing
%      singleton*.
%
%      H = PROCESS returns the handle to a new PROCESS or the handle to
%      the existing singleton*.
%
%      PROCESS('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PROCESS.M with the given input arguments.
%
%      PROCESS('Property','Value',...) creates a new PROCESS or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Process_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Process_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Process

% Last Modified by GUIDE v2.5 13-Sep-2008 11:12:18

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Process_OpeningFcn, ...
                   'gui_OutputFcn',  @Process_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Process is made visible.
function Process_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Process (see VARARGIN)

% Choose default command line output for Process
handles.output = hObject;

%gather the inputs:
varargin{1};
axes(handles.axes1);
imshow(varargin{1});

handles.alpha_1 = -1;
handles.alpha_2 = -1;
handles.beta = -1;
handles.scale = -1;
handles.norm_det = -1;

handles.image = varargin{1};
handles.distance = varargin{2};
handles.base_image = varargin{3};

%disp(handles.base_image)
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Process wait for user response (see UIRESUME)
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Process_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
%varargout{1} = handles.output;
varargout{1} = handles.alpha_1;
varargout{2} = handles.alpha_2;
varargout{3} = handles.beta;
varargout{4} = handles.scale;
varargout{5} = handles.norm_det;

delete(hObject);


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
uiresume;
```

```
%delete(hObject);



% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%set up the "progress bar"

%for now, let's just define a series of contrasts, we may have to relate
%these to image properties later:

%-----Predefined contrast levels and scale series-----%
% Note: scale may change if it over steps the maximum values tolerated by
% image so a second array will be kept to track these changes
% contrast = [50 60 70 80 90 100 110 120 127.5 135 145 155 165 175 185 195 205];
%let's see if I can come up with a better sceme.


%calculate the average
[Y_stop, X_stop,RGB]=size(handles.image);
num_pixels = 0;
pixel_total = 0;
for i =1:Y_stop
    for j = 1:X_stop
        if handles.image(i,j,2) == 255 && handles.image(i,j,1) == 0 && ...
                handles.image(i,j,3) == 0
            %don't want to do anything this is a green pixel
        else
            num_pixels = num_pixels + 1;
            pixel_total = pixel_total + double(rgb2gray(handles.image(i,j,:)));
        end
    end
end

average = pixel_total / num_pixels;

%calulate the standard deviation
stand_dev = 0;
for i =1:Y_stop
    for j = 1:X_stop
        if handles.image(i,j,2) == 255 && handles.image(i,j,1) == 0 && ...
                handles.image(i,j,3) == 0
            %don't want to do anything this is a green pixel
        else
            stand_dev = stand_dev+(double(rgb2gray(handles.image(i,j,:)))...
                -average))^2;
        end
    end
end
```

```
stand_dev = sqrt(1/num_pixels*stand_dev);


%numbers = [-2.0 -1.6 -1.2 -0.8 -0.4 0 0.4 0.8 1.2 1.6 2.0];

%now 10 numbers from experiment 1
numbers = linspace(-2.0,2.0,10);
for i = 1:length(numbers)
    contrast(i) = average + numbers(i)*stand_dev;
end


for i = 1:length(contrast)
    if contrast(i) < 0
        contrast(i) = 0;
    elseif contrast(i) > 255
        contrast(i) = 255;
    end
end
%ok, need to non-dimen this:

%display(handles.distance);

%50 scales from experiment 1
scale = linspace(0,(handles.distance/4),50);

%display(scale)

num_contrast = length(contrast);
num_scale = length(scale);
alpha_1 = zeros(num_contrast, num_scale);
alpha_2 = alpha_1;
beta = alpha_1;
scale_actual = alpha_1;
norm_det = alpha_1;
progress_percent = 0;

for i = 1:num_contrast

    %open level out the image:
    image_temp = Contrast_Level(handles.image, contrast(i));
    for j = 1:num_scale

        %invoke the solver:
        [alpha_1(i,j), alpha_2(i,j), beta(i,j), scale_actual(i,j)] = ...
            outlined_RW(image_temp, scale(j));

        %find the normalized determinant
        norm_det(i,j) = (alpha_1(i,j)*alpha_2(i,j) - 0.25*beta(i,j)^2) / ...
            (alpha_1(i,1)*alpha_2(i,1));

        %update the progress bar:
        progress_percent = ((i-1)*num_scale + j)/(num_contrast*num_scale);
        set(Progress_Update(progress_percent, handles),...
            'BackgroundColor',[23/255 1/255 126/255]);
```

```
        drawnow;

        %debuggin purposes only:
        % fprintf('Contrast = %5.2f and Scale = %5.2f\n', contrast(i), scale(j));
    end
end

%and finally create the plot:
axes(handles.axes3)

[X,Y] = meshgrid(contrast , scale); %didn't pick up the actual scale, will do that later

% disp(size(X))
% disp(size(Y))
% disp(size(norm_det))
surf(X,Y,norm_det');
xlabel('Contrast Level (Pixels)');
ylabel('Scale (Pixels)');

% Used during debug
% save 'test.mat' X Y norm_det
view([90 90 90]);

handles.alpha_1 = alpha_1;
handles.alpha_2 = alpha_2;
handles.beta = beta;
handles.scale = scale_actual;
%handles.norm_det = norm_det; not needed anymore, going to use slopes
%instead
handles.contrast = contrast;

%update this block of code to create the slopes...and the slopes need to go
%back out of the program.

%link to datafile containing the average slopes is hardwired in!
avg_norm_det = load('./Images/avg_norm_det.csv');

%----taken directly from exp 3 source codes----%

%determine the sloped data for normalized determinant:
s_con = zeros(size(norm_det));
s_scale = zeros(size(norm_det));

%create "fake" boundary conditions so that conditionals aren't
%necessary

norm_use = zeros(num_contrast+2, num_scale+2);
norm_use(2:end-1, 2:end-1) = norm_det;


%and duplicate the boundary condition so
%left hand side
norm_use(2:end-1,1) = norm_det(:,1);

%right hand side
```

```
norm_use(2:end-1,end) = norm_det(:,end);

%top
norm_use(1,2:end-1) = norm_det(1,:);

%bottom
norm_use(end,2:end-1) = norm_det(end,:);

%denominators are constant due to setup
%it doesn't matter what these are.  We're only concerned with the sign
%and these are inherently always positive.

temp = 0.;

for i = 2:num_contrast+1
    for j = 2:num_scale+1
        temp = (norm_use(i+1,j)-norm_use(i-1,j));
        if temp > 0
            s_con(i-1,j-1) = 1;
        else
            s_con(i-1,j-1) = 0;
        end

        temp = (norm_use(i,j+1) - norm_use(i,j-1));
        if temp > 0
            s_scale(i-1,j-1) = 1;
        else
            s_scale(i-1,j-1) = 0;
        end
    end
end

%concatenate into one variable because we need both for comparisons
%now, the actual version of the norm_det we wish to send out is this data
handles.norm_det = [s_con; s_scale];

set(handles.pushbutton3,'Enable', 'on');

%set(handles.pushbutton4,'Enable', 'on');
guidata(hObject, handles);

%-----Used for debugging only-----%
%save Data.mat scale_actual norm_det

%disp(handles)


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close;
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% [filename,pathname]=uiputfile({'*csv','Analysis Data File'});
% filename = [pathname,filename,'.csv'];
%

[filename,pathname]=uiputfile({'*txt','Face Analysis Data File'});

%check to make sure extension isn't already there:

Q = length(filename);
if (strcmp(filename(Q-3:Q) , '.txt') == 1) %already there
    filename = [filename];
else
    filename = [filename,'.txt'];
end

test = Faces_Write(handles.norm_det,handles.base_image.path,...
    handles.base_image.name,pathname,filename);
```

Progress Bar Update Function

```
function output = Progress_Update(input, handles)

% input ranges between 0 and 1

choice = round(input * 100);

if choice < 100 / 26
    output = handles.text5;
elseif choice < 100 / 26 * 2
    output = handles.text6;
elseif choice < 100 / 26 * 3
    output = handles.text7;
elseif choice < 100 / 26 * 4
    output = handles.text8;
elseif choice < 100 / 26 * 5
    output = handles.text9;
elseif choice < 100 / 26 * 6
    output = handles.text10;
elseif choice < 100 / 26 * 7
    output = handles.text11;
elseif choice < 100 / 26 * 8
    output = handles.text12;
elseif choice < 100 / 26 * 9
    output = handles.text13;
elseif choice < 100 / 26 * 10
    output = handles.text14;
elseif choice < 100 / 26 * 11
    output = handles.text15;
elseif choice < 100 / 26 * 12
    output = handles.text16;
elseif choice < 100 / 26 * 13
    output = handles.text17;
elseif choice < 100 / 26 * 14
    output = handles.text18;
elseif choice < 100 / 26 * 15
    output = handles.text19;
elseif choice < 100 / 26 * 16
    output = handles.text20;
elseif choice < 100 / 26 * 17
    output = handles.text21;
elseif choice < 100 / 26 * 18
    output = handles.text22;
elseif choice < 100 / 26 * 19
    output = handles.text23;
elseif choice < 100 / 26 * 20
    output = handles.text24;
elseif choice < 100 / 26 * 21
    output = handles.text25;
elseif choice < 100 / 26 * 22
    output = handles.text26;
elseif choice < 100 / 26 * 23
```

```
    output = handles.text27;
elseif choice < 100 / 26 * 24
    output = handles.text28;
elseif choice < 100 / 26 * 25
    output = handles.text29;
else
    output = handles.text30;
end
```

Comparison Toolbox Window

```
function varargout = Comparison(varargin)
% COMPARISON M-file for Comparison.fig
%      COMPARISON, by itself, creates a new COMPARISON or raises the existing
%      singleton*.
%
%      H = COMPARISON returns the handle to a new COMPARISON or the handle to
%      the existing singleton*.
%
%      COMPARISON('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in COMPARISON.M with the given input arguments.
%
%      COMPARISON('Property','Value',...) creates a new COMPARISON or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Comparison_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Comparison_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Comparison

% Last Modified by GUIDE v2.5 10-Nov-2008 22:28:50

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Comparison_OpeningFcn, ...
                   'gui_OutputFcn',  @Comparison_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Comparison is made visible.
function Comparison_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figureCopy_of_Version 2.0
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Comparison (see VARARGIN)

% Choose default command line output for Comparison
handles.output = hObject;

handles.good_data = varargin{1};
handles.base_image = varargin{2}; %to pop up a window for visual que
%disp(handles.good_data)

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Comparison wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Comparison_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
Directory = uigetdir('./','Please Select a Database Directory');

%update the display
Q = length(Directory);
if (Q > 60)
    set(handles.edit1,'String',strcat('...',Directory(Q - 60:Q)));
else
    set(handles.edit1,'String',Directory);
end

%is this a linux or windows
os_type = isunix; %returns 1 for linux, 0 for other (assumed as windows)
%get the file list:
if (os_type == 1)
    file_list = ls(Directory, '-m');

    %let's test this to get ride of that damned space:
    file_list = file_list(1:length(file_list)-1);

    %initializing this messes with strings later:
    %file_list_2 = [];
    flag_back = 1;
    flag_forward = 1;
    num_file_2 = 0;
    for i = 1:length(file_list)
        if file_list(i) == ','
            num_file_2 = num_file_2 + 1;
            flag_forward = i;
            file_list_2(num_file_2,1:(flag_forward-flag_back)) =
file_list(flag_back:flag_forward-1);
            flag_back = flag_forward+2;
        end
    end
    %now we need to catch the end of the string
    %file_list(flag_back:length(file_list))
    num_file_2 = num_file_2 + 1;
    file_list_2(num_file_2,1:(length(file_list)-flag_back+1)) =...
        file_list(flag_back:length(file_list));

    %there now we finally got our shit straight for linux, grrr
    [num_file, max_length] = size(file_list_2);

    %test to make sure the prase didn't screw something up:
    if (num_file ~= num_file_2)
        %opps
        fprintf('ERROR, file parse mistake\n');
    end

    %dump the old file_list
    clear file_list

    %create the correct one
    file_list = file_list_2;
```

```
    %clear the temporary one
    clear file_list_2


else
    %let's get the list
    file_list = ls(Directory);
    %note that matlab in linux doesn't do this:
    %and remove the directory locate
    file_list(1:2,:) = [];
    [num_file, max_length] = size(file_list);
end

%and check for only .txt files:
num_file_2 = 0;
for i = 1:num_file
    for j = 1:max_length-3
        if (strcmp(file_list(i,j:j+3),'.txt') == 1)  %bad file:
            num_file_2 = num_file_2 + 1;
            file_list_2(num_file_2,:) = file_list(i,:);
            break
        end
    end
end

num_file = num_file_2;
file_list = file_list_2;

%add extra entry to file list
%file_list(num_file+1,:) = file_list(num_file,:);

% disp(file_list)
% disp(num_file)
% disp(file_list)
set(handles.listbox1,'String',file_list);
%ok now that we've idiot proofed it a little:

set(handles.pushbutton2,'Enable','on');

%and update the handles
handles.file_list = file_list;
handles.num_file = num_file;
handles.directory = Directory;
% Update handles structure
guidata(hObject, handles);

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listbox1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox1
```

```
% selection = get(handles.listbox1,'Value');
%
% set(handles.edit2,'String',mat2str(handles.result_error(selection)));


% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

num_file = handles.num_file;
file_list = handles.file_list;
Directory = handles.directory;
good_data = handles.good_data;

dummy1 = get(handles.edit3,'String');
tolerance = str2double(dummy1)/100;

% dummy2 = get(handles.edit4,'String');
% counter_goal = str2double(dummy2)/100;

%get the first one to determine size:
test_data = Faces_Read(Directory,file_list(1,:));
[num_contrast num_scale] = size(test_data);
data = zeros(num_file,num_contrast,num_scale);
handles.imGlobal = cell(1,num_file); %setup to store all of the image names
diff = 0;

% for i = 1:num_file
%     [data(i,:,:),dummy] = Faces_Read(Directory,file_list(i,:))
%     handles.imGlobal(i)={dummy};
% end

counter = zeros(1,num_file);


iter = 1;
tol = tolerance*num_contrast*num_scale;
```

```
for k = 1:num_file %march over all the files

    [data,dummy] = Faces_Read(Directory,file_list(k,:));
    handles.imGlobal(k)={dummy};

    %matlab allows us to test equality (boolean operations) through
    %entire arrays, so we just select the data which corresponds to
    %the test or database image.  If the match is true (1 to 1 or 0
    %to 0) it returns 1, 0 otherwise.  So then we just add up all
    %the ones (twice because of the matrix structure).
    matches = sum(sum(good_data == data));

    %is that enough of the total
    if matches >= tol
        %success!
        file_list_2(iter,:) = file_list(k,1:15);
        handles.imGlobal_short(iter) = handles.imGlobal(k);

        %increment iter
        iter = iter + 1;
    end
end


if iter == 1
    file_list_2  = '';
    handles.imGlobal_short = cell(1,1);
end

%file_list_2 = file_list(rank_diff);
%disp(file_list_2)
%turn on the selection box:
set(handles.listbox1,'Enable','on');

set(handles.listbox2,'String',file_list_2);

%turn on the selection box:
set(handles.listbox2,'Enable','on');

%turn on the preview box:
set(handles.pushbutton3,'Enable','on');
%save the errors:
% handles.result_error = diff;
% handles.rank_diff = rank_diff;
% Update handles structure
guidata(hObject, handles);




function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes on selection change in listbox2.
function listbox2_Callback(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listbox2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox2
% selection = get(handles.listbox2,'Value');
%
% set(handles.edit2,'String',mat2str(handles.result_error(...
%     handles.rank_diff(selection))));

% --- Executes during object creation, after setting all properties.
function listbox2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
```

```
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton1



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
im_num = get(handles.listbox2,'Value');
image_dir = cell2mat(handles.imGlobal_short(im_num));

try
    image = imread(image_dir);
catch
    image = imread('./Images/error_image.jpg');
end
Image_Preview_2(handles.base_image.data,image);
```

Image Preview Window

```
function varargout = Image_Preview_2(varargin)
% IMAGE_PREVIEW_2 M-file for Image_Preview_2.fig
%      IMAGE_PREVIEW_2, by itself, creates a new IMAGE_PREVIEW_2 or raises the existing
%      singleton*.
%
%      H = IMAGE_PREVIEW_2 returns the handle to a new IMAGE_PREVIEW_2 or the handle to
%      the existing singleton*.
%
%      IMAGE_PREVIEW_2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in IMAGE_PREVIEW_2.M with the given input arguments.
%
%      IMAGE_PREVIEW_2('Property','Value',...) creates a new IMAGE_PREVIEW_2 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Image_Preview_2_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Image_Preview_2_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Image_Preview_2

% Last Modified by GUIDE v2.5 10-Nov-2008 23:15:14

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Image_Preview_2_OpeningFcn, ...
                   'gui_OutputFcn',  @Image_Preview_2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Image_Preview_2 is made visible.
function Image_Preview_2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Image_Preview_2 (see VARARGIN)

% Choose default command line output for Image_Preview_2
handles.output = hObject;

axes(handles.axes2);
imshow(varargin{2});

axes(handles.axes1);
imshow(varargin{1});

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Image_Preview_2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Image_Preview_2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close;
```