DEVELOPMENT AND EVALUATION OF A DIGITAL TOOL FOR VIRTUAL RECONSTRUCTION OF HISTORIC ISLAMIC GEOMETRIC PATTERNS

A Dissertation

by

RIMA AHMAD AL AJLOUNI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2005

Major Subject: Architecture

DEVELOPMENT AND EVALUATION OF A DIGITAL TOOL FOR VIRTUAL RECONSTRUCTION OF HISTORIC ISLAMIC GEOMETRIC PATTERNS

A Dissertation

by

RIMA AHMAD AL AJLOUNI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Robert B. Warden
Committee Members,	Donald H. House
	Douglas F. Wunneburger
	Richard A. Burt
Head of Department,	Phillip J. Tabb

August 2005

Major Subject: Architecture

ABSTRACT

Development and Evaluation of a Digital Tool for Virtual Reconstruction
of Historic Islamic Geometric Patterns.
(August 2005)
Rima Ahmad Al Ajlouni, B.Sc. Arch. Eng., University of Jordan;
M.Sc. Arch. Eng., University of Jordan
Chair of Advisory Committee: Prof. Robert B. Warden

For the purpose of cultural heritage preservation, the task of recording and reconstructing visually complicated architectural geometrical patterns is facing many practical challenges. Existing traditional technologies rely heavily on the subjective nature of our perceptual power in understanding its complexity and depicting its color differences. This study explores one possible solution, through utilizing digital techniques for reconstructing detailed historical Islamic geometric patterns. Its main hypothesis is that digital techniques offer many advantages over the human eye in terms of recognizing subtle differences in light and color. The objective of the study is to design, test and evaluate an automatic visual tool for identifying deteriorated or incomplete archaeological Islamic geometrical patterns captured in digital images, and then restoring them digitally, for the purpose of producing accurate 2D reconstructed metric models.

An experimental approach is used to develop, test and evaluate the specialized software. The goal of the experiment is to analyze the output reconstructed patterns for the purpose of evaluating the digital tool in respect to reliability and structural accuracy, from the point of view of the researcher in the context of historic preservation. The research encapsulates two approaches within its methodology; Qualitative approach is evident in the process of program design, algorithm selection, and evaluation. Quantitative approach is manifested through using mathematical knowledge of pattern

generation to interpret available data and to simulate the rest based on it. The reconstruction process involves induction, deduction and analogy.

The proposed method was proven to be successful in capturing the accurate structural geometry of the deteriorated straight-lines patterns generated based on the octagon-square basic grid. This research also concluded that it is possible to apply the same conceptual method to reconstruct all two-dimensional Islamic geometric patterns. Moreover, the same methodology can be applied to reconstruct many other pattern systems. The conceptual framework proposed by this study can serve as a platform for developing professional softwares related to historic documentation. Future research should be directed more towards developing artificial intelligence and pattern recognition techniques that have the ability to suplement human power in accomplishing difficult tasks.

DEDICATION

To my father, Ahmad G. Al Ajlouni, and mother, Rayya M. Athamneh
To whom I owe everything I have accomplished in my life.

To my brothers and sisters, for all their love and support.

ACKNOWLEDGMENTS

I would like to thank my committee chair, Dr. Robert, and my committee members, Dr. Donald, Dr. Douglas, and Dr. Richard, for their guidance and support throughout the course if this research.

Thanks also to my friends, colleagues, and the department faculty and staff for making my time at Texas A&M University a great experience. I also greatly appreciate the full sponsorship received from the Hashemite University in Jordan.

Finally, special thanks to my parents, sisters, and brothers for their love, encouragement and support.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I
INTRODUCTION

1.1 Introduction to the Study

Through history questions related to the cultural past and historical remains have triggered man's curiosity to discover their hidden mysteries. The scientific interest in this area was evident with the beginning of the Middle Ages. New methods and forms were invented to recreate places of the past (Goncalves and Mendes 2004); but at the time these methods were all manual. It was not until after the end of the Second World War that approaches changed radically as a result of the rapid urban and industrial development, the emergence of new scientific techniques and philosophies related to the interpretation of cultural artifacts (Bowkett et al. 2001). The explosion of technological development in the second half of the twentieth century has marked another radical change. The advancements in computer and information technology provided scientists in all venues with powerful research tools (Williamson 2000).

Today Heritage preservation has caught under its umbrella a vast variety of technologies and techniques developed originally for other disciplines. Such technologies were adapted to the field of preservation in order to make the process of preserving our past rapid, accurate and scientific. In addition to the fields of Architecture and Archaeology, these methods have expanded to include other human sciences: Ethnography, History of Religion and History of Art; Exact sciences such as: Mathematics, Physics and Chemistry, and lastly engineering sciences like: Semiotics, Computer Sciences, Statistics, Visualization and Image Processing (Iakovlva 2004). The interaction between these different sciences has produced a more firm and concrete methodological foundation (Forte 1997).

This dissertation follows the format of the *Journal of Preservation Technology.*

Visualization sciences and digital technologies represent one band of this wide spectrum. The recent advancements in this field brought to light appealing opportunities to utilize its tools in addressing many challenges in the field of preservation (Bowkett et al. 2001). Tasks involving manual manipulation of vast amounts of data can now be performed more efficiently (Richards and Ryan 1985).

For the purpose of cultural heritage recording, a number of computer aided-documentation techniques are available today, not only to assist in the gathering and organizing of information but also in enabling its manipulation and display (Whiting and Nicjerson 1997). Nevertheless, the use of these techniques for the task of documenting complicated ornamental details still poses many practical challenges.

Detailed ornaments can be found in many cultural heritages of the world. Islamic Geometric Patterns present us with one example of such visually complicated objects, which are generated based on highly mathematical structures (Kritchlow 1976, El-Said 1993). These ornaments were extensively used in architecture all over the Islamic world, from Spain in the west to China and Indonesia in the east (Jones 1978). Until now no digital tools have been available to accurately and efficiently document and reconstructs these patterns.

Although the computer offers many visual tools for handling these tasks, it also poses many practical challenges. The task of identifying spatial patterns is strongly related to the computer's ability to read digital images accurately enough to produce acceptable metric data (Jimenez and Chapman 2002). Unfortunately, existing techniques have limitations when dealing with reading and interpreting visually complex geometric ornaments. Almost all of these techniques rely heavily on the user's input, and on the subjective nature of our perceptual power in understanding the complexity and depicting color differences.

Another challenge facing the field of preservation is posed by the fact that archaeological reality often deals with ornaments that are broken, incomplete or hidden. Therefore, computer visualization is challenged to provide an interpretation of the available data, to simulate what is not there or cannot be seen (Barcelo 2002). One of the

major research challenges for archaeologists is the task of identifying spatial patterns in archaeological sites and features (Jimenez and Chapman 2002). Visualization can help us detect these hidden patterns by highlighting certain features in images and suppressing others. Virtual reconstruction represents an extremely useful medium for exploring and investing historical environments (Guidazzoli et al. 2004). It can be used as an interpretive tool (Forte 1997), which enhances our understanding of complex data, provides us with a deeper insight into these environments and helps in making correct preservation decisions (Lock and Harris 2000).

Today, many techniques are available to enable extensive analysis of the evidence they discover (Bowkett et al. 2001). This can mostly be attributed to the rapid development of computers and a strong trust and reliance on digital data (Reilly 1991), which in turn relies on the development of new technologies and techniques for data acquisition, data processing, structuring and representation (Campo 2004). Moreover, graphical systems are redefining the processes of collecting, representing, and interpreting archaeological evidence (Reilly 1991). Digital Image processing techniques are employed to solve archaeological problems through building virtual reconstructions (Reilly 1991). Specialists have been able to create visualizations of large-scale 3D models (Aliaga et al. 2003), generate automatic models using aerial images (Hu et al. 2003), reconstruct monuments through photogrammetric means (Drap et al. 2002), classify archaeological artifacts (Kampel and Sablatnig 2002), and create Image-based 3D modeling methodologies for archaeological applications (Gool et al. 2002).

To be able to document and reconstruct eroded patterns; recording techniques should be able to accurately and objectively read color information, then interpret such information through a visual tool that contains within its code some built-in-mathematical intelligence that deals with the geometry of patterns. As a result, this research is directed towards utilizing visualization sciences and digital image techniques to develop new tools that have the power to understand the complex nature of such patterns and mathematically predict the missing parts.

1.2 Statement of the Problem

This research addresses two main problems. The **first problem** is related to the difficulties attached to the goals of efficiently and accurately documenting Islamic Geometric Patterns. Available documentation techniques have failed to handle this task without sacrificing one of the above goals. The visual complexity of these patterns represents the main obstacle in the documentation process. The manual and the subjective extraction of pattern features can result in poor accuracy, excessive use of time, money, effort, and other valuable resources. It also contain numerous human and archaeological biases (Reilly 1991).

The **second problem** is related to the difficulties in producing accurate virtual reconstruction of these patterns. Before preserving these delicate artifacts, it is important to produce accurate virtual reconstructions that will help professionals to make decisions and to test the different options (Bowkett et al. 2001). Existing techniques still have many problems when the task involves reconstructing patterns that are deteriorated or not visually clear. Here we must deal with the difficult task of recognizing faint traces of eroded or missing parts. This is combined with the need for specialized knowledge about the mathematical rules of patterns in order to regenerate the missing data. With the note that these tasks are still performed manually or with the aid of drafting software, which consumes time, money and effort. Existing manual methods do not encourage the sharing of information, nor readily lend themselves to multiple access.

Current applications are focused on visualization and image processing, but not much work has been done on utilizing machine vision techniques (Zuech 2000). A variety of computer applications have been developed in the area of creating virtual models based on existing data, but little has been done on developing reconstruction techniques that incorporates the mathematical logic of these geometric patterns into their structure to produce accurate models, which can be reliable for providing real measurements.

1.3 Background

Together with Calligraphy and Arabesque, Geometry represents one of three major art types in traditional Islamic culture. Geometry as an art form was manifested throughout the Islamic world since the dawn of Islamic civilization (Stirlin 1996). It was applied to stonework, metalwork, woodwork, ceramics, textiles, carpets, miniatures and architectural decorations (Jones 1978). For more than thirteen centuries geometric patterns acted as a unifying factor, linking architectural products from all over the Islamic world, from Spain in the west to China and Indonesia in the east (Jones 1978). Today, Archaeologists and Historians are faced with the challenge of preserving the delicate and precious details of these patterns. The precise recording of cultural artifacts is an essential step for their preservation and restoration (Oh et al. 2004). It represents a prerequisite for any accurate reconstruction or future analysis (Mikhail 2001). The tasks of efficiently and accurately recording and reconstructing these visually complicated geometric patterns are far from being solved, and still pose many practical challenges.

Existing documentation techniques have many limitations when dealing with these patterns. Traditional recording technologies such as hand measurement and total station surveying have many efficiency and accuracy problems (Dallas 1980, Mikhail 2001). Such problems are associated with the need for measuring a considerable number of reference points that compose these patterns, which requires time, money and effort (Avern 2001). It is also crucial to provide precise measurements by non-contact methods to avoid damaging historical artifacts (Oh et al. 2004), a condition that is hard to satisfy when applying such methods. Moreover, these methods are invalid when it is physically difficult to approach the surface under investigation (Atkinson 1980).

The methods of photogrammetry[1] represent a major player in the field of historic documentation. Close range Photogrammetry, is considered one of the great contributions to recording and monitoring of cultural heritage. It illustrates the first step in preservation and restoration of any architectural or other cultural monument, object or site. It serves as a support to research in archaeological, architectural or art historical research as well (Grussenmeyer et al. 2002). Photogrammetry removes the need to physically access each point at which a measurement is required. The exact location can be determined from the photography (Mikhail et al. 2001). This allows the field measurements to be done at the office, which reduces the required on-site period and allows a large amount of work to be completed in a short time. (Shmidek 1974).

During the last two decades of the twentieth century, photogrammetry has experienced a major change. The rapid development in imaging and computer technology made possible the development of Digital-photogrammetry[2] (Mikhail et al. 2001). The photogrammetry process using digital data is fast, accurate, reliable, easy and inexpensive; digital photos are high quality and low-cost (Bowkett et al. 2001). Although, it has many advantages over other techniques, still one important drawback is associated with its point-to-point measurement principals. Unfortunately, techniques, which employ this recording procedure, still have many limitations when dealing with recording fragile, visually complex geometric patterns. The limitation of such procedures is related to the fact that the recording processes depend heavily on the users' input and the subjective nature of human perception.

Research in visual perception has proven that the eye is limited in detecting color differences (Smith 2001), and that the human visual system is considered to be a

---

[1] Photogrammetry has been defined by the American Society for photogrammetry and remote sensing (ASPRS) as, "The art, science, and technology of obtaining reliable information about physical objects and the environment through the processes of recording, measuring, and interpreting photographic images and patterns of electromagnetic radiant energy and other phenomena" (Ghosh 1979). The concept of photogrammetry can be summarized as to " accurately establish the geometric relationship between the object and the image. It is a process of obtaining geometric information, e.g. position, size and shape of any object" (Grussenmeyer et al. 2002).

[2] Digital photogrammetry is the technique of measuring objects (2D or 3D) by digital photos; the practice of using pixels and image processing techniques to arrive at geometric information (Hanke 2000).

poor color estimator (Matkovic et al. 2002). Moreover, the subjective nature of human visual perception implies that different human beings perceive color differently (Beals 2001). Color perception of the Red-Green-Blue encoding system used in digital images represents another important limitation for the human vision (Smith 2001). This limitation can be magnified when dealing with images that contain complex geometry. The extraction of features such as lines and points or intersections of lines in many cases can cause geometric optical illusions (Luckiesh 1965, Wade 1982, Gibilisco 1990). The perception of contrast depends mainly on the level of frequency of objects (Mutz 2003). Consequently, patterns with spatially separated areas and uniformed frequency can cause many illusional effects (Zuech 2000).

On the other hand, laser scanners may be more successful in performing 3D digital documentation (Yamada et al. 2003), but in most cases, they fail to detect 2D color differences situated on the same 3D plane which makes them poorly suited for documenting 2D geometric patterns. Moreover, these systems produce point clouds that still need to be processed to extract the right documentation features. This extraction process is usually performed by humans and is subjected to the limitations of the human optical system as explained earlier. Other methods such as rectifying photos only produce a geometrically corrected photo. To extract documentation, we need to trace the features of the image, which brings us back to square one: the subjective nature of our perception.

The second major difficulty facing the preservation of Islamic patterns is producing accurate reconstructions. Through the years, this task has been performed manually or by the aid of some drafting programs. The problem with such methods is related to the subjective and limited human ability of recognizing faint traces of subtle color evidence. Also, the need for special knowledge, skills and deep understanding of the rules for geometric patterns are crucial in its analysis and testing of the different reconstruction options. The key challenge to reconstruction lies in accurately interpreting the existing data (Bowkett et al. 2001).

Identifying spatial patterns in digital data has been one of the major interests for archaeological research. Recently, experts in computational morphology, computational geometry and pattern recognition have developed a range of techniques that deal with reading and interpreting digital data (Jimenez and Chapman 2002). The body of related literature exploring this type of research covers research areas, such as: the use of pattern recognition methods for identifying and extracting features (Guo et al. 2003), edge and corner detection (Kovesi 2003), defining areas with common properties (Yanamura and Saji 2003), and shape matching (Srisuk et al. 2003).

To overcome the limitation of the subjective nature of human perception, the computer offers tools that are more accurate and objective in detecting color within its own digital format. Moreover, for the task of reconstructing patterns, the computer offers tools that are more capable of testing many configurations and theories. It also offers many algorithms for mathematical analysis, shape recognition and generation.

1.4 Purpose of the Study

The purpose of this research is to explore the possibilities of using digital techniques to supplement the human power in accomplishing tasks related to historic preservation. Its main hypothesis is that these techniques offer many advantages over the human eye in terms of recognizing subtle differences in light and color. It also has a great capacity for testing many scenarios quickly, accurately and easily compared to the limitations of the human's capabilities (Bowkett et al. 2001). The automatic process can remove numerous human and archaeological biases (Reilly 1991).

The aim is to investigate the use of digital image techniques to reconstruct and document our architectural cultural heritage. It explores the possibility of incorporating a pattern's mathematics with digital techniques to develop new ways of analyzing and recording geometric patterns. Although this research focuses specifically on Islamic patterns, the underlying goal is to develop general ideas and principles that might then be applied to other ornamental styles. The range of possible styles to investigate is huge, and this is only the first step before moving on with new forms of ornament.

The objective of the study is to design, test and evaluate an automatic visual tool for identifying deteriorated or incomplete archaeological geometrical patterns captured in digital images, and then restoring them digitally, for the purpose of producing accurate 2D reconstructed metric models.

1.5 Significance of the Study

This research explores the world of historical geometry as one important manifestation of the great cultures of the world. It lends itself to examine the historical Islamic patterns as one of such great heritages. It is not until recently that western scholars became aware of the importance of such art and came to acknowledge it as one of the most powerful forms of sacred art (Kritchlow 1976, Abas and Salman 1992). These objects are subject to erosion and vandalism, and as long-lived artifacts, they have gone through many phases of construction, damage, and repair. It is important to keep accurate records of these artifacts, for the purpose of producing reconstructions that preserve and reveal their aesthetic and historic value based on respect for original material and authentic documents. By protecting these precious artifacts we will help to keep one of the great heritage assets of the world.

The art of Islamic symmetrical patterns has proven to be an elegant method for the study of symmetry (Kritchlow 1976, El-Said 1993). These patterns offer a rich source of research for artists and are also of interest to mathematicians, architects, archaeologists and others. Therefore, the development of an analysis tool can be very useful in studying and analyzing the evolution of Islamic symmetric repeated patterns, and also, to explore ways in which these patterns can be created algorithmically. Moreover, these patterns have strong relations with other sciences such as theoretical physics, crystallography, chemistry and biology (Abas and Salman 1992, Kaplan 2000).

This research hopes to draw more attention to the importance of utilizing new technologies -developed by other disciplines- in tackling practical problems related to historic preservation. It belongs to a broad line of research with an ambitious goal of developing tools that can take over some of the difficult and subjective tasks still

performed by humans. By building intelligence into visualization technologies, we can create powerful tools that are able to perform tasks more easily, quickly and efficiently than by human operators.

1.6 Theoretical Framework

My research stance assumes that knowledge is absolute and formal. I observe the world through an analytic structuralist eye. I am relatively independent and stand apart from the phenomenon that I observe. As a researcher, it is crucial to strengthen my credibility by minimizing my tendency to be influenced by any biased or particular position (Nodelman 1970). In the case of this study, such bias is controlled by the fact that I will be dealing with abstract and absolute laws and therefore my subjective interpretation is kept to a minimum.

This research acknowledges that Islamic patterns are products of a unique cultural process, and have an important social and cultural significance; still it is not within the scope of this research to discuss their interpretive meaning. This study focuses on exploring the patterns for their physical and formal qualities. Therefore, even though I am not completely independent from the culture that produced such patterns, my credibility is not compromised by this fact, and I remain objective. Geometrical patterns will be studied based on an absolute knowledge and in relation to their own system (Nodelman 1970).

The internal validity of such research is defined by how successful the instrument is in performing its task (Wohlin et al. 2000). In this case, it will be a measure of how successful and accurate the designed tool is in depicting pattern geometry and measurements. External validity is concerned with the generalization of results (Wohlin et al. 2000), which is concerned with the applicability of the designed tool in reconstructing and documenting other patterns.

1.7 Research Question and Hypothesis

1.7.1 Research Hypothesis

Given the formal spatial nature of the digital image format and the mathematical structure of Islamic pattern geometry, it is possible to incorporate the two logical systems within the computer virtual environment to develop a visual tool capable of interpreting surviving evidence through analyzing digital data of partially deteriorated patterns, and then simulating the missing parts to produce accurate two-Dimensional reconstructed models.

1.7.2 Research Question

This research addresses the question of how to incorporate the mathematical and algorithmic rules of pattern structure with digital image techniques using computer tools to develop a program capable of reading and interpreting partially deteriorated Islamic patterns represented in digital image format to simulate the missing parts and produce an accurate virtual reconstruction?

1.8 Experimental Variables

1.8.1 Independent Variables

a) The designed digital tool (Constant variable).
b) The operator (Constant Variable).
c) Digitizing equipment: The digital camera (Constant variable).
d) Sample generation and deterioration techniques and procedures (Constant variable).
e) Pattern’s base grid and orientation (Constant variable).
f) Unit of assignment (experimental samples):
    - Level of pattern complexity
    - Image resolution
    - The shape and degree of pattern deterioration.

1.8.2 Dependent Variables

a) Structural Accuracy: Does the tool capture the right geometric structure?
b) Repeatability: Is the experiment consistent in terms of producing the same results when repeating the procedures.

1.9 Scope of the Study

The process of automatically producing digital reconstructions of deteriorated Islamic Geometric patterns requires addressing many practical challenges. The tasks required to solve these challenges, range from exploring the fields of image acquisition, image processing, and pattern recognition to the visualization of the final results. This research is only concerned with one link in this long chain. It explores the field of pattern recognition and presents one way of identifying and reconstructing geometric patterns through incorporating its mathematical rules into adapted techniques of pattern recognition.

This study explores the possibility of utilizing digital image techniques to develop a digital tool that proves the validity of such a line of research. It provides a proof of concept. The study does not attempt to develop professional software; rather, it explores an idealized problem whose solution might lead to advances in developing professional tools.

Research results are based on conducting a very controlled experiment that assumes a perfect image as an input for analysis. It does not lend itself to examine or address the challenges attached to image acquisition or processing. Experiment samples include one category of Islamic Geometric patterns. The study tested its hypothesis on straight-line patterns generated based on one basic grid of octagons and squares. Straight-line patterns were assumed. The number of patterns tested was around 30 different configurations. Qualitative analysis was done to evaluate the results.

CHAPTER II
LITERATURE REVIEW

The body of literature related to this research covers two main topics: Historic Islamic Geometric Patterns, which include studies related to its origin, significance, symbolism, types, properties, construction methods, mathematical rules and its application in computer graphics, and Digital Imaging techniques, which explores studies related to computer vision in general and pattern recognition in particular.

2.1 Islamic Geometric Patterns

2.1.1 Introduction

The rise of Islamic culture in the seventh century has marked the beginning of a new artistic, decorative and sacred tradition (Stirlin 1996). The tradition was completely inspired by a deep religious philosophical and cosmological approach, which embodied all aspects of life and manifested itself in every product (Kritchlow 1976, Al-Bayati 1981). The spiritual origin of this sacred art is best explained by Seyyed Hossein Nasr[3]:

> Islamic spirituality could not but develop a sacred art in conformity with its own revealed form as well as with its essence. The doctrine of unity, which is central to the Islamic revelation, combined with the nomadic spirituality, which Islam made its own brought into being an aniconic art wherein the spiritual world was reflected in the sensible world not through various iconic forms but through geometry and rhythm, through arabesque and calligraphy, which reflect directly the worlds above and ultimately the supernal sun of Divine Unity (Kritchlow 1976, 6).

---

[3] Professor Seyyed Hossein Nasr, one of the world's leading experts on Islamic science and spirituality, is University Professor of Islamic Studies at George Washington University. Professor Nasr is the author of numerous books including Man and Nature: the Spiritual Crisis of Modern Man (Kazi Publications 1998), Religion and the Order of Nature (Oxford 1996) and Knowledge and the Sacred (SUNY 1989).

Islamic art encompasses great achievements in Geometry, Calligraphy and Arabesque. These visually diverse art forms grow out from the same spiritual origin to represent the multiple manifestation of the divine (Kritchlow 1976, Al-Bayati 1981). For more than thirteen centuries they acted as unifying factors. They have linked the architectural products from all over the Islamic world, extending across Europe, Africa and Asia (Jones 1978, Kaplan and Salesin 2004). The four fundamental concepts in Islam: beauty, harmony, symmetry and unity are all intrinsic to the contemplative side of Islamic Art (Grube 1978). Expressing these concepts was never a subjective matter. Islamic artists did not seek to express themselves as such, but rather aimed to honor matter, and reveal the objective nature of its meaning (Kritchlow 1976, Jairazbhoy 2000). The use of Geometric patterns is one of the chief characteristics that give the Islamic artistic heritage its distinct identity. Geometry as an abstract art form was developed in part due to the discouragement of images in Islam on basis that it could lead to idolatry (Danby 1995). Like every thing in Islam it has an "*outer*" and "*inner*" meaning (Al-Bayati 1981, Danby 1995). Seyyed Hossein Nasr explained the importance of this for accomplishing the internal balance of Islamic art:

> Islamic art is predominantly a balance between pure geometric form and what can be called fundamental biomorphic form: a polarization that has associative values with the four philosophical and experiential qualities of cold and dry - representing the crystallization in geometric form- and hot and moist – representing the formative forces behind vegetative and vascular form. The one aspect reflects the facts of a jewel, the purity of the snow flake and the frozen flowers of radial symmetry; the other is the glistening flank of a perspiring horse, the silent motion of a fish winding its way through the water, the unfolding and unfurling of the leaves of the vine and rose (Kritchlow 1976, 8).

Recently, these patterns have evoked a new "humanistic" approach to mathematics education. Humanistic mathematics is a philosophy of teaching mathematics, which guides students through mathematical ideas by the use of imagery, history, as well as other interdisciplinary medias (Abas 2001, Tennant 2004).

2.1.2 Types of Islamic Patterns

The vast variety of geometric formation and the strict rules of its generation reveals an important inner dimension of Islamic tradition: “unity in multiplicity and multiplicity in unity” (Jones 1978). This principal is represented by means of different mathematical forms symbolizing the constant celestial archetypes within the cosmos (Mostafa 1955). Most of these geometric patterns can be grouped under the following categories:

1. Geometric patterns based on the Square Repeat Unit and the “Root Two” proportion system. This includes all patterns generated by the division of the circle to four, and all patterns generated from the multiples of four (Kritchlow 1976, El-Said 1993) (Figure 2.1).

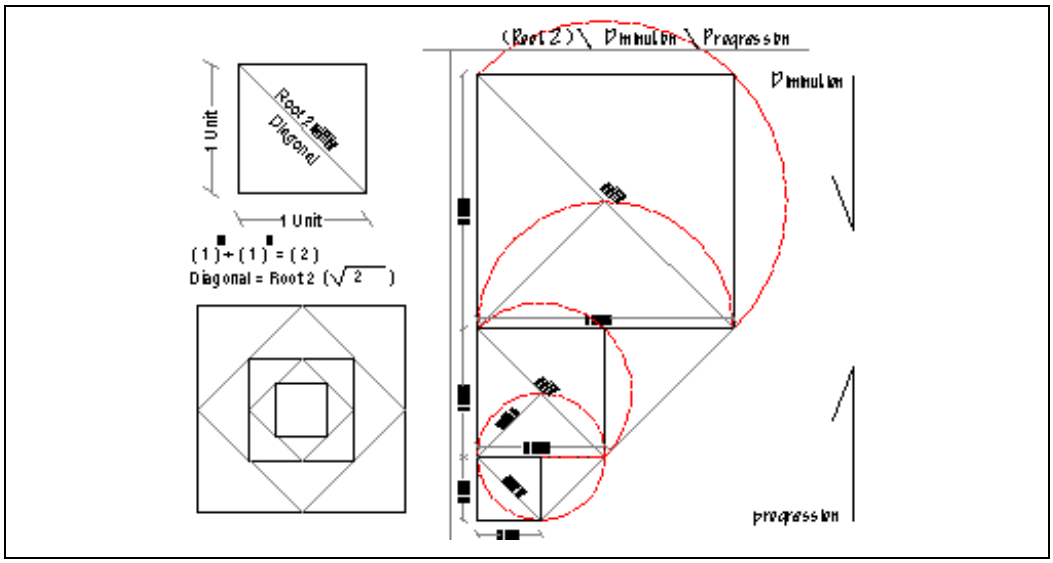


Figure 2.1 Root Two Proportion System.

2. Geometric patterns based on the Hexagonal Repeat Unit and the “Root Three” proportion system. This includes all patterns generated by the division of the circle into three or six, and all patterns generated from the multiples of six

(Kritchlow 1976, El-Said 1993) (Figure 2.2). For example Hexagons and Dodecagons.

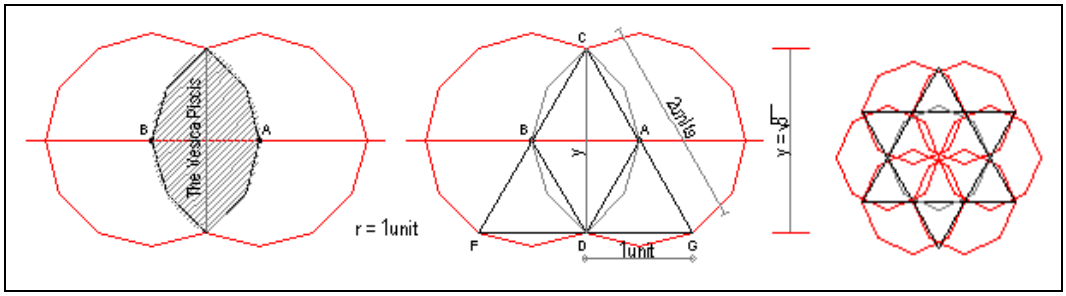


Figure 2.2 Root Three Proportion System.

3. Geometric patterns based on pentagon Repeat Unit and the "Golden Ratio" proportion system. This includes all patterns generated by the division of the circle into five, and all patterns generated from the multiples of five (Kritchlow 1976, Danby 1995) (Figure 2.3). For example the ten folded base pattern.

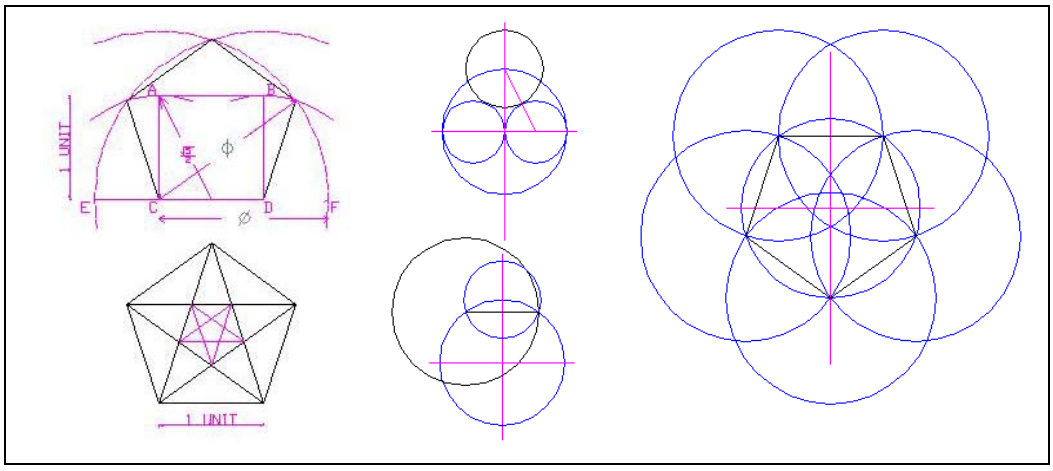


Figure 2.3 The Golden Ratio Proportion System.

2.1.3 Patterns' Properties

Islamic geometry has successfully integrated the bounding laws of pattern geometry with the beauty and harmony of colors and rich combinations (Kritchlow 1976). Although, these patterns are physically fixed in time and space, still their visual manipulation and rhythm indicates movements in these dimensions (Danby 1995).

a) Geometrical Properties

1. They are made up of a single repeated geometric unit; through the principals of repetition and symmetry (Kaplan and Salesin 2004). The making of a complex whole by the repetition of basic elements satisfies both an apparent need for intense visual stimulation and the Islamic search for unity (Kritchlow 1976, El-Said 1993).
2. They are two-dimensional. Geometric designs maintain two-dimensionality by fitting all the polygonal shapes together like the pieces of a puzzle, leaving no gaps. Such a construction is known mathematically as a "*tessellation*" (Montesinos 1985).
3. They radiate symmetrically from the center points. Any polygon in a design can be chosen as a central point from which the pattern radiates symmetrically (Kaplan and Salesin 2004, Ostromoukhov 1998). Some patterns can be seen as radiating from a star surrounded by different polygons (Jones 1978).
4. They are not designed to fit within a frame. Expansion forms the basis of the characteristic of the geometric patterns (Clevenot 2000). Since the patterns consist of repeating elements that move outward from a central point, they could go on indefinitely but are limited by the edges of the space they decorate.
5. They are constructed based on a limited number of basic grids, which are generated from patterns of circles (Gonzalez 2001). The complex geometric patterns are all elaborations of simpler constructions of circles, which are often used to determine grids (Kritchlow 1976, El-Said 1993). Mathematically these

grids are known as regular tessellations, in which one regular polygon is repeated to fill the plane.

The main basic grids are: a) Basic grids based on the equilateral triangles and hexagons and its multiples (Figure 2.4). b) Basic grids based on the squares, octagons and their multiples (Figure 2.5). c) Basic grids based on the pentagon and its multiples (Figure 2.6). d) Basic grids based on a combination of basic categories (Figure 2.7).

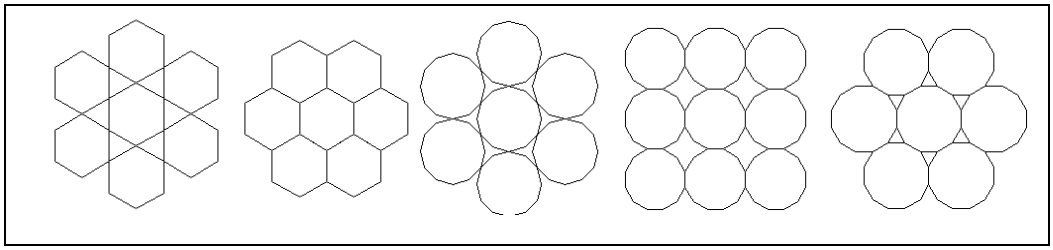


Figure 2.4 Basic Grids Based on Equilateral Triangles and Their Multiples.

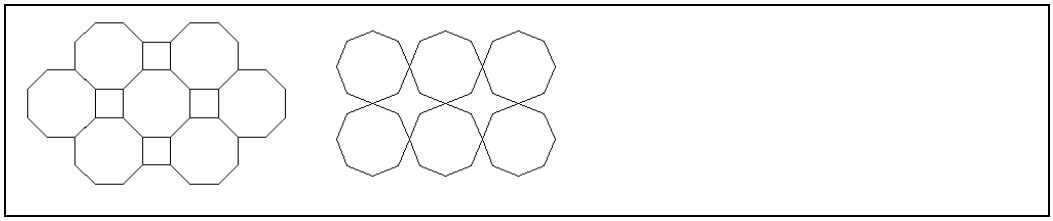


Figure 2.5 Basic Grids Based on Squares and Their Multiples.

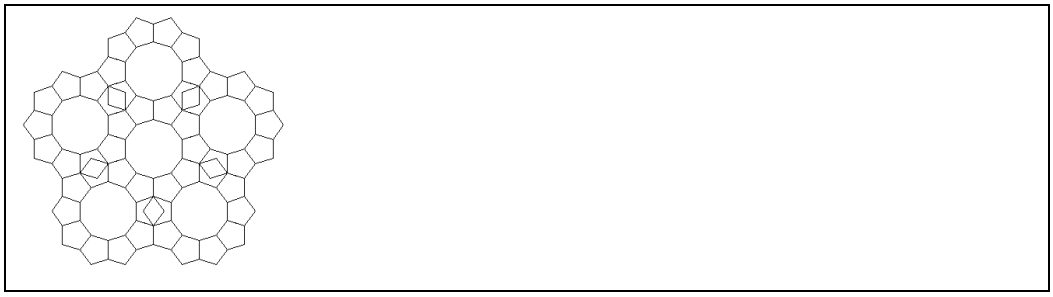


Figure 2.6 Basic Grids Based on The Pentagon and Its Multiples.

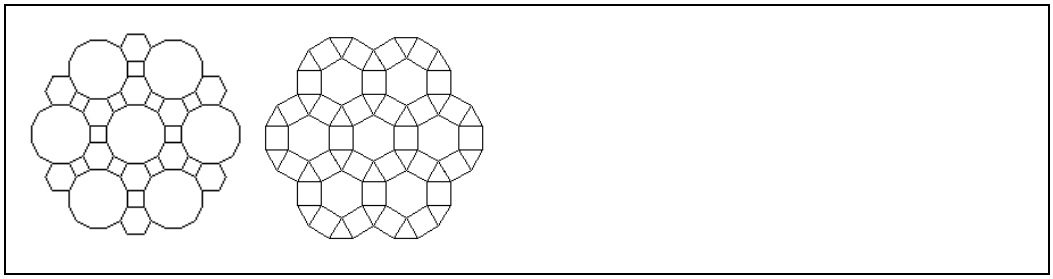


Figure 2.7 Basic Grids Based on a Combination of Different Basic Categories.

Even though geometric patterns are generated from simple forms; they were combined, duplicated, interlaced, and arranged in fascinating combinations, which became one of the most distinguishing features of Islamic art. Although they are generated based on very strict rules of geometry, the geometric ornamentation in Islamic art suggests a remarkable amount of freedom; in its repetition and complexity. It offers the possibility of infinite growth and can accommodate the incorporation of other types of ornamentation as well (Jones 1978).

b) Visual Properties

The complexity of Islamic patterns is stressed through the use of different colors, reflecting and shining materials and glazes, the contrasting of textures and the manipulation of planes (Jones 1978, Al-Sayyed 1999). Visual effects are based on these properties:

1. Color

The use of color is an important characterization of Islamic decoration, which was supported by great skill in mixing and producing colors. By applying these colors to different designs, a new relationship is developed based on the established rules of density, succession and contrast (Al-Sayyed 1999). Generally in Islamic art, certain colors are used more often than others. The choice of colors should reinforce the effect of the decorative composition, which creates special effects intended to highlight the optical effects of the geometrical compositions (Jones 1978). Red, orange and gold, are

associated with fire and the sun. Therefore, then are used to suggest warmth and heat. Green and blue, are associated with cold lunar developments and are used to suggest cold. If colors from different groups are applied, the effect will produce a sense of dynamics and amazing visual pleasures (Al-Sayyed 1999).

2. Texture

The application of patterns in different materials added a rich diversity of textures, which contributed to the fascinating optical effects. Even though different materials were used, they were unified by geometric principals (Jones 1978). These patterns were applied to stonework, metalwork, woodwork, ceramics, textiles, carpets, miniatures and architectural decoration (Clevenot 2000, El-Said 1993).

3. Contrast

This effect was achieved by playing with negative and positive areas. These were created either by using different contrast colors (Jones 1978, Al-Sayyed 1999), by the curved patterns, which permits the play of light and shade (Jones 1978), or by using different surface textures and different reflective materials (Jones 1978, Al-Sayyed 1999).

2.1.4 Geometric Construction of Islamic Patterns

All Islamic designs were constructed by only using a compass and a straightedge; therefore the circle becomes the foundation for Islamic patterns (Kritchlow 1976, Jones 1978, El-Said 1993). The use of geometry through out the Islamic world expressed a strong intuition of a universe based upon logic and order. The generating rules of this art were founded on a strong mathematical background. The work of Euclid and other Greek mathematicians were well known in the Islamic world (Kritchlow 1976, Jones 1978, El-Said 1993).

Islamic Geometric Patterns were applied to all kinds of materials: metal work, woodworks, ceramics, textiles, carpets, stone, fabric and miniatures. The universal

application of these patterns implies that they were created based on solid formal methods (El-Said 1993). The act of designing and applying these patterns was considered a form of worship and encapsulated a divine religious experience. These artists and the methods they used were secretive, and only few passed on this tradition until it was lost (Abas and Salman 1992, El-Said 1993).

The generating force of patterns lies in the center of the circle, which represents the point at which all Islamic patterns begin. It is the symbol of a religion that emphasizes one God, the center of universe (Kritchlow 1976). This decorative art is generated from a discrete geometrical unit using the circle as its basis, and then applying the principles of repetition, and symmetry to it (Kritchlow 1976, El-Said 1993). Although each pattern has its own distinct geometrical design, the vast varieties of ornamental compositions are based on a simple constitutive geometry, which is generated from a limited number of simple base grids of polygons (Gonzalez 2001).

A. The Construction of the Basic Grid

The basic grids are constructed based on a systematic formation of circles by connecting the circles' intersection points. Figure 2.8 demonstrates the generation process for the octagon- square basic grid (Al-Ajlouni 1999). It is possible to generate any regular polygon by equally dividing the circumference to the required number of parts (El-Said 1993).

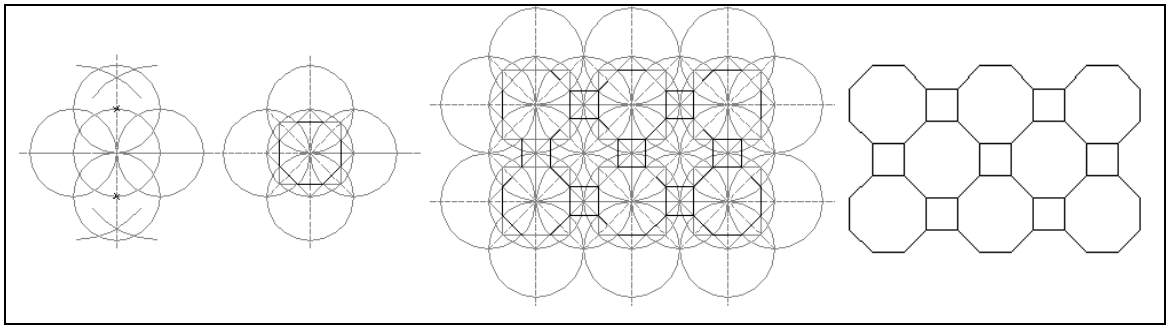


Figure 2.8 The Generation Process for an Octagon- Square Basic Grid.

B. The Construction of the Repeated Unit (Star Pattern)

The repeat unit is constructed using single polygons. The star unit is formed by an array of lines connecting either the intersection points of the polygon's sides, or connecting the mid points of the polygons sides (Danby 1995). The repeat star can be as simple as one array of lines or a combination of two or more simple stars (arrays), as shown in Figure 2.9.

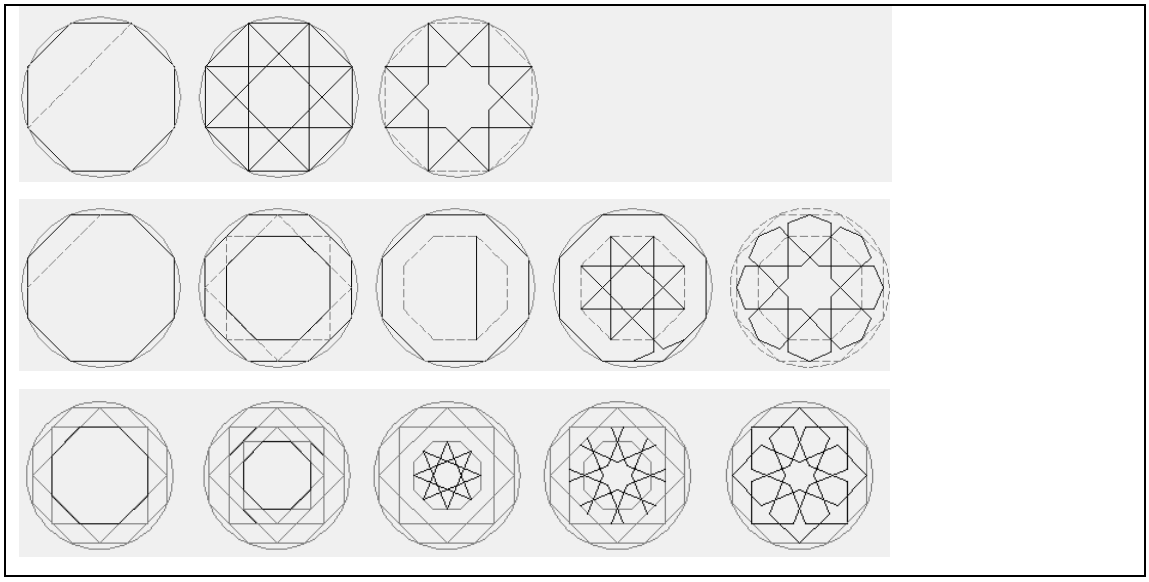


Figure 2.9 The Process of Constructing the Repeated Star.

C. Pattern Generation

Combining the repeated units with the basic grid will generate the final pattern. The basic grid will be filled-in with any choice of repeat star unit, and then the lines of all stars will be extended to meet other lines at the symmetrical lines. The end result will not show the basic grid, but only the replication of the star units. Figure 2.10 demonstrates the process of generating the final line pattern. Filling the same basic grid with different repeat stars will result in different final patterns.

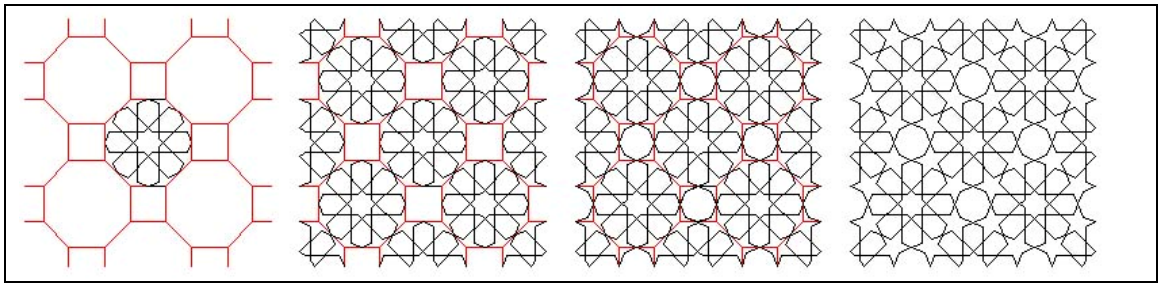


Figure 2.10 The Process of Generating the Final Line Pattern.

D. Rendering the Final Line Pattern

Historically, these patterns were never merely drawn as lines. Often, the lines are thickened when incorporated into different material and sometimes broken up to suggest an interlacing pattern (Gonzalez 2001). Figure 2.11 shows some final rendering effects of an octagon-based pattern.

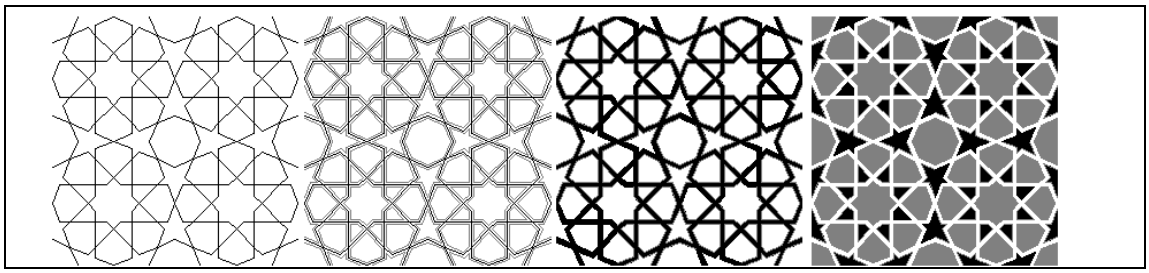


Figure 2.11 Some Final Rendering Effects of Islamic Patterns.

2.1.5 The Application of Geometry in Mathematics and Computer Graphics

It was not until the 1980s that mathematical research geared some of its interest toward exploring the field of Islamic geometric patterns as a method for the study of symmetry and mathematics (Grunbaum and Shephard 1987). In fact Grunbaum and Shephard noted that the most exciting developments related to this field were only twenty years old. In 1987 they wrote a book that explores geometric patterns. It deals

with mathematical analysis of tiling using regular polygons. Other research developed theories discussing the formation and analysis of mosaics, lattices and 2D plain ornaments (Wieting 1982), and explored the relationships between tessellations and the different manifolds (Montesinos 1985).

Since 1990 much research has been done to study and analyze the evolution of the Islamic symmetric repeat patterns, and to explore ways in which these patterns can be constructed algorithmically. Abas and Salman (1992) conducted an analytical study of different group patterns, through which they developed an algorithm based on group theoretic approach[4] to be used with computer graphics to generate two-dimensional symmetric periodic patterns. Ostromoukhov (1998) extended Abas and Salman's analysis to develop mathematical tools for two-dimensional pattern analyses using planar symmetry groups and Cayley diagram[5].

In the field of computer graphics, many tools and algorithms were developed for automatic construction, drawing and visualizing of ornamental patterns. At the second SIGRAPH conference Alexander (1975) presented a system for drawing figures constrained to seventeen ornamental design types. Others have developed mathematical software such as Kali (Amenta and Phillips 1996) and recently Tess (Pedagoguery Software Inc. 2002). This can be used to interactively generate different planar ornamental tessellations by applying the rules of symmetry. Andrew Glassner has developed many related computer algorithms. In 1996 he examined the synthesis of frieze patterns and the application of these symmetry patterns in computer graphics (Glassner 1996). In 1998 he talked about the relationship between the geometry of reflection in a line, and specular reflection in a mirror (Glassner 1998). In 2000 he showed how to create a variety of patterns using hierarchies of symmetry elements and image-processing actions (Glassner 2000).

---

[4] Group theory explains how two-dimensional geometric patterns can be analyzed to seventeen different basic types. It also defines the minimum amount of information to generate any pattern (Abas and Salman 1992).

[5] Cayley diagram is a graph, which explains the consecutive states of the basic geometrical grid under a sequence of transformations (Ostromoukhov 1998).

Recently research in computer graphics adapted the technique of reconstructing geometric patterns by placing star units within a basic grid of polygons. Kaplan (2000) used this approach to develop a software implementation of the technique. The study of Kaplan and Salesin (2004) extends Kaplan's analyses to introduce a more generalized system within a novel parameterized collection of tiling formations. Moreover, they introduced the idea of absolute geometry, which allowed them to create designs on the sphere and in the hyperbolic plane. Arabeske software (Dispot 2004) is a Java tool designed to help construct and draw patterns. It is particularly aimed at 3D artists wishing to use complex patterns for their scene settings. It can also be used as a special 2D drawing tool to design original patterns. It relies on a heavy use of symmetry groups. Recently, Kaplan (2005) presented a method for constructing transformable Islamic patterns by using different star unit "rosette" within the same basic grid of polygons. His construction was based on Hankin's "polygons-on –contract" method, which uses a grid of polygons as the basic grid.

2.2 Digital Imaging

2.2.1 Introduction

The advancement of digital imaging[6] technologies brought to light appealing opportunities to utilize its tools in addressing many challenges in the field of historic preservation. New modeling methods rely on digital data as an ideal media for representing historical sites and features. The increasing reliance on such fields is due to the development of many new techniques for data acquisition, data processing, computer vision, and computer graphics (Campo 2004). Their adaptation to the fields of archaeology and historic preservation may eventually prove to be one of the most important tools for discovery, analysis, and management of data (Williamson 2000). This is due to the price decline in digital computing technologies and advancements in capabilities (Jahne 1991). The ability to capture, store, present, and analyze images has

[6] The term digital imaging is used to encompass any manipulation of image-related data by the computer. This includes computer graphics, image formation, image processing, and computer vision.

proved to be particularly powerful for historic preservation (Williamson 2000). Each of these venues represents a new research opportunity ready to be explored.

In general, the field of digital imaging includes two main applications: the use of computer graphics tools to generate images such as CAD, visualization, and animation systems, etc., and the use of computers to operate on acquired images such as image analysis and computer vision (Zuech 2000). Computer vision is concerned with applying computer-based image analysis, interpretation, and pattern recognition, while computer graphics is concerned with generating images from object description (Shapiro and Stockman 2001). In the last twenty years, many advances have been made in the field of computer vision and computer graphics (Beraldin et al. 2004)

Both technologies have proved that they can contribute significantly to improving the productivity and quality of many operations (Stevens and Beveridge 2001, Zuech 2000). Moreover, the integration between them proved to be even more successful in addressing many practical problems in the fields of archaeology and historic preservation. Computer graphics are used to display the results of computer vision, and this is used to make object models (Shapiro and Stockman 2001).

2.2.2 Computer Graphics

Computer graphics is concerned with the graphical representation of conceptual ideas or mathematical descriptions. It deals with building images from non-iconic information (Castleman 1996). Graphic modeling has introduced new ways for presenting archaeological evidence (Reilly 1991).

The advances in this field have made it possible to generate accurate and large-scale pictorial databases of historical objects and sites. Computer visualization is becoming increasingly successful in capturing the richness of surface detail (Vasaros and Divinyi 2004).

Visualization should not be confused with seeing. Visualization refers to the creation of visual models of an object or process in order to examine it further

(Williamson 2000). Mechanical devices are used for translating the perceptual inputs into a geometric model interpreting color contrast (Barcelo 2002).

The applications of visualization have emphasized the creation of archaeological models to determine their interpretive potential (Earl and Wheatley 2002). Computer graphic techniques are employed to solve archaeological problems through building virtual reconstructions (Reilly 1991). Successful techniques usually are very specific and directed to solving a very specialized interest. Nevertheless, existing technologies can also be adapted to solve research problems (Zuech 2000). It is also important to note that current applications are more focused on visualization, and not much has been done on utilizing machine vision techniques (Zuech 2000).

2.2.3 Computer Vision

The key challenge for digital reconstruction lies on accurately interpreting the existing data (Bowkett et al. 2001). Identifying spatial patterns in digital data has been a major interest for archaeological research. Recently, experts in Computational Morphology, Computational Geometry and Pattern Recognition, have developed a whole range of techniques that deal with reading and interpreting digital data (Jimenez and Chapman 2002). The body of research exploring computer vision[7] generally covers three main areas: image formation and acquisition, image processing and image analysis (Castleman 1996, Zuech 2000).

The main characteristics of computer vision techniques include non-contact sensing, operations on an image, the use of computers to process and analyze data and an understanding of the data (Zuech 2000). It is concerned with developing techniques that can interpret image content (Castleman 1996). Computer vision has been defined as[8]:

---

[7] The terms “computer vision” “machine vision” “Image analysis” and “image understanding” are often used to denote the same thing (Snyder and Qi 2004, Shapiro and Stockman 2001, Castleman1996)

[8] This definition is by the Machine Vision Association of the Society of Manufacturing Engineers and the Automated Imaging Association.

> The use of devices for optical, non-contact sensing to automatically receive and interpret an image of a real scene in order to obtain information and /or control machines or processes (Zuech 2000,46).

Computer vision should not be confused with image processing. The purpose of image processing is to enhance the visual quality of the image. The output of image processing is an image, while, computer vision process analyze the image for its information content (Snyder and Qi 2004, Shapiro and Stockman 2001). Therefore, the goal of computer vision is to reach decisions about objects or scenes represented in the image (Shapiro and Stockman 2001). In the field of computer vision, it is extremely important to understand human vision for two main reasons. The first is because images are created for humans consumption, and secondly, it provides a principal guide for developing new algorithms (Shapiro and Stockman 2001, Jahne 1991)

2.2.4 Human Vision versus Computer Vision

The performance of the human visual system is much more sophisticated and powerful than any computer vision system (Zuech 2000, Jahne 1991). If compared to the eye-brain capacity, available computer vision systems are considered very primitive (Zuech 2000), and have lower dynamic range (Jahne 1991). Human vision, on the other hand, can handle a wider range of objects, and offer many advantages in terms of speed of interpretation, sophistication in dealing with lighting problems, the easy detection of minor variation of texture and objects (Jahne 1991), and the qualitative ability for object recognition (Jahne 1991).

Human vision is considered best for the qualitative interpretation of complex, unstructured scenes (Zuech 2000), but it has some deficiencies when dealing with structured scenes or complex geometry. These deficiencies are mostly related with its relative estimation of gray levels, distances, and areas (Jahne 1991). The key difference is related to the nature of interpretation; computer vision can be quantitative, absolute, and objective, while human vision is considered subjective, qualitative and comparative (Russ 1992, Zuech 2000). Moreover computer vision is more able to deal with two-

dimensional interpretation of well-defined features than three-dimensional, in which human vision is highly developed (Zuech 2000).

Computer-based image processing employs methods that do not seem to have counterparts in human vision (Russ 1992). For the task of recognizing uniform geometry, structured scenes, image shapes or distances, computer vision is considered ideal for performing quantitative measurements, and best for dealing with high image redundancy (Zuech 2000). The human eye is poor in judging shapes or brightness of different features in an image unless they are adjusted and rotated to the same position in order to perform a direct comparison between them (Russ 1992). The human vision can distinguish relative brightness differences, but not absolute. In a wide range of brightness, it can only resolve two percent of the relative differences (Jahne 1991). Moreover, human vision is considered poor in detecting different gray shades (Harrison and Jupp 1990, Jahne 1991), and does not recognize gradual or slight changes in color or intensity (Russ 1992). Human estimation of distances and areas can also be biased by image context (Jahne 1991). This is known as optical deception. Our human vision can be easily subject to optical illusions (Russ 1992), which can occur when dealing with geometric formations. These include parallel lines, vertical and horizontal line, length and angles, perspective effects, size, shape, etc, (Gibilisco 1990).

Another deficiency is related to the fact that, humans have a limited attention span when dealing with uniform redundancy. This makes them susceptible to distractions. They are also inconsistent in performing the same task; for example, people exhibit different sensitivities from time to time and from person to person (Zuech 2000).

2.2.5 Computer Vision System

Computer vision is the process where the computer recognizes the content of the image. It involves three main functions: image formation; image processing; and pattern recognition (Zuech 2000)

A) Image formation and representation

Machine vision is a term associated with the merging of sensing techniques and computer technologies (Zuech 2000). It starts when converting an optical scene to a digital image; machine vision systems operate on a projected image of a three-dimensional scene into a two-dimensional plane (Zuech 2000). The geometry of image formation can be understood as the projection of each point of the real scene through the lens center onto the image plain (Shapiro and Stockman 2001). It is a two dimensional array of quantized sample values in a rectangular form (Castleman 1996) The quantized information content of each pixel corresponds to its intensity, or image brightness (Zuech 2000, Shapiro and Stockman 2001). A digital image is" a two-dimensional image *I[i,j]* represented by a discrete Two-Dimensional array of intensity samples, each of which is represented using limited precision" (Shapiro and Stockman 2001, 29). This precision is called image resolution, which refers to the precision of the sensor in making image measurements. Image spatial resolution is the number of pixels available per unit area (Shapiro and Stockman 2001, Castleman 1996). The greater the spatial and color resolution, the more certain the fidelity of the image the system receives as the basis on which to make decisions (Zuech 2000).

B) Image Processing

Digital image processing is concerned with transformations that enhance images (Shapiro and Stockman 2001). It is defined as the process of "subjecting a numerical representation of an object to a series of operation in order to obtain a desired result" (Castleman 1996, 5). Real world objects yield exorbitant variations when sensed through images. The challenge for computer algorithms is to extract the essence of objects. The object appearance can be affected significantly due to its environmental factors such as lighting effects or the presence of other objects (Shapiro and Stockman 2001, 14). Image

processing may include enhancement[9], coding[10], compression[11], restoration[12] and reconstruction[13] (Snyder and Qi 2004, Bose 2004).

Image processing is used for two main purposes: improving the visual appearance of images for the human viewer and preparing the image for feature management and pattern interpretation (Russ 1992, Shapiro and Stockman 2001). Image processing is usually based on treating the image as a continuous function *F(x, y)* of two spatial parameters *x* and *y*, whereas in computer vision it is mainly treated as a discrete Two-Dimensional array *I[i, j]* of integer brightness samples (Shapiro and Stockman 2001, Castleman 1996).

C) Pattern Recognition

The field of artificial intelligence and computer vision is concerned with developing techniques for analyzing the content of images. Decision about object structure must often be made by integrating a variety of information from many pixels of the image (Shapiro and Stockman 2001). Pattern recognition is the main function of computer vision. This function is divided to two main components: measurement of features and pattern classification (Snyder and Qi 2004). Measurements of features focuses on processing image pixels or groups of pixels for the purpose of extracting measurements of image component or the entire image (Snyder and Qi 2004). Pattern classification is the process of making decisions about the measured features. Based on

---

[9] "Enhancement systems perform operations which make the image look better, as perceived by a human observer. Typical operations include contrast stretching (including functions like histogram equalization), brightness scaling, edge sharpening, etc" (Snyder and Qi 2004, 3).

[10] "Coding is the process of finding efficient and effective ways to represent the information in an image. These include quantization methods and redundancy removal. Coding may also include methods for making the representation robust to bit-errors which occur when the image is transmitted or stored" (Snyder and Qi 2004,3).

[11] "Compression includes many of the same techniques as coding, but with the specific objective of reducing the number of bits required to store and/or transmit the image" (Snyder and Qi 2004,3).

[12] "Image restoration refers to the process of restoring a corrupted image to its original state" (Bose 2004, 10).

[13] "Reconstruction usually refers to the process of constructing an image from several partial images" (Snyder and Qi 2004, 4).

our prior-knowledge and on the nature and characteristics of the measurement, we can make the decision to assign this measurement to a certain class (Snyder and Qi 2004).

Many different approaches for pattern recognition have been proposed, all of which follow the same process (Castleman 1996, Snyder and Qi 2004): Image Segmentation or Object Isolation and Noise Removal; Feature Extraction and Consistency Analysis; Classification and Matching. The organization of these tasks is demonstrated in Figure 2.12.

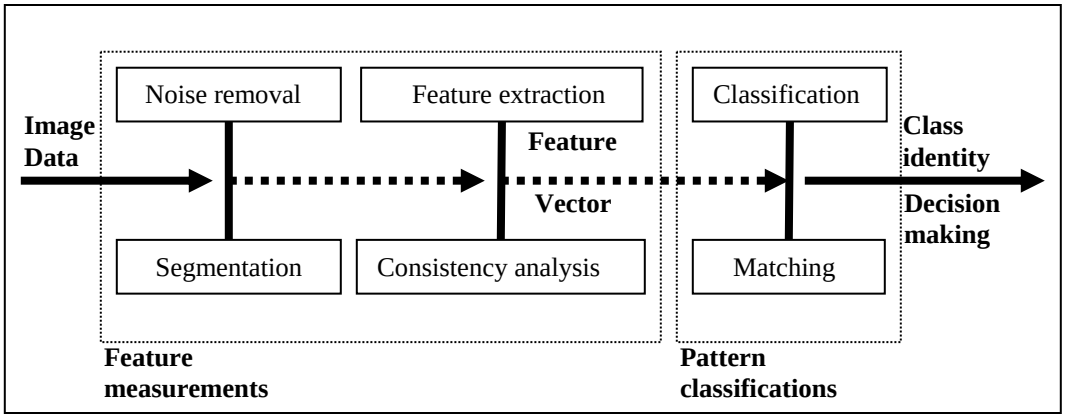


Figure 2.12 Organization of the Machine Vision System.

Pattern Recognition Tasks

1- Noise Removal

Images can have many distortions; they can be noisy, blurred, degraded or have other distortions. Before doing any kind of analysis these distortions need to be removed or reduced. The aim of this process is to prepare the image, so features can be derived more cleanly for segmentation. There are two general approaches to denoising: Image relaxation, which uses an iterative classification method (Snyder and Qi 2004), and morphological operations (Snyder and Qi 2004, Russ 1992), which includes binary morphology, Gray-scale morphology and the distance transform. The two approaches are used for image restoration, contrast improvement, and image sharpening (Bose 2004).

2- Segmentation

The purpose of image segmentation is to simplify image content (Shapiro and Stockman 2001) by dividing the image into regions that corresponds to image objects (Russ 1992). It is one of the basic functions of automatic object recognition (Shapiro and Stockman 2001, Snyder and Qi 2004, Jahne 1991). This operation subdivides the image into foreground and background based on some defined parameters, through which objects are isolated and prepared for feature extraction (Snyder and Qi 2004). It may be object-oriented, texture-oriented, or frequency spectrum-oriented (Castleman 1996). In general there are four concepts of segmentation: threshold/pixel-based methods, edge-based methods, region-based methods (Jahne 1991), and surface-based methods (Snyder and Qi 2004).

A) Threshold/Pixel-Based Methods

In this approach, a certain brightness threshold is chosen to segment objects from the background (Jahne 1991, Snyder and Qi 2004). All pixels above (or below) a specified threshold are assigned to the same region (Jahne 1991). This can be done either by using global thresholding by using a single threshold for the whole image (Castleman 1996), or adaptive by applying a more sensitive approach of choosing local thresholds for different image areas (Snyder and Qi 2004, Castleman 1996). These techniques in general produce closed regions. A threshold can be one dimensional, which means that only one band is used, two-dimensional with two bands, or multi-band, which uses red, green, and blue (Russ 1992).

B) Edge-Based Methods

The detection of object boundaries is one of the most fundamental problems in pattern recognition and computer vision (Castleman 1996, Gauch 1998). Recognizing shapes in images is not an easy task; nevertheless, if color contrast is strong, then lines edges and corners can be found using many edge/line-detecting techniques. Edges and lines are those places in an image that correspond to object boundaries. They are areas in

the image where the brightness changes suddenly or, in technical terms, where the derivatives of the intensity function have a large magnitude (Snyder and Qi 2004). Edge and line detection lies in the first phase of object isolation (segmentation), which represents an important prerequisite for many kinds of image understanding (Russ 1992).

An edge is a vector variable, which includes magnitude of the gradient, and a direction (Latecki 2003). Edges can be categorized as step, roof, and ramp (noisy) (Snyder and Qi 2004)**.** The main challenge for edge detection methods is the task of distinguishing edges from non-edge. Information in an image is found by looking at the relationship a pixel has with its neighborhoods. If a pixel's intensity value is similar to those around it, there is probably not an edge at that point. If a pixel's has neighbors with widely varying levels, it may present an edge point (Gauch 1998). Most of the Different approaches for edge\line detection share the same common ground (Latecki 2003).

Many are implemented with a convolution mask and based on discrete approximations to differential operators. Differential operations measure the rate of change in the image brightness function. Some operators return orientation information. Others only return information about the existence of an edge at each point. There are many ways to perform edge detection. However, there are two different general approaches. The first approach is the First Order Differential methods, which applies a first derivative operator and looks for local maximum and minimum and compares this with a threshold value (Little 2004). The Gradient-based edge detection is a first order differential method, which uses Gradient operators by applying simple first-order derivatives (e.g. the detectors of Sobel and Kirsch). (Snyder and Qi 2004, Argialas and Mavrantza 2004).

The second approach is the Higher Order Differential methods, which define pixel edges based on the zero crossing of the second derivative of the intensity function (Little 2004). In this method no threshold value is needed, but on the other hand, it is not always easy to estimate the second derivative (Little 2004). The Laplacian zero crossing is an important second derivative method (Jahne 1991, Gauch 1998).

C) Region-Based Methods

This method starts with different regions and splits or merges them based on a threshold value and connectivity (Jahne 1991). If a pixel is above a certain threshold and is adjacent to a pixel in region, then it is merged to that region (Snyder and Qi 2004, Castleman 1996). This includes two main approaches, Recursive region growing algorithm, and iterative approach to connected component analysis (Snyder and Qi 2004).

D) Surface-Based Methods

These methods operate on the gradient of the image and work to segment different surfaces in the image. There are two general approaches to handle this task. The first approach simply seeks surfaces that do not bend too quickly. This method applies algorithms to segment regions along lines of high surface curvature with smooth surfaces (Snyder and Qi 2004). The second approach assumes some surface equation and looks for all adjacent points that satisfy the equation to assign it to the same surface (Snyder and Qi 2004).

3- Feature Extraction

The basic approach to pattern recognition describes the recognized feature as a vector measurement (Russ 1992). A feature is a group of measurements, computed to quantifiably describe object characteristics (Castleman 1996). Line extracting represents one important challenge in the field of pattern recognition. This also can be cast as a vectorisation problem. Raster-to-vector conversion is a central task of pattern recognition (Tombre and Tabbone 2000, Russ 1992). A wide span of vectorization methods has been designed throughout the years. Nevertheless, they still have many problems related to accuracy, robustness, and stability (Tombre and Tabbone 2000). There are three main approaches for extracting lines; Skeleton/thinning-based-methods; Matching Opposite Contours; and Contextual Information-base methods (Tombre and Tabbone 2000).

The skeleton/thinning-based approach is the most common for vectorization. Its methods and techniques are the most related to this study.

Skeletonization and Thinning Methods

When dealing with line patterns or any structural grids, skeletonization methods provide an excellent step for pattern recognition (Tomita and Tsuji 1990). Skeletonization is a global space domain used for shape recognition, which plays an important role in most computer vision systems (Zou 2003, Tombre and Tabbone 2000). It was introduced as a method for describing the global properties of objects, and to reduce the size of the image into a more compact representation (Arcelli and Frucci 1992). The skeleton of a two-dimensional object is a transformation of image objects into a one-pixel width line. The produced skeleton represents the minimum information needed to preserve the general structural of image objects (Kasturi et al. 1992). This technique has been applied successfully in solving problems related to structural pattern recognition problems (Zou 2003).

There are two main types of skeletonization methods: Pixel-based methods, and non-pixel-based methods (Zou 2003, Zhong and Yan 1999). In pixel-based operation, all image pixels are used in the skeletonization process (Zou 2003), and often use thinning techniques to reduce all black components in a binary image to one-pixel width (Toriwaki and Mori 2001). Contour pixels are removed interactively, using 3x3 templates, where pixels are classified as removable or not by the template matching parameters (Zhong and Yan 1999). These methods are mostly concerned with line connectivity (Russ 1992, Chundy and Chundy 1997), end-point preservation (Chundy and Chundy 1997), simplicity, noise immunization, parallelism (Arcelli et al. 1994), and speed (Arcelli and Frucci 1992). Nevertheless, when dealing with line patterns, these methods produce many intersection distortions (Zhong and Yan 1999).

A none-pixel-based method uses only the contour pixels of the shape, which makes it much faster than the first group (Zou 2003). These methods include many algorithms such as: Medial Axis Transform (MAT) (Russ 1992, Castleman 1996), distance transformation (Tomita and Tsuji 1990), Hough Transform (Castleman 1996),

contour vectorization (Tombre and Tabbone 2000), Vornoi diagram (Snyder and Qi 2004), run-length coding (Hu and Li 1994), Discrete Local symmetries (Zou 2003). etc,.

Preserving the accurate geometry of line intersections is considered one of the main challenges facing skeletonization of line patterns (Hu and Li 1994). Most of the available techniques produce broken connectivity at intersection regions (Zhong amd Yan 1999). Nevertheless, some of the successes in this field were accomplished by using non-pixel-based methods (Zhong and Yan 1999). One of the most successful methods was done using the run-length encoding and vertorization (Zhong and Yan 1999). In 1994 Hu and Li presents an algorithm using Run-length coding for working with uniform width line patterns. The results were very good in preserving X-Crossings (Hu and Li 1994). Zhong and Yan (1999) modified this algorithm to capture the accurate geometry of non-uniform line width patterns by combining a run-length-wise method with thinning algorithms.

4- Consistency Analysis, Classification and Matching

Consistency analysis is concerned with the problem of interpreting the entire image from local measurements (Snyder and Qi 2004). One approach to the solution is using consistent labeling. This technique uses labeling to mark features that are consistent based on some defined parameters (Snyder and Qi 2004). Another approach to the solution is using parametric transformation, which assumes that the object we are looking for can be described by a mathematical expression (Snyder and Qi 2004).

Consistency analysis is used extensively for inferring structured features. The structural approach to feature extraction assumes that pattern structure is composed of spatial arrangements of repeated units. In this case structure extraction becomes the task of identifying feature locations and quantifying their spatial arrangements (Castleman 1996).

Classification and matching are the processes of making decisions about the measurement (Snyder and Qi 2004). A recognition system must contain some knowledge about the object it is trying to recognize. This knowledge might be

programmed in terms of feature descriptions, or a library of shapes (Shapiro and Stockman 2001). The output of the classification process is a decision about the class to which the recognized objects belong (Castleman 1996). The extracted objects are assigned to different groups based on their vector measurements (Castleman 1996, Russ 1992).

CHAPTER III
METHODS OF INQUIRY

3.1 Introduction

This study is seeking to build a visualization[14] tool that inputs digital images of deteriorated geometric patterns to predict the features that are not observed and output a complete reconstructed two-Dimension accurate measurable model. The problem in dealing with archaeological patterns is that in most cases data is not easily accessible, either they are worn-out or cannot be seen. To simulate what is missing data; research is faced with the challenge of how to scientifically arrive at the missing information by utilizing surviving fragments (Zemanek 2004). The simulation process in these cases depends primarily on finding the parameters necessary to predict information at other locations based on the relationships embedded in the existing data and in the prior-knowledge explaining these relations (Barcelo 2002). Nevertheless, there are no standard methods for defining the parameters of archaeological reconstruction or interpretation. The only restrictions when designing such a method are imposed by computers themselves, common sense and learned rules (Reilly 1991). In the reconstruction method, the aim is to build up from the fragmented data and from the historic and general knowledge, a model or an imagination of the reconstructed object (Zemanek 2004). Reconstruction in this context can be interpreted as:

> The integration of entities not present, totally or partially, in the archaeological scenario, moving beyond the archaeological reality to produce a possible scenario, following a deductive process (Viti 2004, 525).

Reconstructions can be understood as the visual interpretation of results (Jablonka 2004). The process starts from an assumed objective data, then process the information based on the estimations of certain parameters, finally arrive at interpretations that will

[14] Visualization within an archaeological context has been described as:" the use of mechanic devices for translating perceptual inputs into geometric models interpreting luminance contrasts. Its goal is to explain spatial regularity between archaeological inputs" (Barcelo 2002, 21).

define the integrations (Viti 2004). This process involves creating a hypothetical model, which will be fitted to the partial data, and then use the model to fill in the missing gaps (Barcelo 2002). Building this model can be understood as a synthesis of four main processes: data acquisition, pre-processing, pattern recognition and visualization (Castleman 1996, Zuech 2000, Barcelo 2002). This research is only concerned with the last two processes. The integration between computer graphics and computer vision proved to be even more successful in addressing practical problems (Shapiro and Stockman 2001).

Depending on the nature of prior knowledge, virtual reconstruction methods have two approaches: qualitative and a quantitative. The qualitative approach is used when the only things known are analogies of the project and similar case studies. In this case we must use qualitative models to simulate the missing data. The knowledge used in building the model depends mostly on analogy and is completely subjective (Barcelo 2002).

The quantitative approach is used if prior knowledge can be employed to produce a geometric model or mathematical guidelines that can simulate the missing information from the partial data. In this case the reconstruction is a mere task of mathematical calculations (Reilly 1991). The three preconditions for quantitative archaeological reconstruction are extrapolation, interpolation and smoothing (Zemanek 2004). The extrapolated or interpolated environments are tied to archaeological interpretation through highlighting links and relations between different elements (Earl and Wheatley 2002). Extrapolation is the virtual extensions of the object, interpolation is the virtual addition of the missing parts and smoothing is the virtual completion of the damaged parts (Zemanek 2004).

The research at hand encapsulates the two approaches within its methodology. Qualitative approach is evident in the process of program design, algorithm selection, and evaluation. Quantitative approach is manifested through using mathematical knowledge of pattern generation to interpret available data and to simulate the rest based on it. The reconstruction process is designed to measure certain pattern variables. These variables are then used within a mathematical formula to produce the whole system. The research

tests the validity of incorporating prior-knowledge of pattern rules with the adaptation of existing pattern recognition techniques for the purpose of producing structurally accurate virtual reconstruction. The reconstruction of pattern structures is a generalization of the observed data by the use of geometric shape models (Barcelo 2002). The key aspect of this model is related to its spatial nature, which is considered as a visual representation reflecting a spatial decomposition of reality in geometric units (Barcelo 2002)
For the purpose of reconstructing geometric patterns, this synthesis implies the definition of a formal mathematical system and uses it within a logical framework to design a tool that has analytical power over all instances of pattern combination. Although this research is generally based on deductive logic, it still uses induction to generalize output beyond the observed instances. The reconstruction process involves "*induction, deduction and analogy*" (Barcelo 2002, 21).

3.2 Research Design

The project described here explores the possibilities of incorporating the mathematical structure of Islamic patterns within the structure of digital images to develop a visual tool capable of interpreting partial evidence to simulate the missing parts. The definition implies two main tasks. The first task includes the development of the visual tool (computer program) and the second task includes testing and evaluating its performance. These tasks represent the first phases of developing professional software and can be situated under the umbrella of software engineering[15] methods. Accordingly, it follows an empirical paradigm in developing its visual tool and uses experimentation as its main strategy.

The main reason for using experimentation in this study is the ability to maintain one or more variables and control all other variables at fixed levels, which enable testing, measuring and identification of different relationships (Wohlin et al. 2000). Experiments are also employed to manipulate variables directly, precisely and systematically. Other important advantages include the possibility of replication (Wohlin

[15] The IEEE defines software engineering as:" the application of a systematic, disciplined, quantifiable approach to develop, operate and maintain of software" (Wohlin et al. 2000, 1)

et al. 2000). Within this area there are two types of experiments: true experiments and quasi-experiments. True experiments use full randomization while the selection of samples in quasi-experiments does not (Castle 1992). For conducting the above tasks included in this study; quasi-experimentation methods are employed. These methods are important and provide valuable results. Moreover, there are two types of research paradigms that have different approaches to empirical studies: Qualitative & quantitative. Quantitative strategies are appropriate for testing the effect of a treatment, while qualitative studies are appropriate to find out why the results are as they are (Wohlin et al. 2000).

In the first task of developing the software, both quantitative and qualitative approaches are used. A qualitative approach is used to design the program, while a quantitative approach is used through applying algorithms to pattern measurements. The later is mainly concerned with quantifying a relationship or to compare two or more groups (Creswell 94). The aim is to identify a cause-effect relationship. In the second task, again both quantitative and qualitative approaches are used. A quantitative approach is used through setting up the controlled experiments to test the software (Wohlin et al. 2000), while a qualitative approach is used to evaluate the results. The two approaches should be regarded as complementary rather than competitive (Castle 1992). The goal of the experiment is to analyze the output reconstructed patterns for the purpose of evaluating the digital tool in respect to reliability and structural accuracy, from the point of view of the researcher in the context of historic preservation. Included in the definition are two distinctive aspects of research (1) quantitative (or metric), which involves accurate dimensional measurements to obtain direct information related to size and shape of patterns, and testing the process. (2) qualitative (or interpretive), which deals with the recognition, interpretation and evaluation of results. Quantitative investigation is appropriate when testing the effect of some instrument or activity. It also promotes comparisons (Wohlin et al. 2000). Qualitative methods, on the other hand can be employed for analysis and evaluation (Castle 1992). It is concerned with discovering causes noticed by the subject and understanding their view of the problem at

hand. The subject is the person, who is taking part in an experiment in order to evaluate an object.

3.3 Program Development

The development of this visualization tool involves designing; testing and evaluating a computer program through employing an experimental research method (Groat and Wang 2002). At the first stage, the development process started with a very controlled pilot study as a way for testing available digital tools, techniques, methodologies and to test the validity of using such tools in accomplishing the tasks required for this research. It also provided me with a guide for roughly defining the key variables, constrains, limitations and define research boundaries. This stage set the platform for the development of the main tool. In this next stage, I employed experimental research methods to examine how to design, test and evaluate such a tool. Designing the tool itself also involved four main tasks:

A) Program Design

Program structure: This included planning the outline structure of the program, relations and proposing a flow chart for all the tasks involved. It also included defining techniques, approaches and conceptual framework for accomplishing technical tasks.

Interface design: This included designing the interface of the program, defining working space, menus, colors, objects and boundaries.

B) Implementation

Coding: Writing the code for the implementation based on the design.

Compiling: Compiling the program until no additional faults can be found.

C) Testing Code

Execution and Testing: This involved executing the program and testing it on sample data, and correcting it until all faults that have been found are removed.

D) Analysis and Evaluation

This included evaluating outcomes, performing analysis on collected data, and validating the results.

The process that is used to organize these four tasks is depicted in Figure 3.1, which shows that all the tasks complement each other. The designing process was an ongoing interaction between the four tasks, and the program was modified and updated based on the operation feedback. The researcher defined the end of this process when all the tasks required by the program were satisfied. Moreover, the mathematical-formal system embedded within the design logic was tested continually with each use (Groat and Wang 2002).

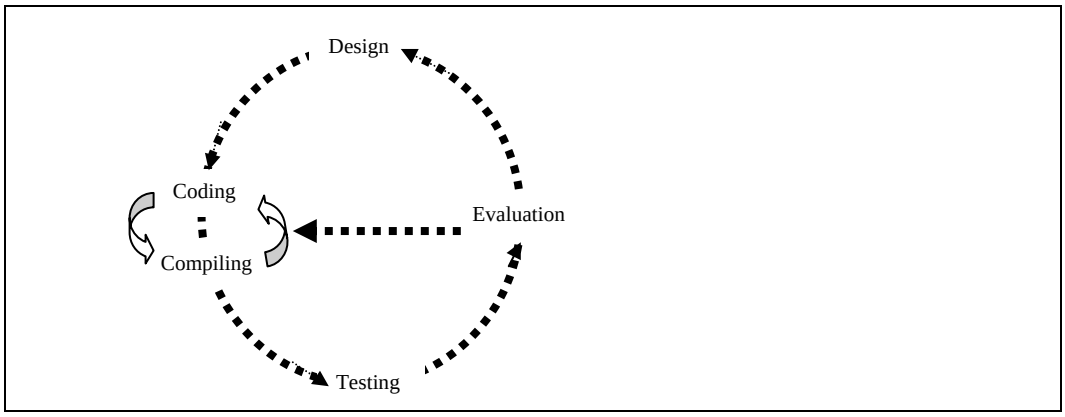


Figure 3.1 The Process for Organizing the Main Tasks Related to Designing Computer Programs.

3.4 Pattern Analyses

An Islamic Geometric Pattern is defined mathematically as "a planar arrangement of line segments that together delineate copies of a small number of different shapes" (Kaplan 2000). Although each pattern has its own distinct geometrical design, the vast varieties of ornamental compositions are based on a simple constitutive geometry and are generated from a limited number of simple base grids (Gonzalez

2001). To understand these patterns mathematically, it is important to study their abstractions; through which all rendering effects and colors are discarded and only the basic line structure is kept (Kaplan and Salesin 2004). Therefore, the analysis of surviving historical Islamic patterns was the first step in conducting this research.

Pattern analysis was done manually and with the aid of AutoCAD tools. This analysis aimed at understanding the mathematical base of pattern geometry, and to define the formal rules of pattern generation, which was used later for pattern reconstruction. Pattern analysis was done through deconstructing the geometric formations to their elementary components, and then investigating the rules that organize the relations between geometric primitives (points and lines). Such rules were translated into mathematical structure, which then programmed into the computer to help interpret digital data representing the patterns

To control research variables, the scope of the study was limited to the group of geometric patterns generated from one specific basic grid, and was limited to two-Dimensional space. The fewer dependent variables the system has, the clearer the resulting model is (Barcelo 2002).

3.5 Target Population

The general goal for this research is to test an idea that can be applied to all types of Islamic Geometric Pattern. Therefore, the general targeted population includes all categories of Islamic Geometric Patterns. Nevertheless, for the purpose of conducting this study, the experiment specifically targets one group of these patterns, with the assumption that it provides a true representative sample of the general populations. This specific target population includes all straight-lines patterns generated based on the octagon-square basic grid. Patterns generated based on this basic grid were used extensively all over the Islamic world. They represent nearly twenty percent of all pattern combinations.

3.6 Sample Procedure

The number of patterns that can be generated based on the octagon-square basic grid can be extremely large, and is related to the number of different probabilities of pattern combinations. The degree of complexity of these combinations is related to the number of simple stars used to generate the final repeat star. For the purpose of this study, it was important to sample over different complexity levels. Therefore, this research used a quota sampling technique, in which samples were chosen from predefined categories of the targeted population. To cover all levels of complexity, the population was divided into three main categories: simple, medium, and complex. The researcher used patterns' rules to generate an equal number of samples for the three different categories of complexity. In this case, the sampling of the population was a non-probability sample, which means that the probability of selecting each subject is unknown (Wohlin et al. 2000). On the other hand, a simple random sampling technique was used when sampling from each category. This later technique is called a probability sample (Bowen and Weisberg 1980).

The researcher was the only subject conducting the experiment, and she generated thirty different pattern designs; ten designs of each different category. The choice of designs depended on using a systematic method of convenience sampling (Wohlin et al. 2000). All patterns had the same number of star units.

3.7 Sample Treatment

Patterns were generated using AutoCAD tools and were constructed based on a well-established systematic method (Kritchlow 1976, Jones 1978, El-Said 1993). Line patterns were generated with a fixed line width and were printed using black ink on (8.5 x 11 in) white sheets. For each pattern category, the researcher produced ten different pattern structures. Pattern's size, line width and arrangements were kept constant for all samples.

Digital images with a fixed medium resolution (1600x 1200 pixels) were taken for the chosen samples using an off-the-shelf digital camera (Canon, PowerShot A70).

Digital images were manipulated through Photoshop or Gimp in order to produce different resolutions and deterioration structures. Depending on the location of surviving data, sampled patterns were treated to produce three different deterioration structures. Finally, the resulting images were then saved using the PPM raster based image file format, which is used as an input to the developed visual tool. The program outputs an EPS vector based image file format, which is then used for analysis.

3.8 Instrumentation

To perform the tasks proposed by this study, several graphical tools, and measuring instruments were required. For the first part, pattern analysis was performed manually and by using the AutoCAD program. Manual analysis included the use of a compass, straight edge, tracing paper, and a pencil.

To generate the experimental samples, AutoCAD 2005 was used to construct the original complete patterns. Patterns were printed using a laser printer on 8-inch x11-inch white sheets. Then digitized using an off-the-shelf camera. The camera was fixed on a tripod, with a fixed distance (2ft) from the target. Digital images were then manipulated graphically to create the deterioration effect using the Photoshop and Gimp programs.

Program design was performed using computers supporting C, C++, OpenGL and the OpenGL interface API GLUT. The designed program itself was the main instrument, and was used to produce the experimental documentation and reconstruction of the deteriorated image.

3.9 Pilot Testing

A pilot study was done to explore and validate the use of some techniques in performing the key research tasks. The aim of the pilot study was to accomplish three main tasks: test the validity of using available algorithms for geometric pattern recognition; modifying existing algorithms to match certain tasks in the research and

proposing new algorithms with built-in pattern rules. The final results defined what will be used to generate the missing patterns.

In this process several separate programs were written. Here I present an overview of the general algorithms used to accomplish the main tasks:

1- Defining the repeated unit.

Defining the repeated unit is a major challenge. This includes locating the two centers governing the repeat unit, and defining its area boundary.

The first algorithms were written to automatically locate all centers by scanning the whole image. Although, they were successful in performing the task, the problem was with the execution speed. To overcome this problem, the decision was made to involve a user input in order to make it fast and more efficient. It is now the responsibility of the user to pick a point within the central region of each of the two centers. The program will process the input points to locate the exact centers. Next, it will define the repeat unit and generates the main axes. Results of the modified program showed that the algorithms were faster and were able to accomplish the three tasks. Figure 3.2 demonstrates the output result.

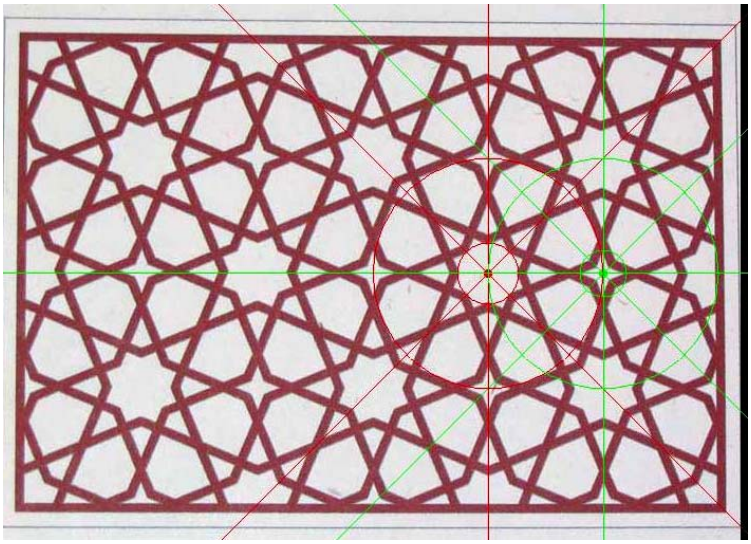


Figure 3.2 The Visual Results of Defining the Repeat Unit Through User Input.

2- Finding intersection points within straight-line geometry.

Defining the intersection points was the most challenging task. Literature exploring this type of research usually applies thinning or skeletonization methods (Zou 2003, Tombre and Tabbone 2000). Many techniques are available to deal with line recognition. However, most of them have problems with intersection distortions (Zhong and Yan 1999). The only promising techniques were related to run-length skeleton-based methods. These methods were designed to deal with preserving line intersection (Hu and Li 1994, Zhong and Yan 1999). As a result, this program adopted some of the algorithms and modified them to suit its purpose. These algorithms addressed three main tasks:

1- Image segmentation.

The program used thresholding segmentation technique (Jahne 1991, Snyder and Qi 2004) to segment pattern lines from the background.

2- Applying Run-length-encoding algorithm.

This includes two tasks: saving image in a run-length- format (Hu and Li 1994, Zhong and Yan 1999), and defining the conversion points of each intersection (Hu and Li 1994) (Figure 3.3). A conversion point is the location where two lines join. This point is demonstrated in Figure 3.3 as *Cpoint* at position *xj, yi.* Conversion points are found by checking the current run-length with the previous run-length on the previous scan line for any overlap. If the current run length overlaps with two run-lengths on the previous scan line, then we define a conversion point at the midpoint of the current run-length.

3- Defining the intersection points.

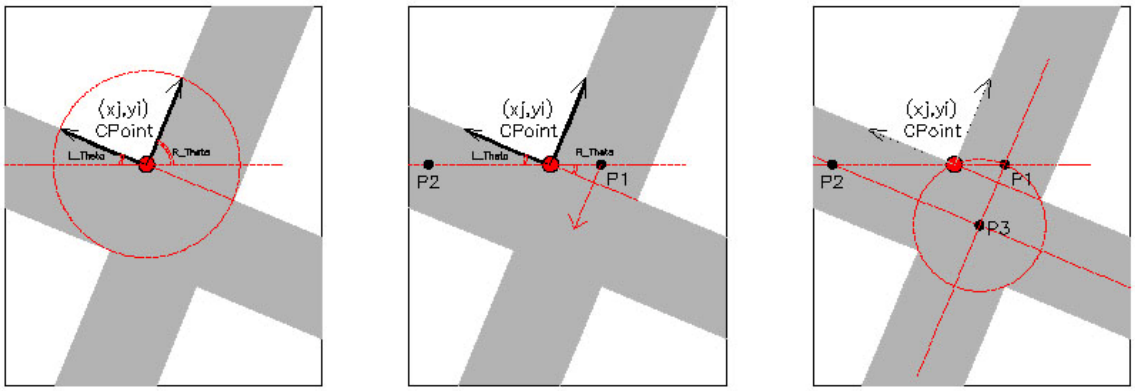
The conceptual process of calculating the intersection point is explained in Figure 3.3. The program uses the conversion point *Cpoint* at position (*xj*,*yi*) as the starting point. The algorithms were designed to find the angles of the two intersecting lines. As demonstrated in Figure 3.3.a, the angles *L_Theta* and *R_Theta* are found by drawing a circle with the conversion point at its center, and then scanning its circumference to define the location where the pixel's color changes. A line is drawn

from this location to the conversion point. Finally the angle is calculated using the two points vector.

Figure 3.3.b, demonstrates the second step of defining the two points *P1* and *P2*, which locates the center axes of each intersecting line. These points are located at the middle pixel of each of the two run-lengths on the previous scan-line that overlaps with the current run-length. The number of pixels in each run-length should be saved while scanning the image.

Figure 3.3.c, demonstrates the estimation of the final intersection point. For each point (*P1* and *P2*), a line parallel to the line edge is drawn and extended to meet the other line. This is done using the angle found in the first step. Finally, the central point *P3* is defined by the location where the two lines intersect.

The final result from applying this technique to image data is shown in Figure 3.4. Based on the results, I was able to define the most applicable techniques, algorithms, and structures. Moreover, it defined constrains, limitations and research boundaries.



a **b** **c**

Figure 3.3 The Conceptual Process of Calculating the Center Point of an Intersection.

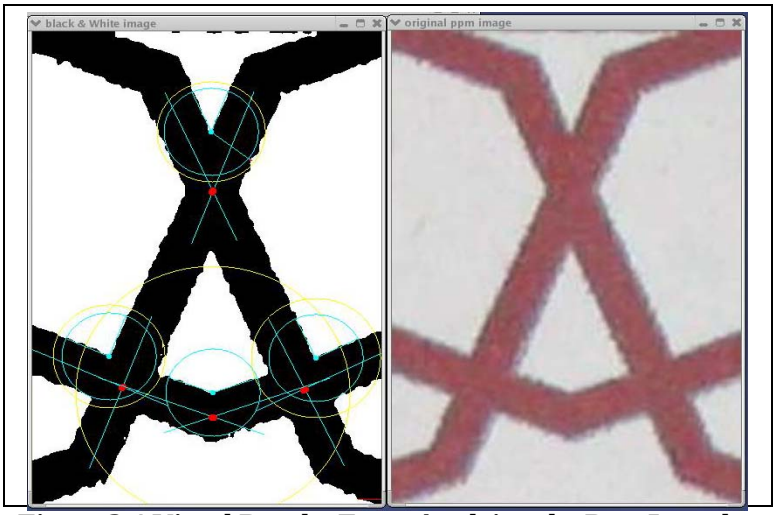


Figure 3.4 Visual Results From Applying the Run-Length Algorithm to an Image Containing Multiple Intersections.

3.10 Assumptions of the Study

1- Octagon-based patterns are assumed to be a true representative of all Islamic pattern types. Results drawn from testing them can then be generalized to include the general population of all Islamic Geometric patterns.

2- In the operational phase of the experiment, the operator is assumed to be familiar with the required tasks and to have some basic knowledge about the underlying structure of Islamic patterns. For the visual tool to work properly, it is important to input the right information. It is also assumed that if the operator makes any mistake in the input information, the result will also be invalid.

3- In order for the program to work properly we assume certain parameters. The research assumes that the deteriorated pattern has enough information to generate the whole reconstruction. The minimum information needed for the pattern to be reconstructed is the repeat unit, which includes two main centers.

4- The program assumes a clean image as its input. Images do not need to be processed to enhance their quality. This research is not designed to handle images of real life objects. Rather it deals with a very controlled experiment, which tests a clean image.

5- This research assumes two-dimensional straight-line patterns. The study does not include any three-dimensional patterns, or patterns with curved lines.

3.11 Limitations of the Study

1- The study tests only one type of Islamic patterns. This includes all patterns generated based on an octagon-basic grid. Moreover, it limits itself to deal with only one rendering treatment. Historically, these patterns were never merely drawn as lines. Often, the lines are thickened when incorporated into different material and sometimes broken up to suggest an interlacing pattern. This research is not designed to deal with all pattern-rendering treatments. It test its hypothesis with only one treatment; the outline style, in which pattern lines are thickened to add weight and character to the lines of the plain pattern. Moreover, the difficulty with experiments is that they are weak on representation, and may include sampling errors (Bowen and Weisberg 1980).

2- Another limitation is related to the need for user input. The process of reconstructing these patterns is not completely automated. The operator needs to interactively define some parameters, before the program can generate any valid results. This process requires the user to have some basic knowledge about pattern structure and about operating the program. Nevertheless, any amateur operator can be directed easily to do so by following some simple instructions.

3- Another limitation is imbedded within the use of qualitative evaluation. Qualitative research begins with accepting that there is a range of different ways of interpretation (Wohlin et al. 2000). Interpretation inevitably brings with it a loss of objectivity (Bowkett 2001). Methodological limitation is mainly based on the methods of interpretation but not on the process to gain the primary facts or the archaeological record (Doneus and Neubauer 2004).

4- This research limits itself to dealing with a very controlled study. It is not designed to handle problems of real life objects. It provides a conceptual solution that can be

used as a first step in addressing the practical task of reconstructing images of real Islamic patterns.

3.12 Validity and Reliability

This evaluation is concerned with the validity of results for the general population of Islamic patterns. First, the result should be valid for the specific population form-which the sample was drawn, which in this case refers to octagon-based patterns. Secondly, it is important to be able to generalize the results to a larger population. In this research some results will be drawn about other categories of patterns. In experimental research there are four threats to validity (Wohlin et al. 2000):

1- Conclusion validity. The relationship between the treatment and the outcome should be clear.
2- Internal validity. The relationship between the treatment and the outcome should be casual, and we have control over all independent variables.
3- Construct validity. This has to do with the relation between theory and observation, and does it observation satisfy the claim put by the theory.
4- External validity. Can the results be generalized outside the scope of this study?

For applied research such as this one, it is more important to evaluate the internal and external validity (Wohlin et al. 2000).

Reliability evaluation is done through testing the precision of reconstructing different patterns. This evaluation is concerned about the repeatability of measurements (Lichten 1999). The tool is tested on complete patterns repeatedly with different user input each time. The purpose of the replication is to show that the results from the experiment are consistent and valid for the whole population.

CHAPTER IV
PROGRAM DESIGN AND STRUCTURE

4.1 Introduction

The objective of this study is to develop, test and evaluate a computer program that is able to digitally capture the right geometry of an incomplete Islamic Geometric Pattern. The aim of this process is to produce a complete and structurally accurate two-dimensional reconstructed model.

This chapter addresses the development process of the program, which includes two main phases: first, the conceptual design phase, which establishes the basic functions, operational characteristics and the overall organization of tasks. This phase also includes designing the visual interface, constructing tasks' flow chart, and investigating algorithmic approaches to identify workable techniques. Next is the implementation phase, which includes coding, testing, revising, and evaluating results. These four tasks must work simultaneously to reach the final design. They involve writing algorithms, correcting faults, testing using sample data and evaluating the results.

4.2 Conceptual Phase

4.2.1 Context Selection

Since many of the tasks involved developing specialized algorithms and making use of graphics libraries, implementation required the researcher to have a strong mathematical background, knowledge about digital image tools and pattern recognition techniques, programming skills (C, C++ languages), analytical skills, and general photographic skills. The study was conducted in the VIZ lab, at the College of Architecture, Texas A&M University, USA. Work was done on computers supporting C, C++, OpenGL and the OpenGL interface API GLUT.

4.2.2 Program Structure and Flowchart

A program is an extremely detailed set of operations to be followed by the computer. Program structure and organization represent the planning phase of the whole process. A general flowchart shown in Figure 4.1 depicts the main operations, organization of the different tasks, and sequence. The program starts by inputting a PPM raster based image file format. The program reads the image and saves two copies in two different two-dimensional arrays. The program then displays one image on the screen with a menu that has a slide bar for controlling segmentation threshold. Based on user's selection, the program processes the second saved copy of the image to convert it to black and white image. The program interactively updates the image based on the user input and displays it on screen. After reaching the best visual result of image contrast, the user is required to define two center points of the repeated unit. This is done by clicking anywhere inside a central region and the program will process it to calculate the exact center. Based on these two centers, the program defines the boundaries of the repeat unit. Based on the measurements of the repeated unit, the program calculates the estimated position for all other center points in the pattern. These center points are then saved in a two-dimensional matrix for future processing. Next, the program scans and processes the repeated unit to locate all intersection points. Based on these intersection points, it produces one complete star unit using the principals of symmetry. Finally it copies this unit to all other center points locations, which were saved earlier to produce the whole pattern. The program then displays the results on the screen and saves a copy to an EPS vector based image file format.

Workable techniques were identified earlier in the course of this study while conducting the pilot study. The program was designed to operate in an interactive mode, which permits direct communication with the computer and instant reply. The process is a semi-automatic one, which requires user input. This process represents one step before complete automation. One purpose of this research is to define ideal parameters that can be used later for full automation. Moreover, user selection is used as a way to speed up

the process and to save redundant work, which might consume time, effort and valuable space.

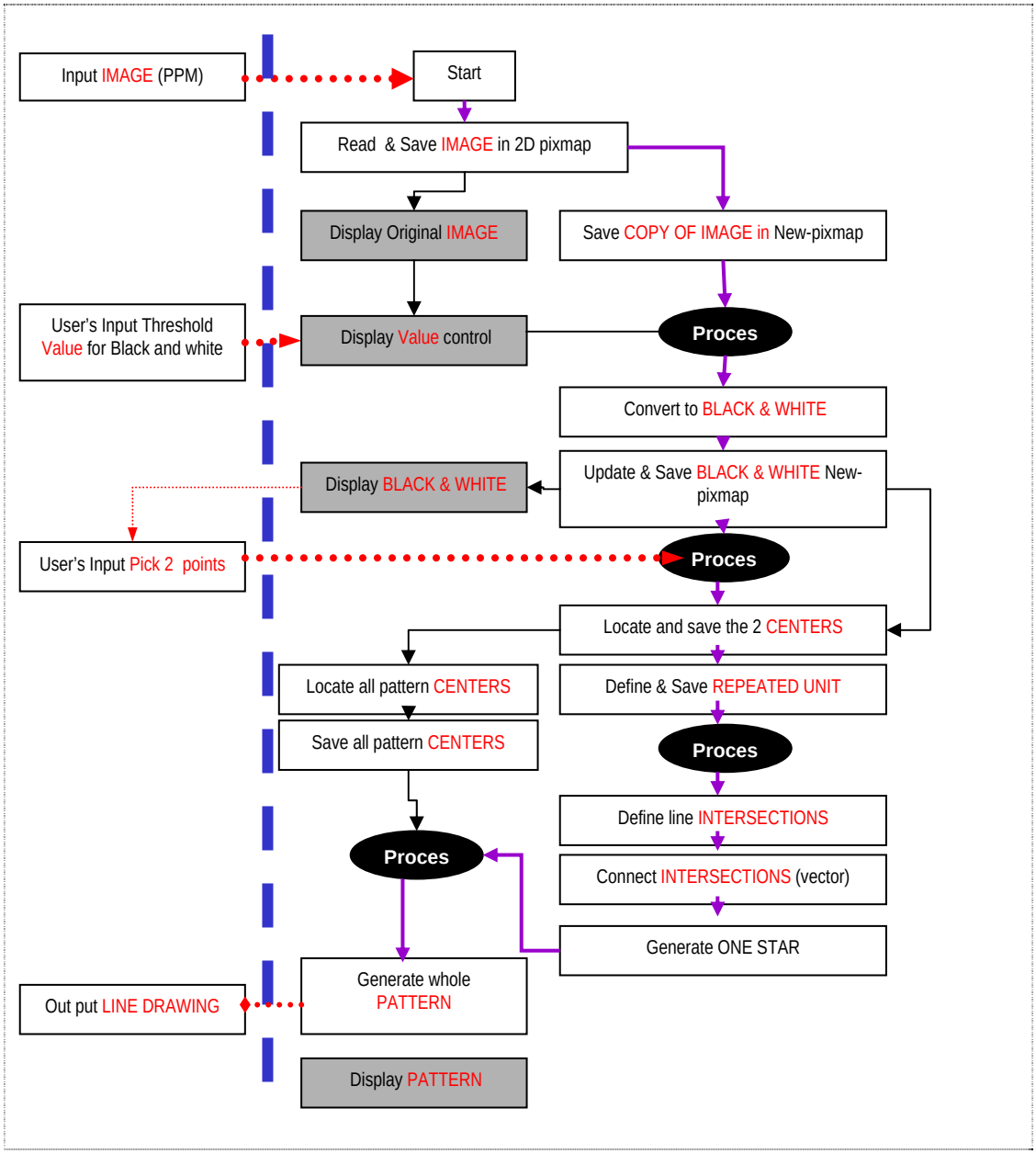


Figure 4.1 A General Flowchart Depicting the Main Operations and Organization of the Different Tasks to be Executed by the Program.

4.2.3 User Interface

Good User Interface Design requires a systematic approach to the design process (Castleman 1996). This includes designing the visual appearance, defining working space, menus, colors, objects and boundaries. In this process, it is important to develop a Human-Computer Interaction (HCI) that permits naive users to operate easily, and provide data about what works as anticipated and what does not work. For the purpose of conducting this research, the user interface is designed to give the user control over the main functions of the program. The tasks performed by the user are very simple, easy to learn, and do not require any specialized skills or experience. Nevertheless, to be able to properly perform the selection tasks required by the program, the user is assumed to have some basic knowledge about Islamic geometric patterns. The interface design for this program is shown in Figure 4.2. It includes two main parts, the display window, and the menu toolbar at the bottom part of the screen.

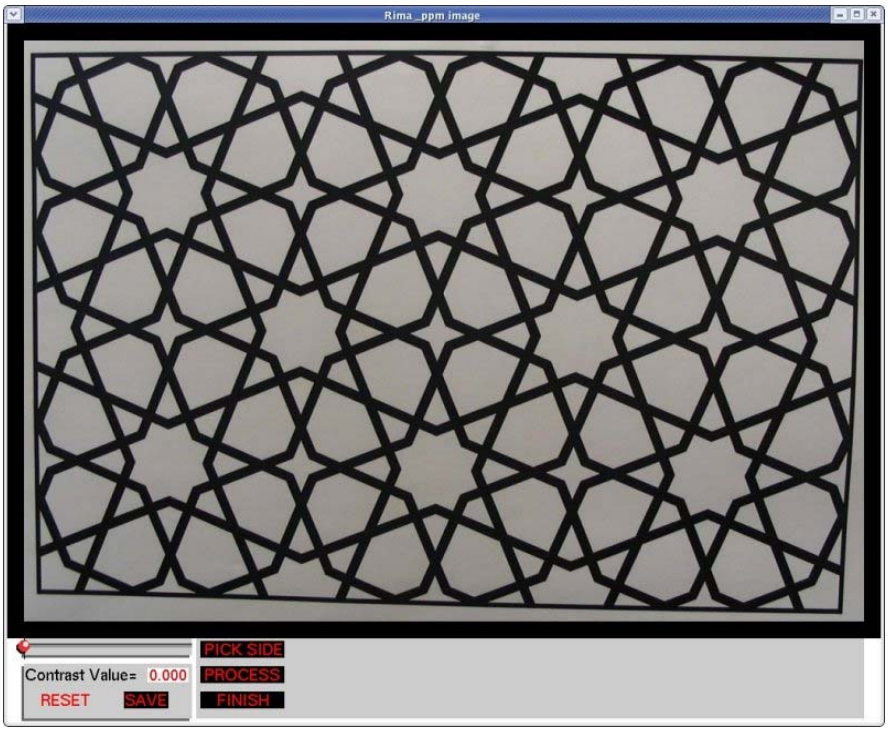


Figure 4.2. The Program's User Interface.

The different menu items shown in Figure 4.3 control three main program tasks: The contrast value slide bar allows the user to control the contrast value of the image. The program updates the image based on that value and outputs a black and white image. The user can update this bar interactively through out the process. The three buttons on the right side of the menu represent three main pattern recognition functions in the program. They allow the user to make a selection, process that selection, and then finish by generating the final results. These functions will be explained in detail later in this chapter. Finally the RESET and SAVE buttons allow the user to save bitmap images or reset the image to its original state.

Figure 4.3. The Program's Main Menu.

4.3 Program Implementation and Algorithm

A digital image can be defined as a discrete representation of a continuous real image, in which color information is stored within the computer structure as numeric codes for each image sample (pixel) (House 2002). Work is done within RGB color and HSV color systems. RGB organizes color into three components: Red, Green, and Blue, And HSV organizes color into three components: Hue, Saturation and Value. Color information will be analyzed in relation to pixels location (x, y) in reference to image space.

4.3.1 Program Input and Command Line Parameters.

The program is designed to input a PPM Image file format. It reads the image, saves it within a two-dimensional array and displays it on the screen. The algorithms

treats the image as a discrete 2D array *I[m,n]* of integer pixels (Shapiro and Stockman, 2001, Castleman 1996), where *m* is the number of rows or scan-lines and *n* is the number of pixels per scan-line.

Each pixel contains four color channels; red, green, blue, and alpha. Figure 4.4 shows the main definition and algorithms used for saving the different color channels.

```
unsigned int  **pixmap, **bitmap;

/*initiate 2D arrays for red, green, blue, and alpha */
rows = ppm.height;
cols = ppm.width;

pixmap = (unsigned int**)malloc (rows*sizeof(int*));
for(x=0;x<rows; x++)
       {pixmap[x]   =(unsigned int*)malloc(cols*sizeof(int));}

/*Reading channels information and saving it into a 2D array pixmap */
       for(i=rows-1 ;i>=0;i--)
              for(j=0;j<cols;j++)
              {
              red= fgetc(input);
              green= fgetc(input);
              blue= fgetc(input);
              alpha= 255;
              pixel =  alpha << 24 | blue  << 16 | green  << 8 | red;
              pixmap[i][j] = pixel;
              }
```

Figure 4.4 Algorithms of Saving Image Colors into Two-Dimensional Array.

Command line parameters needed to execute the program are shown in Figure 4.5. It includes program name, input image in PPM file format, output image name for saving a black and white image, and an EPS image name for saving the final output.

```
/u/rajlouni/research/final% prepare_image complex_1_1200_det.ppm out.ppm out.eps
```

Figure 4.5 Command Line Parameters.

4.3.2 Image Processing

Image processing is done to improve image quality for extracting information and preparing the image for analysis (Shapiro and Stockman 2001). The aim of this stage is to improve the desired features of the visualization and to suppress those, which

seem less desirable (Aspinall and Haigh 1999). In this process, two main standard techniques of image processing for preparing the image for analysis are applied.

2- Color segmentation and enhancing colors contrast.
3- Converting the image to a black and white (Wu and Yu 2003).

These tasks are done interactively controlled by the user by adjusting the slide bar in the main menu. The user selection triggers three main functions in the program: **writeValue(),** which saves contrast value and displays it on the screen, **Bitmap(value),** which uses the saved value to define the threshold for black and white conversion (Figure 4.6), and **display_Bitmap(),** which displays the resulting image. Figure 4.6 shows the algorithms for converting the image to black and white based on the user selection threshold value.

```
void Bitmap(float valueControl){
for(i=rows-1 ;i>=0;i--)
        for(j=0;j< cols;j++)
        {
        pixel = pixmap[i][j];
        red= pixel;
        green= (pixel >> 8) & 0xff;
        blue= (pixel >> 16) & 0xff;

        RGBtoHSV( red,  green,  blue,  hue,  sat, val);

        if(val < valueControl){red=0;green=0;blue=0; }
        else {red=255;green=255;blue=255; }
        pixel =  alpha << 24 | blue  << 16 | green  << 8 | red;
        bitmap[i][j]= pixel;
        }
}
```

Figure 4.6 Algorithms of Converting the Image to Black and White.

Within the Bitmap function, the program converts the RGB color primary values from the scale of 0 – 255 to HSV color system; Hue on scale 0-360, Saturation and Value on scale 0-1. The program uses threshold/pixel-based segmentation (Jahne 1991, Snyder and Qi 2004) through using global thresholding. All pixels above a specified threshold are assigned to the same region (Jahne 1991). Figure 4.7 shows some contrast

threshold values and the corresponding images. The upper left hand figure shows the initial state of the image before processing. The upper right hand figure shows an image with a contrast threshold value of 0.460. This value seems to be perfect for producing a balanced black and white. The lower left hand figure shows an image with a very low contrast threshold, which resulted in some missing data. The lower right hand figure shows an image with a high contrast threshold value, which produced too many noise at one edge of the image.

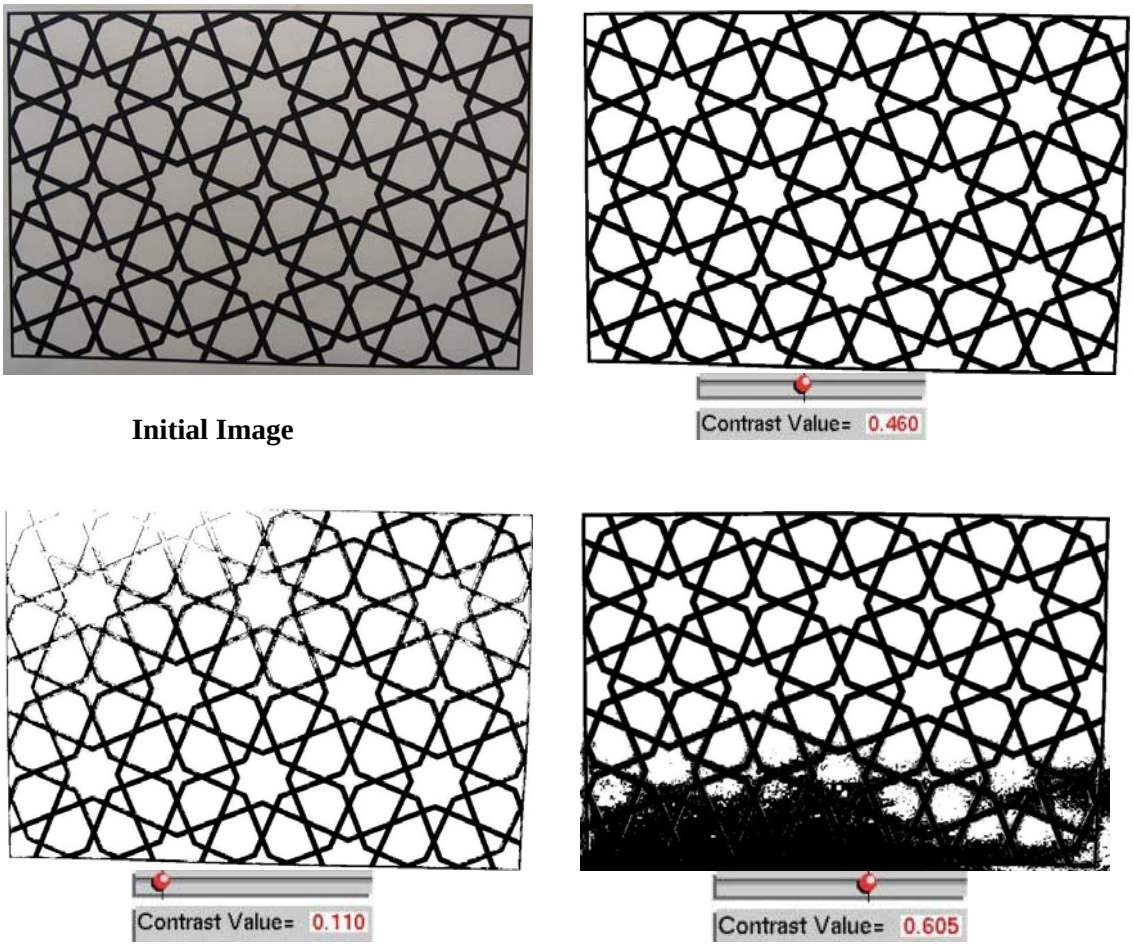


Figure 4.7 Different Contrast Threshold Values and the Corresponding Images.

4.3.2 Pattern Recognition and Interpretation

Information that is necessary to interpret an image is not represented in single pixels but in meaningful image objects and the relations between them. If we can specify a degree of spatial regularity in a region, we can then use it to reproduce the whole system (Barcelo 2002). By operating on the relations between networked objects, it is possible to classify local context information. This context information can be used together with form of image objects to improve understanding. Reading and interpreting data requires discovering object structure within the image pixmap. Analyzing such data will be based on understanding color information, which is captured through the locations of discrete image elements. Approximation will be used to read digital image data and then interpret based on pattern rules. This means that knowledge of pattern structure will be incorporated into a formal mathematical logic to identify pattern units through reading the digital image information. This phase involves two main steps:

1- Identifying the Repeated Pattern and Generating the Basic Grid

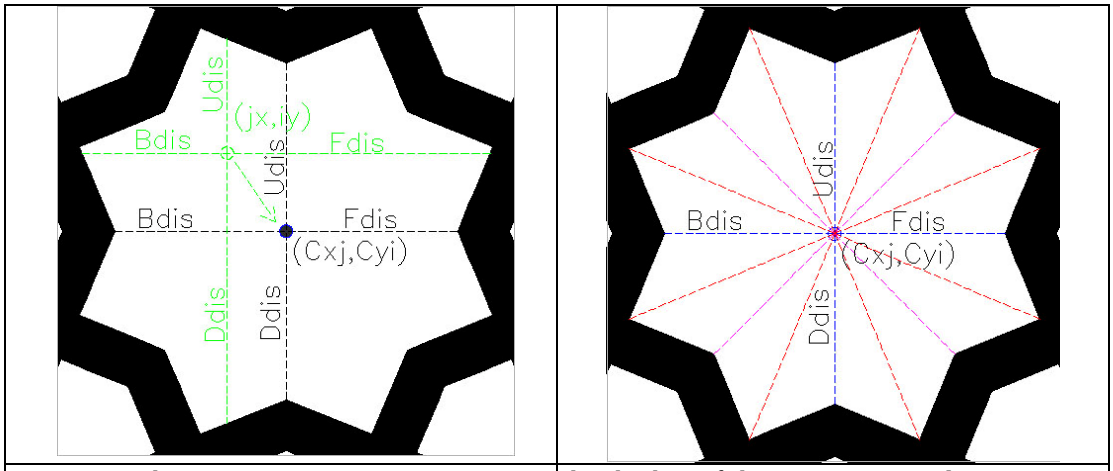
Geometric patterns are generated from a basic geometric unit using the circle as its basis, and then applying the principles of repetition, and symmetry (Kritchlow 1976, El-Said 1993). In order to be able to reconstruct the pattern accurately, it is crucial to read the available data and define a complete repeated unit and interpret it based on pattern structure. We assume that deteriorated patterns should have at least one intact repeated unit of the original structure; otherwise the program will not be able to generate the rest of the pattern. Tasks include defining the repeat unit based on the user input and then generating the whole basic grid based on that.

a) Defining the Repeat Unit

To identify this unit for octagon based patterns, the user is required to identify the two adjusting centers that constitute this unit. The user will pick any point within the central area of a star formation and the program will calculate and locate the center points. Based on the two locations, the program will define the repeated unit and will

save it within an array for further analysis. Figure 4.8 shows the two main steps for finding the center point of a region. First, the algorithms use the selected location of any point inside the region *(jx, iy)* to calculate the center point *(Cxj, Cyi)*. This is done through performing simple calculations using the four distances *Fdis, Bdis, Udis*, and *Ddis* (figures 4.8.a). To define the horizontal *x* coordinate of the center point location *(Cxj)*, we calculate the horizontal span by adding the two distances *Fdis* and *Bdis*, and locating the midpoint of that span. To define the vertical *y* coordinate of the center point location *(Cyi)*, we calculate the vertical span by adding the two distances *Udis* and *Ddis*, and then locating the midpoint of that span.

In the second step, the program checks for shape symmetry by comparing the sixteen distances from the center as shown in Figure 4.8.b. The algorithms shown in Figure 4.9, define certain difference threshold -which can be modified based on image resolution- for depicting the degree of shape symmetry. If the shape is proven symmetrical then it is saved into an array, and displayed on the screen. It is also possible to automatically determine the number of the polygon sides. This can be done by scanning the area around the centers in a form of an array starting from the located centers.



a. Locating the center point | b. Checking if shape is symmetrical.

Figure 4.8 The Process of Locating the Center Point of Pattern's Shapes.

```
/***********check if it is a symmetrical shape*********/

void check_if_symetrical(int xx, int yy){

               findCenter(int(Cxj),int (Cyi));
               if((fabs((Fdis-Bdis))<=6)&&
                  (fabs((Fdis-Ddis))<=6)&&
                  (fabs((Fdis-Udis))<=6)&&

                  (fabs((FUdis-BUdis))<=6)&&
                  (fabs((FUdis-FDdis))<=6)&&
                  (fabs((FUdis-BDdis))<=6)&&

                  (fabs((F2U1dis-B2D1dis))<=6)&&
                  (fabs((F1U2dis-B1D2dis))<=6)&&
                  (fabs((B2U1dis-F2D1dis))<=6)&&
                  (fabs((B1U2dis-F1D2dis))<=6))
                      {
                      test=Fdis;
                      if (Center==0)
                             {
                             array[0].x=Cxj;
                             array[0].y=Cyi;
                             array[0].rad=test;
                             array[0].color[0]=1;
                             array[0].color[1]=0;
                             array[0].color[2]=0;
                             save_flag=1;
                             }
```

Figure 4.9 The Algorithms for Depicting the Degree of Shape Symmetry.

The repeat unit of a geometrical pattern of octagon-based grid is defined by two centers; the main and the secondary centers. The main center is found in the dominant star unit, which is located at the center of the octagon shape in the basic grid. The secondary center is located in the shape that results from joining the star units. Unless the user selects two valid adjusting center points in a certain order, the program will not be able to define the repeat unit. The user has to select the main center first and then the secondary next. Moreover, the program will not allow the user to operate any further unless the user makes the right selection.

The program next calculates the distance between the two centers and automatically corrects the angles between them to calculate the area of the repeated pattern. If the two center points are located on a horizontal direction, the program checks if the *y* position of the two points are located exactly on the same vertical line. If the two center points are located on a vertical direction, then the program checks for the shift in the horizontal axes. In both cases, if a difference is found, the program modifies the secondary center position to be aligned with the main center.

Based on user selection and pattern rules, the algorithm is designed to locate the two main triangle shape repeat units located on both sides of the center line connecting the two selections (Figures 4.10.a and 4.10.b). The program needs only one side to generate the rest of the pattern. The choice of which is controlled by the user and based one the amount and quality of information retained in the image. Moreover, the program is designed to handle the eight different choices that might be selected by the user. These choices are related to the orientation of the main and secondary centers, and side selection. Figure 4.11 demonstrates the eight different triangle selection options marked in bold lines.

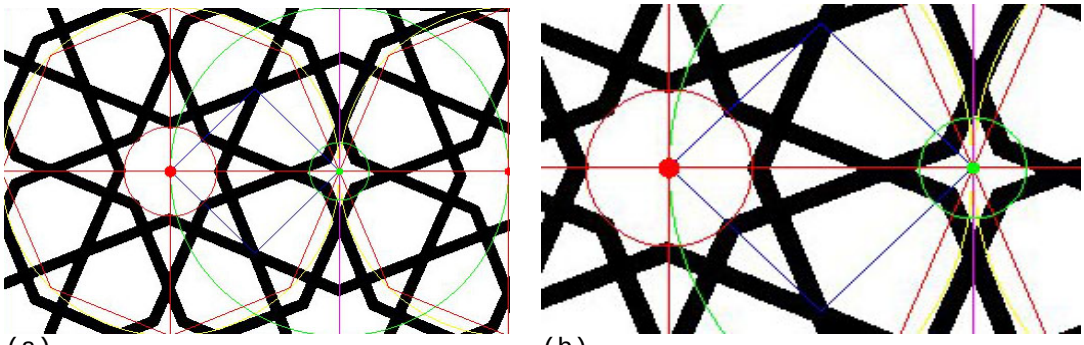


Figure 4.10 The Boundaries of the Two Repeated Units Triangles on Both Sides of the Two Selected Centers Selected by the User.

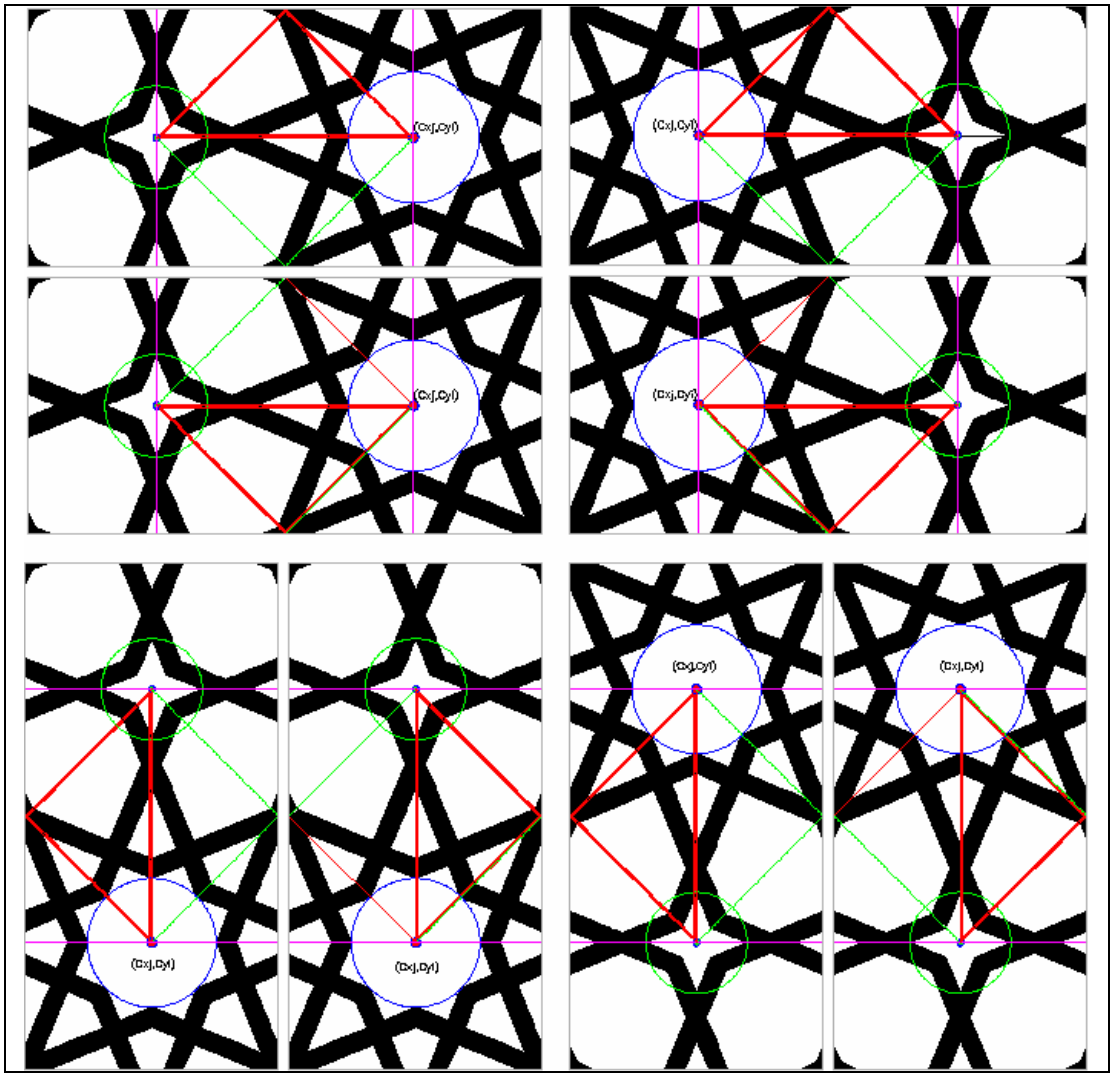


Figure 4.11 Visual Demonstration of the Eight Different Triangle Selection Options, which the Program Is Designed to Deal With.

b) Generating the Basic Grid

Based on the repeat units parameters defined by the users, the program calculates the rest of the center points using pattern-generating rules. Generating the rest of the center points is done based on the spatial relationship between the two identified centers of the repeated unit. The program will generates center points as long as it is within the boundaries of the image. The location of these points will provide the main matrix of the pattern that will be used in the second step to generate the rest of the pattern. If some part

is off, the program will relocate it to its right position before any further analysis. The final step here is to save all the locations in an array to be used for infilling the repeat stars. The generation of the basic grid is also possible based on these center points. Figures 4.12 and 4.13 demonstrate the generation of octagon basic grids. Basic grid information is saved in a two-dimensional array and also displayed on the screen for user evaluation.

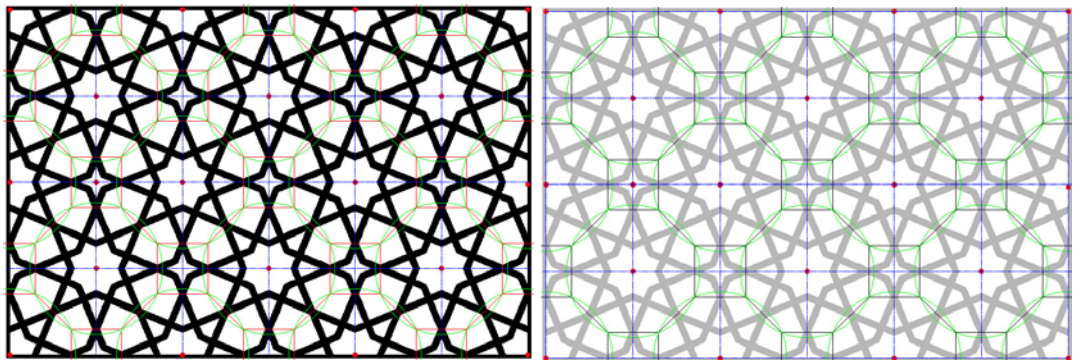


Figure 4.12 Generation of the Octagon-Square Basic Grid.

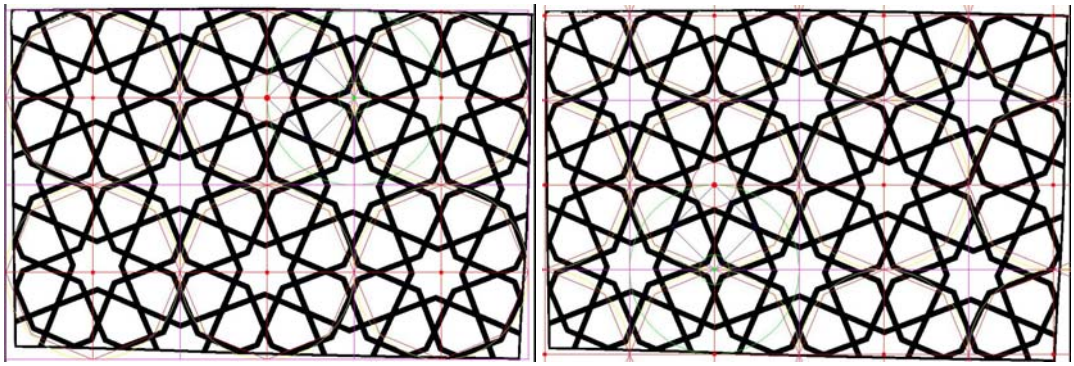


Figure 4.13 Generation of the Octagon Basic Grid.

2- Reading and Interpreting the Repeated Unit

In Islamic patterns, the repeat unit is generated from connecting intersections (points) of circles drawn by a prescribed graphical process (Kritchlow 1976). Based on this fact and on available data, this study aims at defining such intersections and then connecting them using vector data. To accomplish this aim, the first step was to look for existing thinning and skeletonization techniques to help in extracting the centerlines of the image features (thinning or skeletonization) (Hu and Li 1994, Zou 2003). Unfortunately, most of these techniques have problems at line pattern intersections, and are not able to preserve them (Zhong and Yan 1999). The only methods that looked promising are those techniques using run-length-wise processing (Li and Suen 1991, Hu and Li 1994, Zhong and Yan 1999). Therefore we chose run-length-wise processing to find pattern line width and to locate the convergence and divergence points at the intersection area. These points are used to define the beginning of the intersection area and then to calculate the exact intersection point. It also takes advantage of the fact that geometric patterns have a uniform line width, which can be discovered by conducting simple mathematical procedures (Hu and Li 1994). The process of defining the intersections in the repeat unit includes four main tasks: run-length-scan of the repeat unit, locating convergence and divergence points, defining intersection points, and finally connecting them using vector data.

a) Run-Length-scan of the repeat unit

We make use of the first step of run-length-wise operation to find and mark intersection regions (Hu and Li 1994, Zhong and Yan 1999). Run-length scan is carried out on the saved black and white image of the area around the repeat unit. This region is highlighted in Figure 4.14. The first scan is carried out from the left to the right and black run-lengths are saved in a linked list. The algorithms are shown in Figure 4.15. This scan process is repeated eight times (left-to-right, right to left, top to bottom, bottom to top, diagonally from the upper left corner to right bottom corner, etc.

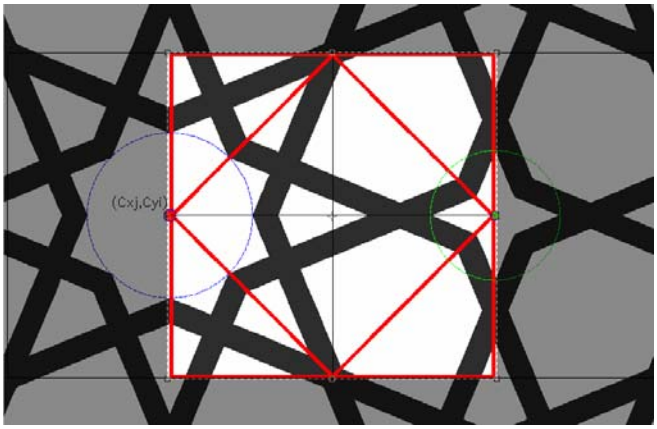


Figure 4.14. Area of Run Length Scan.

b) Locating convergence and divergence points

After saving run-lengths in linked lists, each list is processed to find the convergence and divergence points that define the intersection regions (Hu and Li 1994, Zhong and Yan 1999). These points are found by checking the current run-length with the previous run-length on the previous scan line for any overlap. There can be four cases:

Case 1: no overlap (Line starts or ends).

Case 2: One overlap (Line continues).

Case 3: Two overlaps (Convergence or divergence condition)

Case 4: More then two overlaps (More that one intersection)

The resulting points are saved into different arrays based on the direction of scan. The algorithm for doing this process is displayed in Figure 4.16.

```
void Link_List(void){
unsigned char red,current,previous;
int i,j, iy, jx;
count=0;

/****scan horisontally and save in link list**********/

for (i=int (array[3].y+radd/2); i>= int (array[3].y-radd/2);i--)
  {
  previous=255;
  for(j= int (array[3].x-radd/2); j<= int (array[3].x+radd/2); j++)
        {
        pixel = bitmap[i][j];
        red= pixel;
        current=red;
        if( previous==255/*white*/&& current == 0 /*black*/)
                {
                count++;
                /*alocate space for struct ( row, column, length)*/

                if (head == NULL)
                      {
                      head= (RUN*)malloc(sizeof(RUN));
                      tail=head;
                      }
                 else  /* is not the first time */
                      {
                      tail-> next=(RUN*)malloc(sizeof(RUN));
                      tail = tail->next;
                      }
                tail-> row=i;
                tail-> col=j;
                tail-> length=1;
                }
                else if( previous==0 && current == 0)
                        {
                        tail-> length++;
                        }
                else if( previous==0 && current == 255 )
                        {
                        tail-> next = NULL;
                        }
        previous= current;
        }
   }

}
```

Figure 4.15. Algorithms for Scanning the Repeat Unit Horizontally and Saving It in a Linked List.

```
void find_Conv_Div(void)
{
RUN *change, *element;

/****search link list (head)for conversions and diversions*******/
if(head!=NULL)
{
for (change=head; change!=NULL; change=change->next)
 {
 /* Start ===find conversion points  */
 for (element=head;element!=NULL;element=element->next)
  {
  if(element->row==change->row-1&& element->length>=4
    &&change->length>=4)
    {
    L_C_index=0; /* left conversion index  */
    L_D_index=0; /* left diversion index  */
    for (c=change->col;c<=((change->col+change->length)-1);c++)
     {
     for(e=element>col;e<=((element->col+element->length)-1);e++)
      {
      if (c==e)
        {L_C_index++;L_D_index++;}
      }
     }
   if(L_C_index >=1 && element->row== change->next->row-1&&
      change->next->length>=4)
     {
     R_C_index=0;/* right conversion index*/
     for(c=change->next>col;c<=((change->next->col+change->
        next->length)-1);c
       {
     for(e=element->col;e<=((element->col+element->length)
          -1);e++)
         {if (c==e)R_C_index++;}}
            if(R_C_index>=1)
               {conversion(change, C_indx);}}
  if(L_D_index >=1 && element->next!= NULL && element->next->
     row==change->row-1&& element->next->length>=4)
        {R_D_index=0;
        for(c=change->col; c<=((change->col+change->length)-1;c++)
  for(e=element->next->col; e<=((element->next->col+element->
      next->length)-1);e++)
        {if (c==e)R_D_index++;}}
        if(R_D_index>=1)
        {diversion(element, D_indx);
             }  }
       }  }
}  }
```

Figure 4.16 Algorithms for Finding the Conversion and Diversion Points.

c) Calculating the Estimated Intersection Points

At the beginning, the algorithm proposed by the pilot study was implemented. Unfortunately, many problems were found while executing the program on complicated patterns. These problems were related to defining the conversion and diversion angles. Based on these problems, another process of calculating the estimated intersection location is done based on the convergence and divergence points. This new process is demonstrated in Figures 4.17 and 4.18. The conversion point *Cpoint* at position *(xj,yj)* is used as the center for a polar scan until it hits a white pixel (The edge of the line). In Figure 4.17 these points are marked *P1* and *P2*. Because of the distorted nature of the image, these two lines do not necessary represent the exact intersection edges. Therefore, the program needs to find a way to correct its position as much as possible. A line then is drawn between these two points and the center point is identified as *P3. P3* might not represent the exact intersection location. To correct this problem, the program draws a line from the original convergence point to *P3*, and then extends this line until it hits a white pixel. It then calculates the center point of the extended line and locates that point *(P4)* as the intersection location. For X crossings this process is done four times and then the final result is averaged from the four intersection points Figure 4.17. For V crossings, only one convergence point is calculated as shown in Figure 4.18.

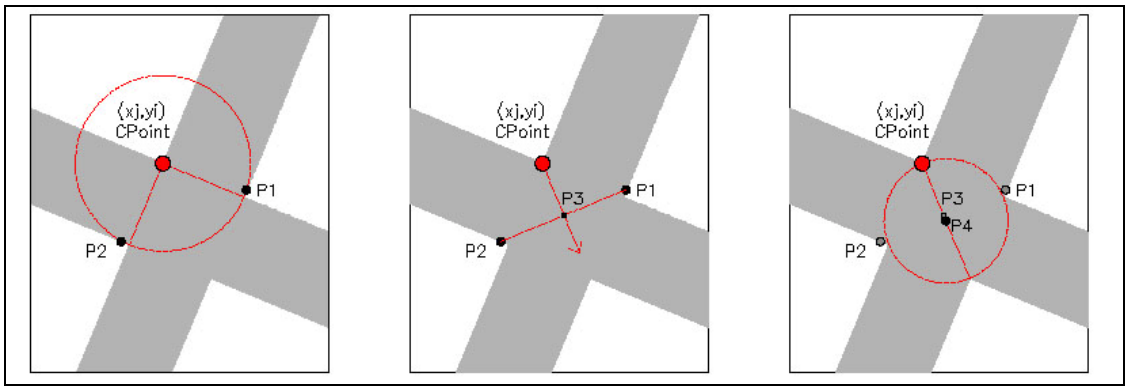


Figure 4.17. Calculating the Exact Intersection Location for X Crossing.

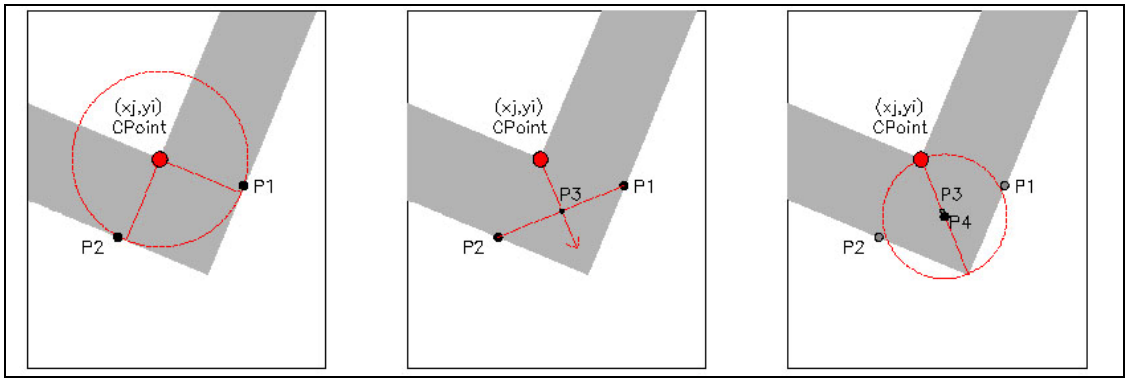


Figure 4.18. Calculating the Exact Intersection Location for V Crossing.

The algorithms for finding the two edge points (*P1*, and *P2*) are shown in Figure 4.19. The algorithms for finding the final intersection location based on one conversion point is displayed in Figure 4.20. Detailed results from applying these algorithms on image data are shown in Figure 4.21. These algorithms are used to define the intersections inside the repeat unit's boundaries. The boundaries themselves are scanned to define the intersection on the edge of the repeated unit using simple algorithms. The results from the two algorithms are then combined.

3- Connecting Points Using Vector Data

This is done by simply scanning the area between each two points to check if it is covered all the way by black pixels. If white pixels are found the program does not connect them. Final lines are then saved in an array of a line structure (Figure 4.22).

These algorithms were tested on images holding Islamic patterns and results from applying them are shown in Figure 4.23. Figure 4.24 shows the results from applying these algorithms to more complicated patterns.

```
/******************find edge points*****************/
void Final_intersection_C(float x0,float y0, float Dis)
{
int  flag_R=0,flag_L=0;
unsigned char red;
float theta,distance, x, y, D, X_R, Y_R, X_L, Y_L, r,theta1,
final_x, final_y ;

for( r=4; r<=3*Dis; r+=0.3)
  {
  for(theta=PI*35/18; theta >=(PI*30/18) ; theta -= CIRC_INC)
    {
    x= x0+ (r*cos(theta));
    y= y0+ (r*sin(theta));
    pixel = bitmap[int (y)] [int(x)];
    red= pixel;
    if(red==255 &&flag_R==0)
      {
      X_R=x;
      Y_R=y;
      flag_R=1;
      }
   }
for(theta=PI*19/18; theta <=(PI*25/18) ; theta += CIRC_INC)
  {
  x= x0+ (r*cos(theta));
  y= y0+ (r*sin(theta));
  pixel = bitmap[int (y)] [int(x)];
  red= pixel;
  if(red==255 &&flag_L==0)
    {
    X_L=x;
    Y_L=y;
    flag_L=1;
    }
  }
}
if(flag_R==1&& flag_L==1)
  {
  Final_point(x0, y0, X_R,Y_R ,X_L ,Y_L);
  F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
  F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);
  Draw_line( X_R, Y_R ,X_L , Y_L);
  }

}
```

Figure 4.19 The Algorithms for Finding the Two Edge Points (*P1* and *P2*).

```
/************************final point**************************/
void Final_point(float x0, float y0, float X_R,float Y_R ,float X_L
,float Y_L){

theta=degrees(X_R, Y_R ,X_L , Y_L);
distance=(findDis(X_R , Y_R ,X_L , Y_L ));
final_x= extendx(X_R , Y_R, theta, distance/2 );
final_y= extendy(X_R , Y_R, theta, distance/2 );
F_Circlef_rgb( final_x ,final_y, 1, 0,1,0);

degree=degrees(x0,y0, final_x, final_y );
theta1=degree * PI/180;
distance1=(findDis( x0,y0, final_x, final_y ));

for ( r=0; r<=(4*distance1);r++)
        {
        x= final_x+ r *cos(theta1);
        y= final_y+ r *sin(theta1);
        pixel=bitmap[int(y)][int(x)];
        red=pixel;
        current=red;

        if(r==int(4*distance1) && White==0)
          {current=255;}
        if(current==255 && White==0)
          {
           White=1;
           xx=x;
           yy=y;
           r=(4*distance1+2);
          }
        }
if(White==1)
        {
        distance2=(findDis( x0,y0,xx ,yy ));
        Draw_line( x0,y0, xx, yy);
        theta2=degrees( x0,y0, xx, yy);
        final_xx= extendx(x0, y0, theta2, distance2/2 );
        final_yy= extendy(x0, y0, theta2, distance2/2 );
        F_Circlef_rgb( final_xx ,final_yy, 2, 1,0,1);
        }
inter[inter_count].x = final_xx;
inter[inter_count].y = final_yy;
inter_count++;
    }
   }
}
```

Figure 4. 20 The Algorithms for Finding the Final Intersection Location Based on One Conversion Point.

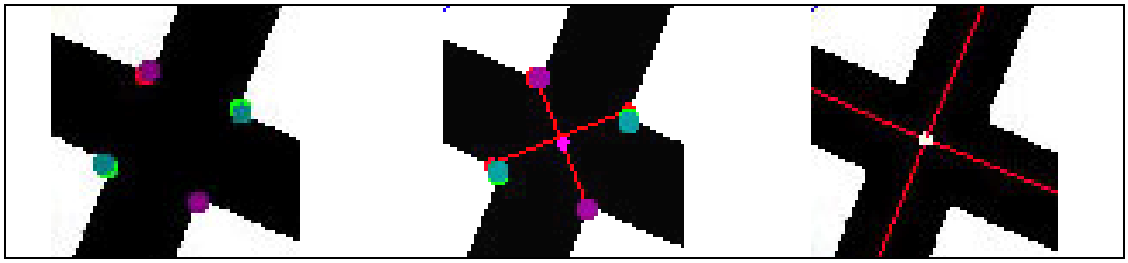


Figure 4.21. Results from Applying Intersection Finding Algorithms.

```
typedef struct  Tow_points {
        float x0,y0,x1,y1 ;
}LINE;

LINE line[100];
```

Figure 4. 22 Final Vectors Are Saved in a Line Array.

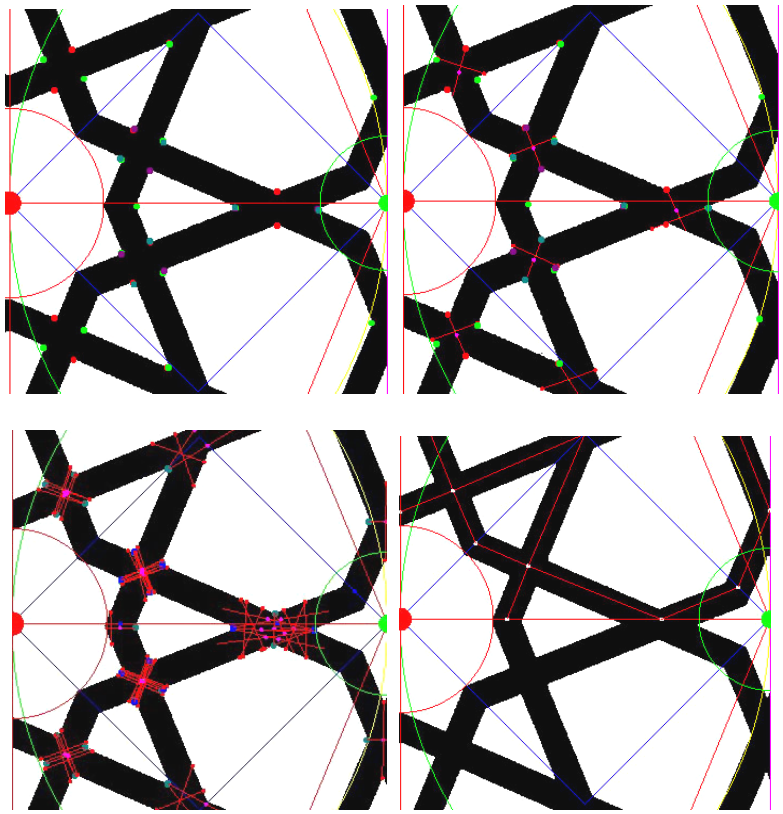


Figure 4.23 The Process of Applying the Program to an Islamic Pattern of Simple Complexity.

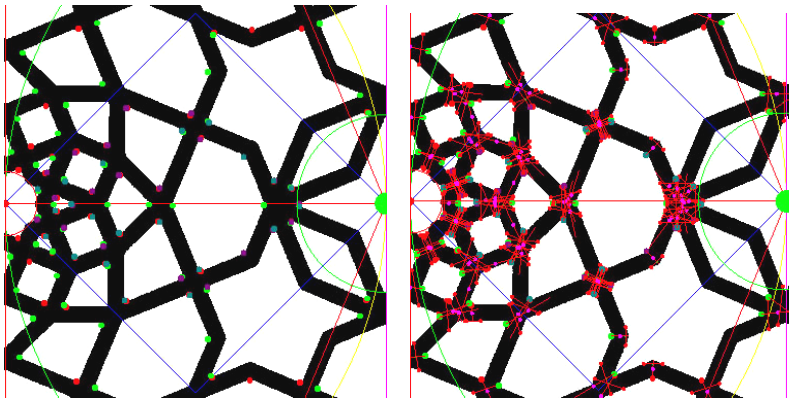


Figure 4.24 The Process of Applying the Program to a Complex Islamic Pattern.

4- Pattern generation

Pattern generation includes two main steps:

a) Generating one complete star unit by applying the principles of repetition and symmetry to the identified repeated unit based on the two located centers and the rules of pattern's generation (Figure 4.25).

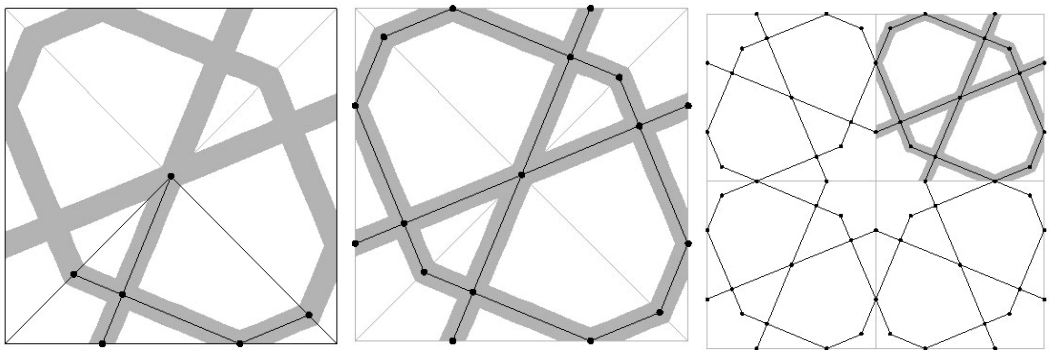


Figure 4.25 The Process of Generating a Complete Star Unit.

b) Constructing the whole pattern through copying the star unit to the basic grid pattern generated earlier (Figure 4.26). Different locations of the saved center points throughout the pattern. The final step will be trimming the pattern based on image boundary.

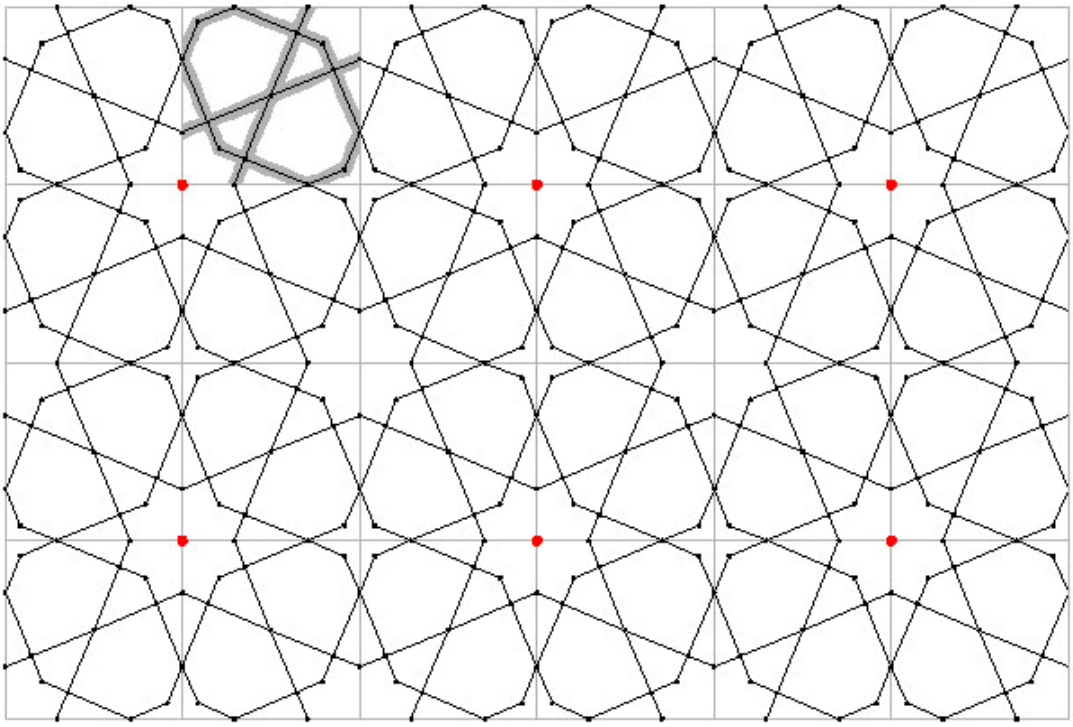


Figure 4.26 The Process of Constructing the Whole Pattern.

4.3.3 Program Output

This program is designed to display all results on screen and output three different sets of information.

a) The Final Reconstructed Pattern.

The program is designed to output the final reconstructed pattern as a line drawing (Postscript). Figure 4.27 displays an EPS image file of the final reconstructed pattern.

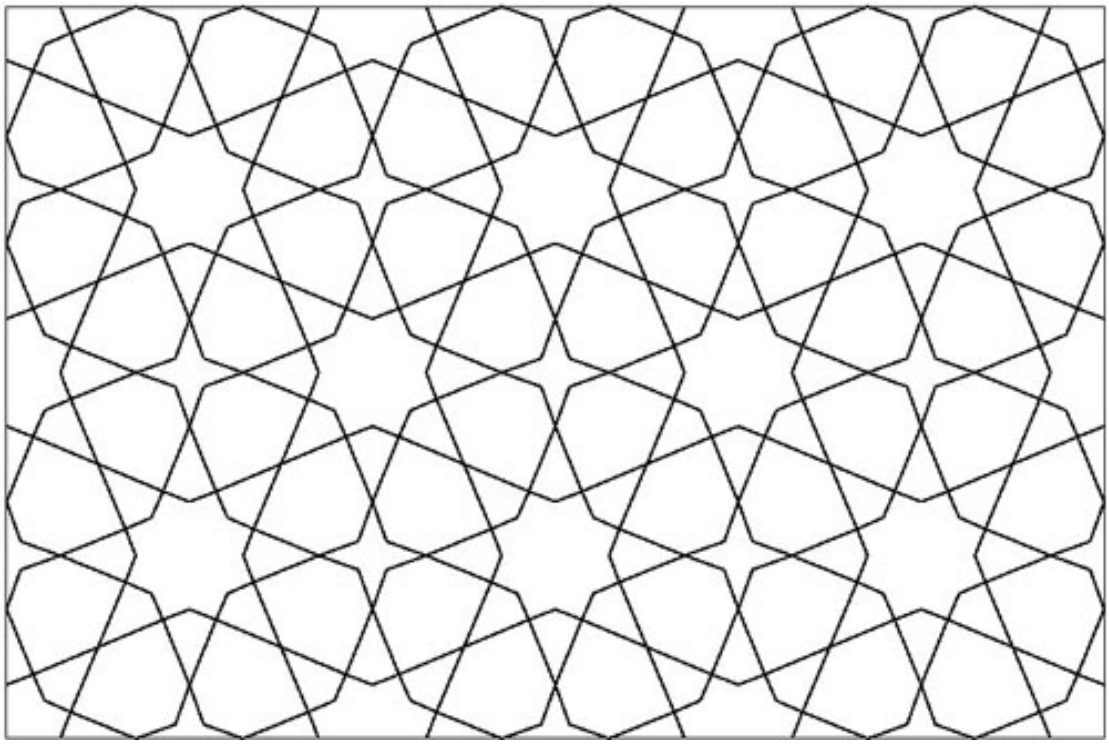


Figure 4.27 An EPS Image File Format of the Final Reconstructed Pattern.

b) Pattern Measurements

The program also outputs the main measurements of the reconstructed pattern. All measurements are proportional and can be scaled to any ratio. These measurements are:

1) The (X, Y) location of all the intersection points of the main repeat unit (Figure 4.28).
2) The distances from the main center of the repeat unit to each of the intersections of the repeat unit (Figure 4.28).
3) The spacing between the repeat pattern centers, pattern height, and pattern width (Figure 4.28).
4) The number of run-length encoding and the minimum and maximum length of the run-length elements (Figure 4.29).
5) The location of all the center points of the basic grid (Figure 4.30).

6) Moreover, the program is able to calculate any pattern measurements through using these basic measurements.

```
inter[0].x =246.000000
inter[0].y =206.500000
inter[1].x =287.000000
inter[1].y =206.500000
inter[2].x =348.910950
inter[2].y =206.500000
inter[3].x =253.000000
inter[3].y =190.138565
inter[4].x =310.000000
inter[4].y =166.528397
inter[5].x =364.500000
inter[5].y =166.000000
inter[6].x =336.503601
inter[6].y =176.994400
array[0].x =213.50
array[0].y =206.50
spacing between centers=191.50
pattern height=766.00
pattern width=1149.00
dis from array[0] to inter[0]=32.50
dis from array[0] to inter[1]=73.50
dis from array[0] to inter[2]=135.41
dis from array[0] to inter[3]=42.75
dis from array[0] to inter[4]=104.45
dis from array[0] to inter[5]=156.34
dis from array[0] to inter[6]=126.49
```

Figure 4.28 The (X, Y) Location of All the Intersection Points of the Main Repeat Unit.

```
count =504
Min =1
Max =43
```

Figure 4.29 The Number of Run-Length Encoding and the Minimum and Maximum Lengths of the Run-Length Elements.

```
start_y =600.500000
start_y =207.500000
start_x =214.500000
 centr[0 ][0 ].x =214.50
 centr[0 ][0 ].y =207.50
 centr[0 ][1 ].x =607.50
 centr[0 ][1 ].y =207.50
 centr[0 ][2 ].x =1000.50
 centr[0 ][2 ].y =207.50
 centr[1 ][0 ].x =214.50
 centr[1 ][0 ].y =600.50
 centr[1 ][1 ].x =607.50
 centr[1 ][1 ].y =600.50
 centr[1 ][2 ].x =1000.50
 centr[1 ][2 ].y =600.50
 indx_y =2, indx_x =3
```

Figure 4.30 The Location of All the Center Points of the Basic Grid.

c) PPM Black and White Image

The program also out puts a black and white PPM image, which is based on the user's input contrast value (Figure 4.31). This output is optional and only executed if the user selects the SAVE button in the menu tool bar.

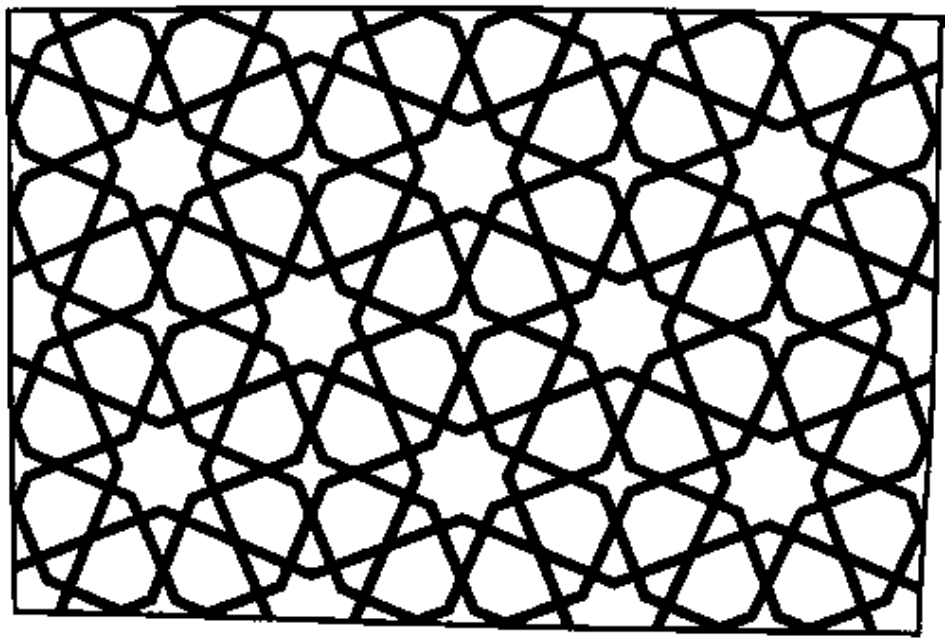


Figure 4. 31 The Program Outputs a Black and White PPM Image.

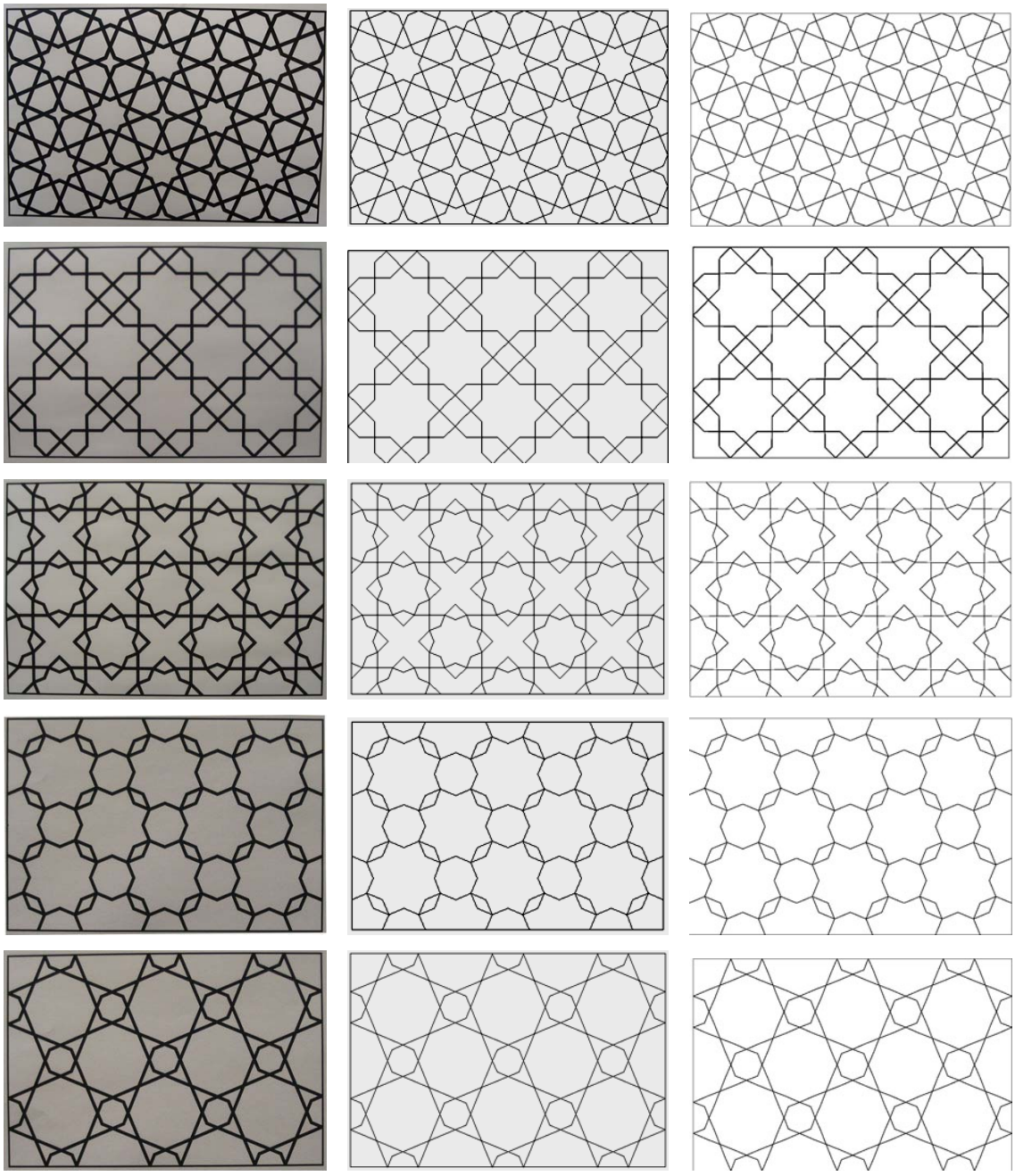
CHAPTER V
ANALYSIS AND RESULTS

5.1 Analysis and Interpretation

Analysis and interpretation are based on six different sets of experiments executed by the researcher. These experiments were designed to test research hypothesis by evaluating the effect of the different independent variables on the two main dependent variables: structural accuracy and repeatability. A qualitative approach based on visual observation was used to evaluate and interpret the different outcome results. In this section I will present each of the six different sets of experiments coupled with their analyses and results.

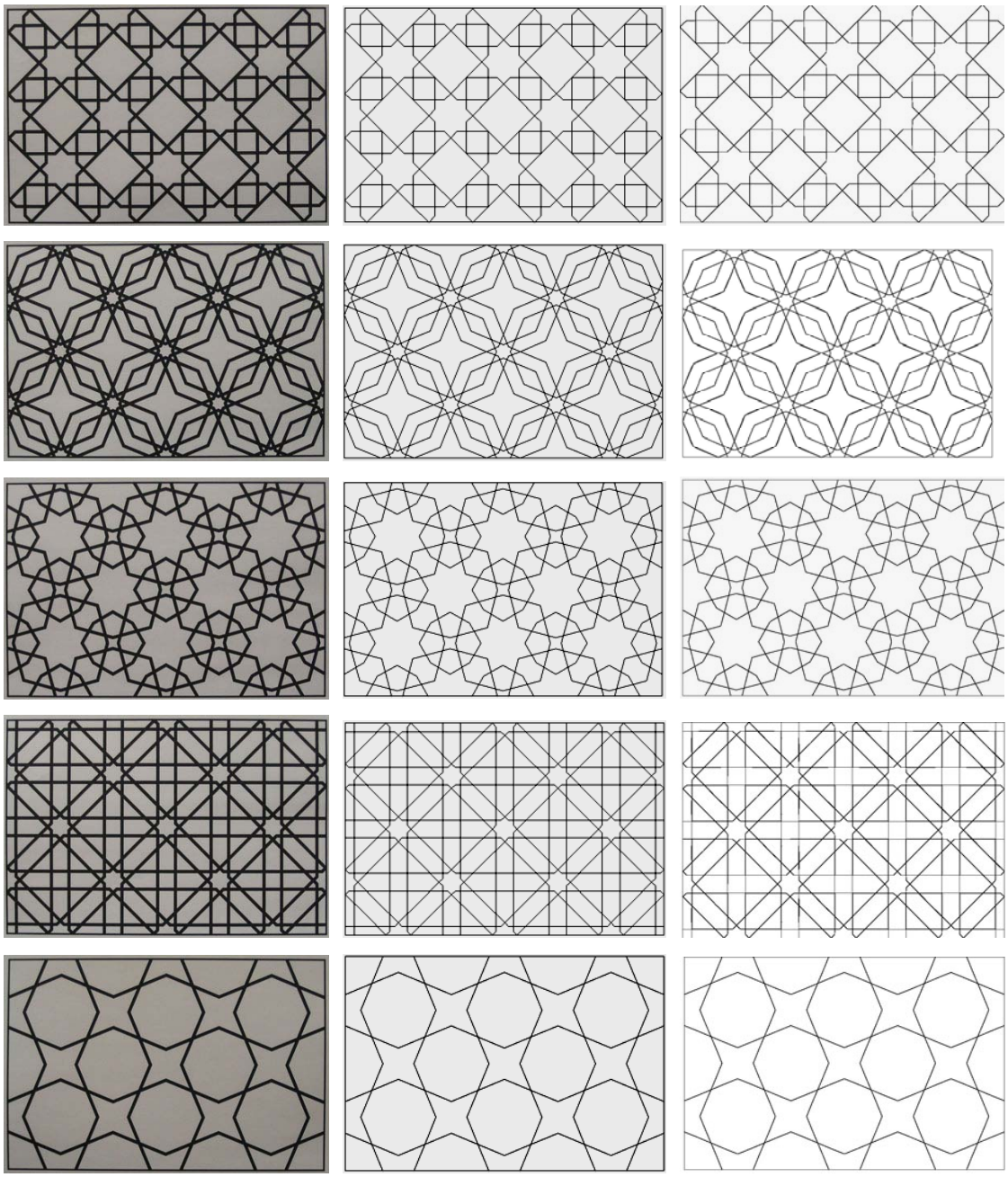
5.1.1 First Set of Experiments

The first group of experiments is designed to test the ability of the program to capture the right geometry of full patterns with different levels of complexities. It tests the performance of the program in documenting different levels of pattern complexity in order to evaluate its effect on the two dependent variables: structural accuracy, and repeatability. The resolution and contrast threshold value 0.50 was kept constant through out the experiments. Input samples were complete geometric patterns with an approximate resolution of 1200x800 pixel/image. Thirty input images were tested; ten different pattern designs from three different complexity levels. The operator is assumed to have some knowledge of how to operate the program. Evaluation of the program's performance was done by comparing the simulated structures with the original patterns. Visual comparison and manual measuring methods were used to define the degree of structural similarities between the outcome data and the original patterns. Figures 5.1a, 5.1b, 5.2 a, 5.2b, 5.3a and 5.3b display the reconstructed output results of all the samples. The tables display the final output patterns in EPS file format, the input image, and the original line geometry, which was created by AutoCAD. The six tables display the three different groups of complexity: simple, medium, and complex.



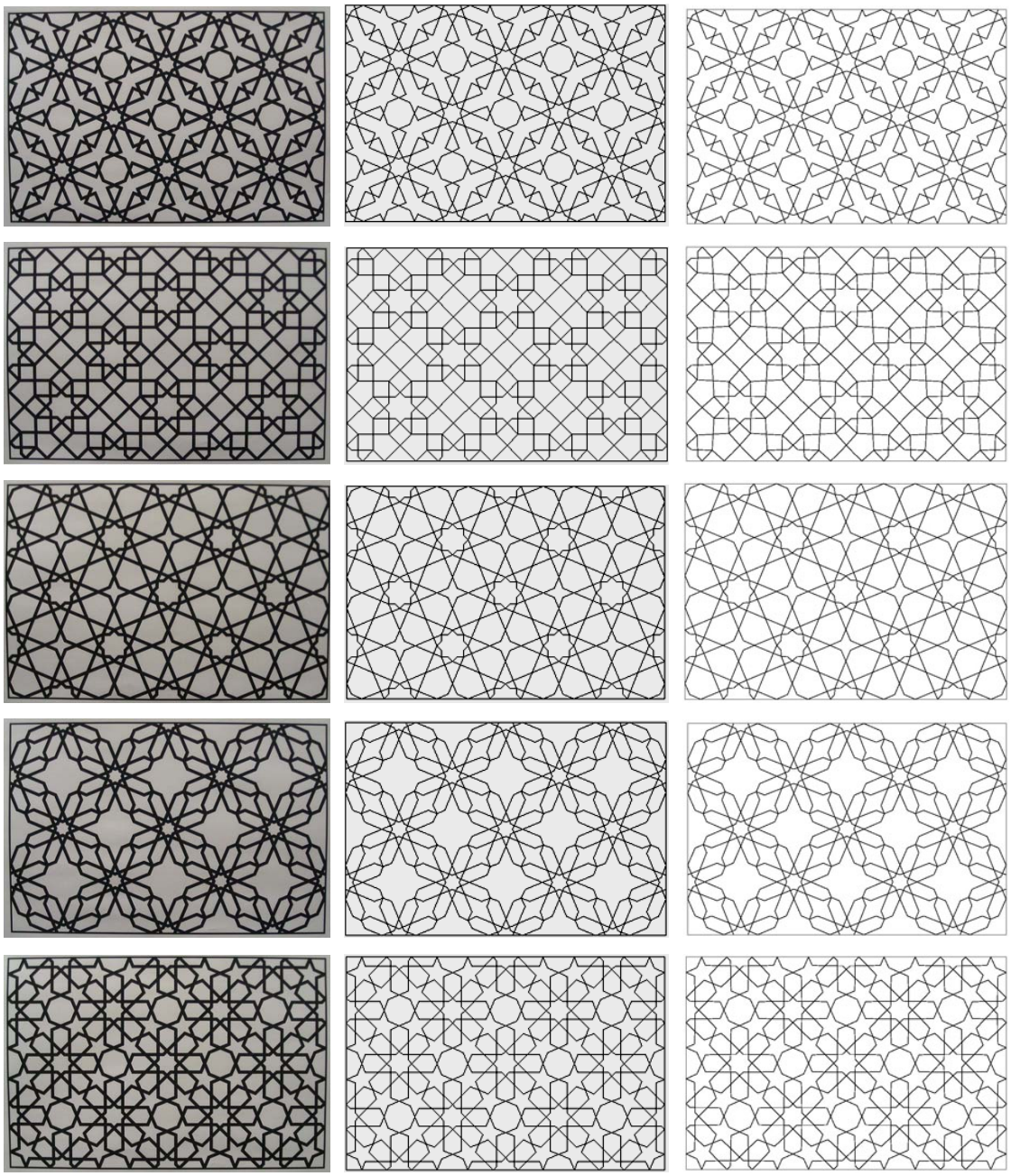
Input image	AutoCAD original drawing	Output pattern

Figure 5.1a. The Original Input Image (left), the AutoCAD Original Drawing (middle) vs. the Reconstructed Output (right) of Simple Patterns.



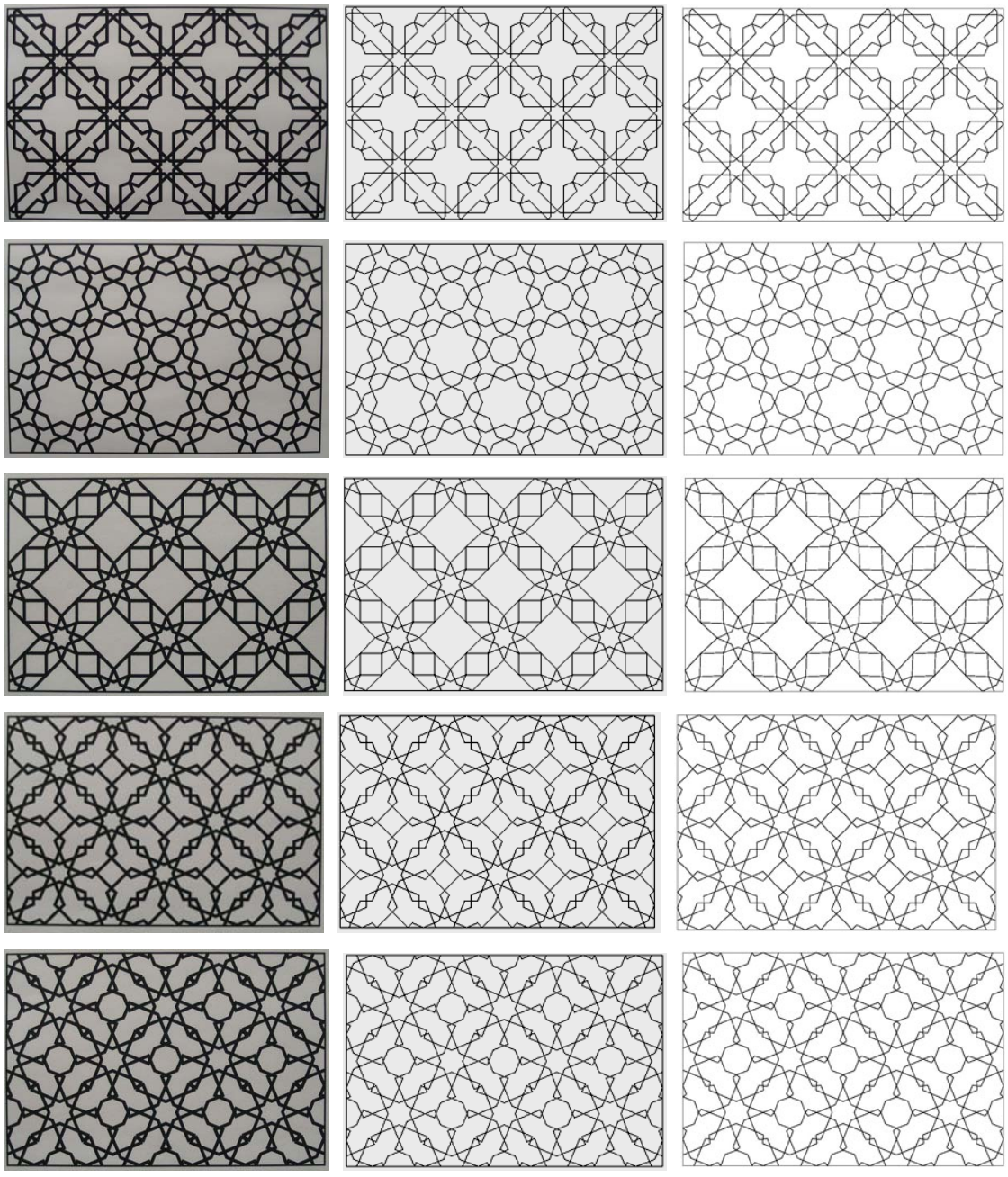
Input image	AutoCAD original drawing	Output pattern

Figure 5.1.b. The Original Input Image (left), the AutoCAD Original Drawing (middle) vs. the Reconstructed Output (right) of Simple Patterns.



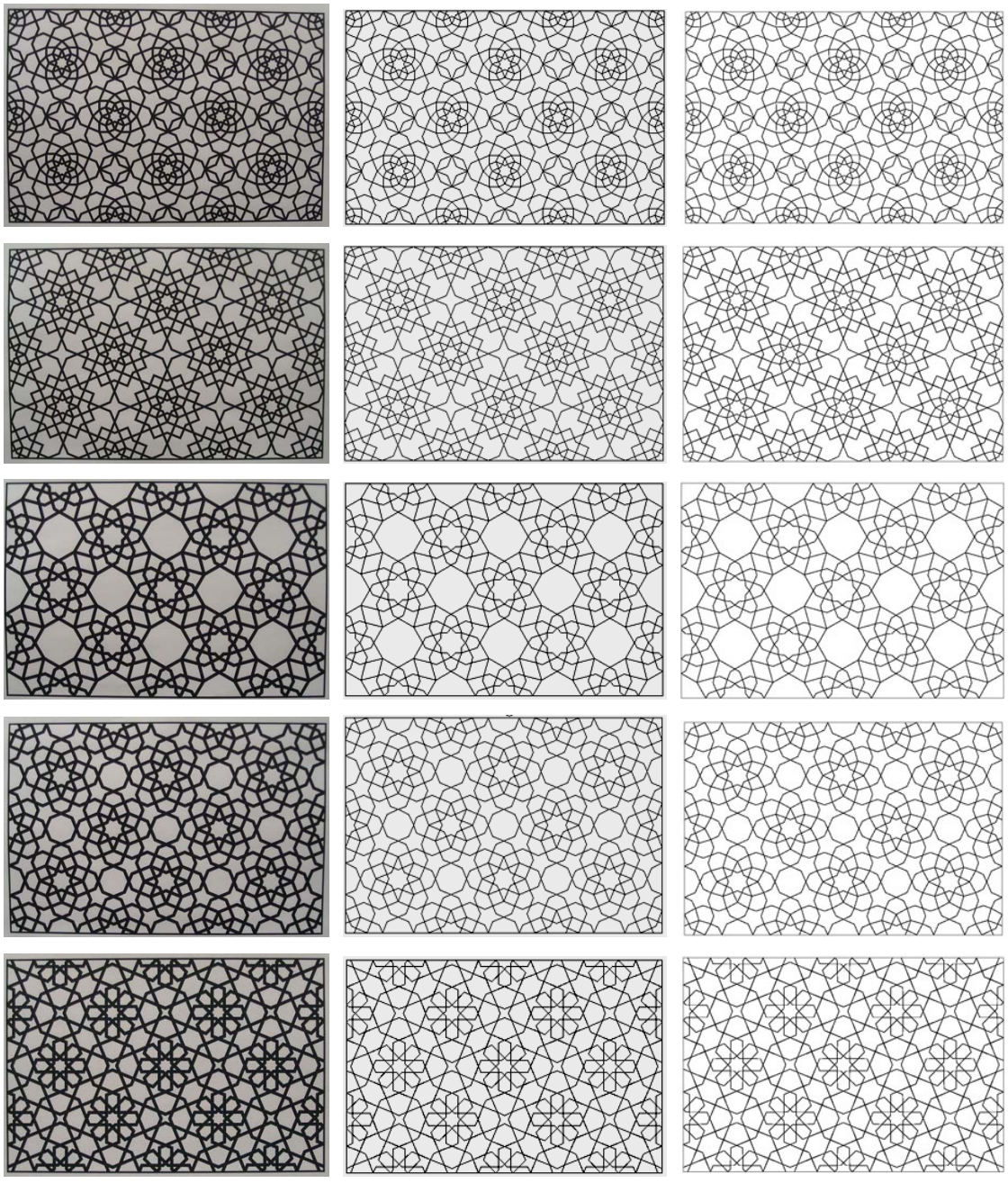
Input image	AutoCAD original drawing	Output pattern

Figure 5.2.a. The Original Input Image (left), the AutoCAD Original Drawing (middle) vs. the Reconstructed Output (right) of Medium Patterns.



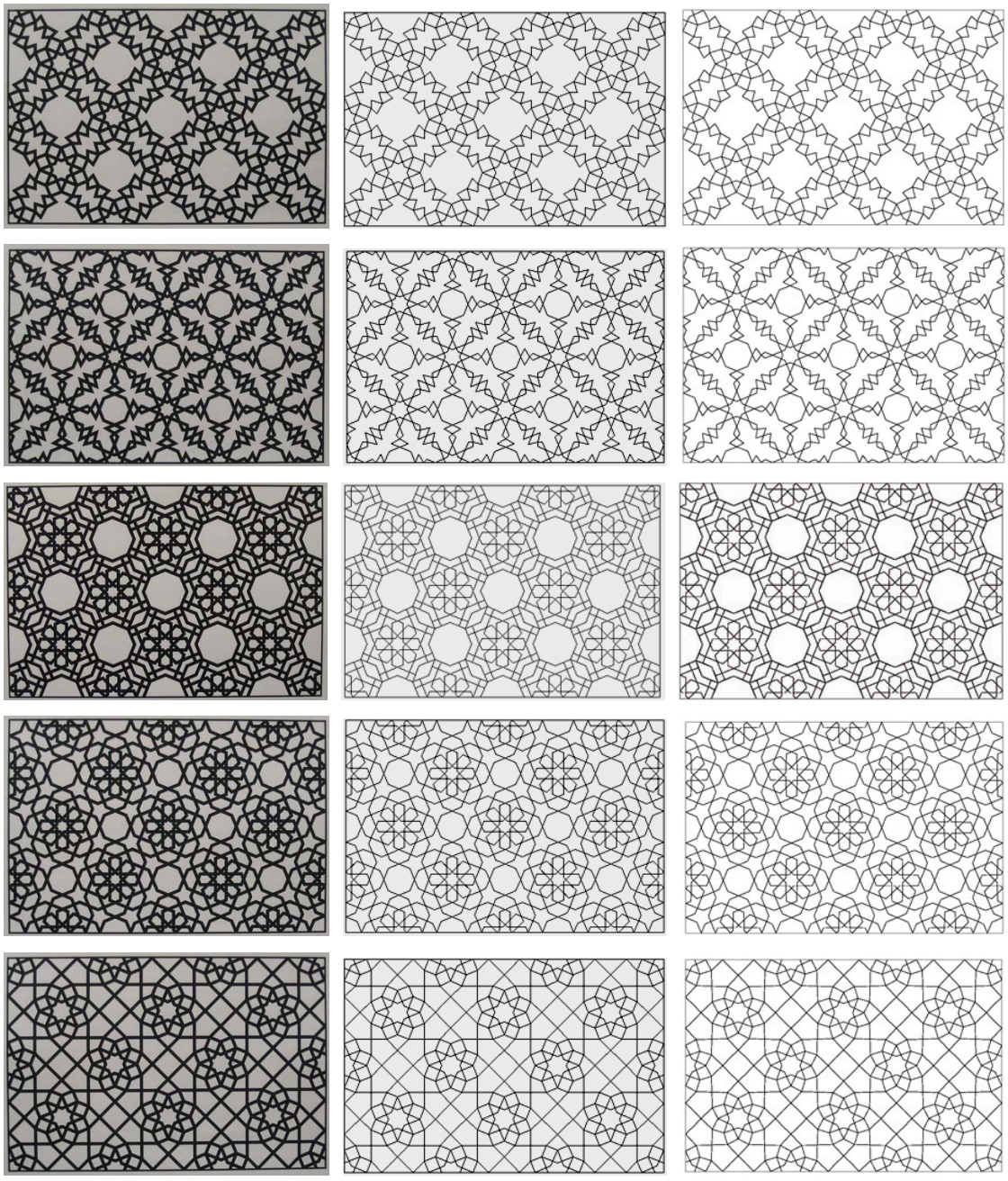
Input image	AutoCAD original drawing	Output pattern

Figure 5.2.b. The Original Input Image (left), the AutoCAD Original Drawing (middle) vs. the Reconstructed Output (right) of Medium Patterns.



Input image	AutoCAD original drawing	Output pattern

Figure 5.3.a. The Original Input Image (left), the AutoCAD Original Drawing (middle) vs. the Reconstructed Output (right) of Complex Patterns.



Input image	AutoCAD original drawing	Output pattern

Figure 5.3.b. The Original Input Image (left), the AutoCAD Original Drawing (middle) vs. the Reconstructed Output (right) of Complex Patterns.

Results and Findings

1- The program is found successful in capturing the accurate geometry of full patterns within the different levels of complexity.
2- The program is found consistent and reliable in documenting full patterns with different complexities. Good repeatability results were accomplished.
3- The resolution of 1200x800 pixel/image seemed to work with all patterns. Moreover, the 0.50 threshold contrast value was ideal for all patterns.
4- Within the defined parameters; the level of complexity did not have any effect on the final results.
5- The time needed to process each pattern was short (only a few seconds).

5.1.2 Second Set of Experiments

The second group of experiments is designed to test the ability of the program to capture the right geometry of full patterns with different resolutions. It tests the performance of the program in documenting full patterns within different resolutions in order to define the acceptable lower resolution boundaries for each complexity group. Thirty patterns were tested; ten different pattern designs from three different complexity levels. Each pattern was tested with different resolutions starting with 200x137 pixel/image and over. Analysis and evaluation of program performance were done through comparing the simulated structures with the original pattern structure through visual comparison. Patterns from each complexity group are tested to define the working range of the lower resolutions. Figures 5.4, 5.5, and 5.6 display some final reconstructed results applied to images with different resolutions. The chosen patterns represent the two edges of the working resolution range.

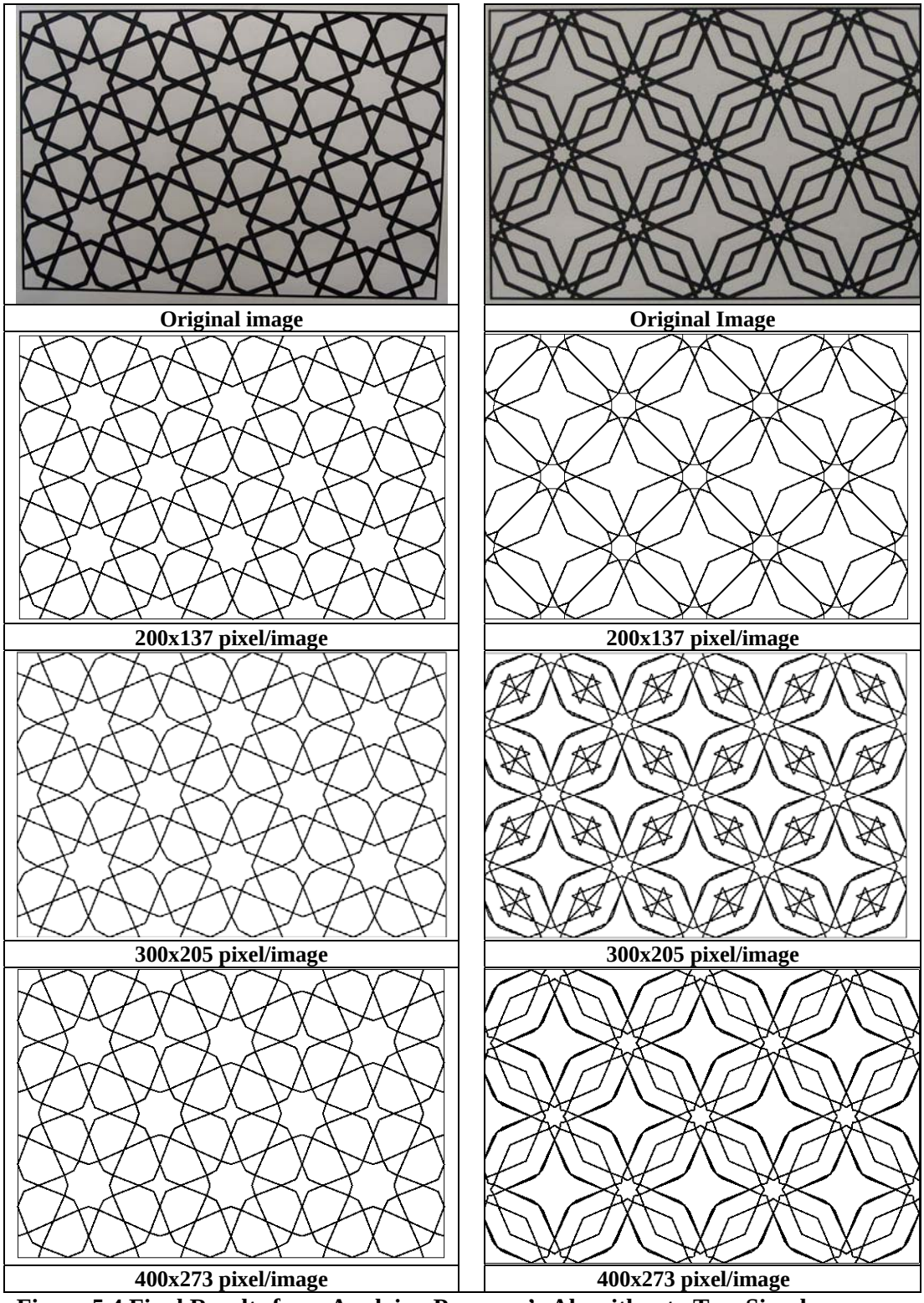


Figure 5.4 Final Results from Applying Program's Algorithm to Two Simple Patterns with Three Different Resolutions.

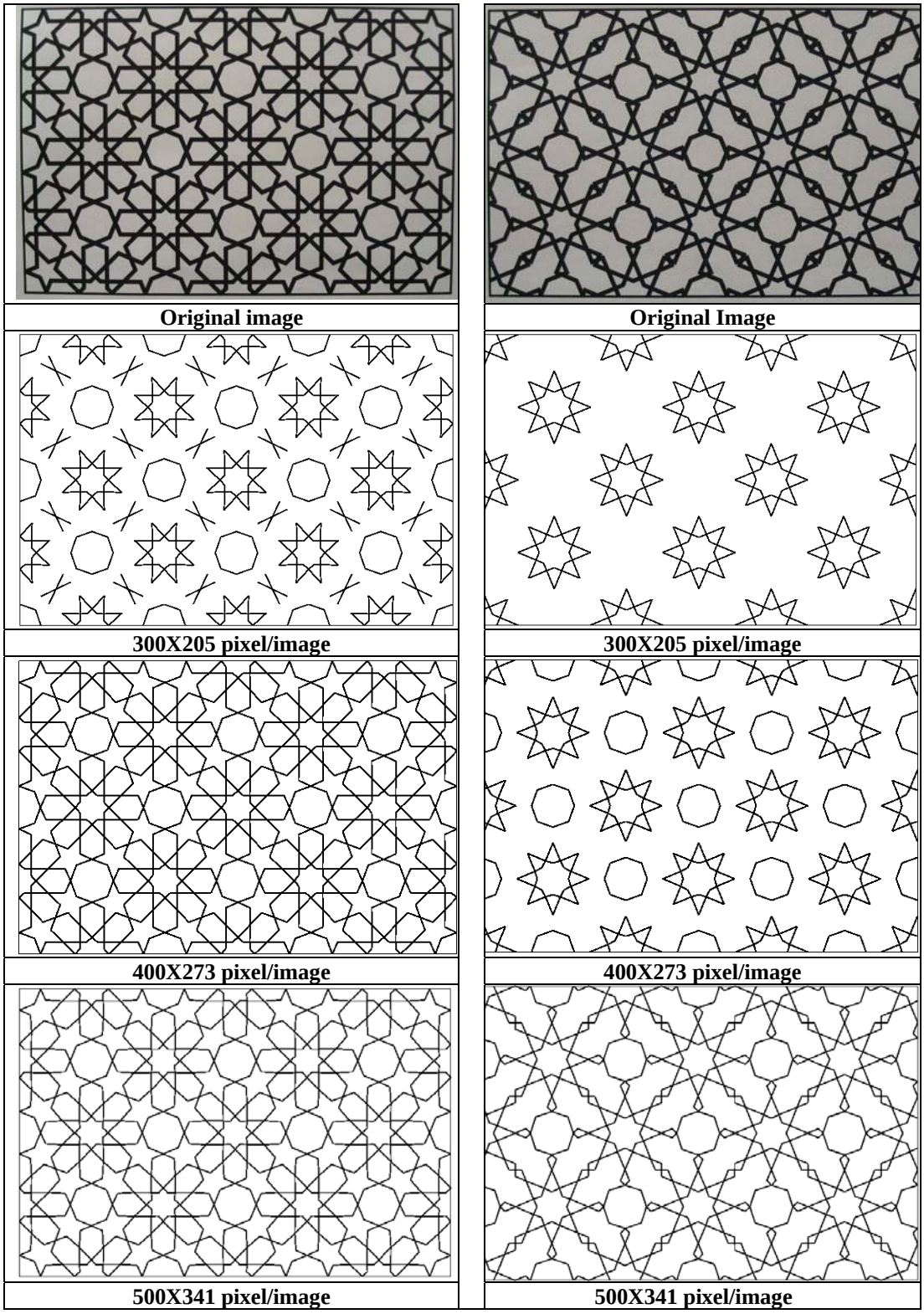


Figure 5.5 Final Results from Applying Program's Algorithm to Two Medium Patterns with Three Different Resolutions.

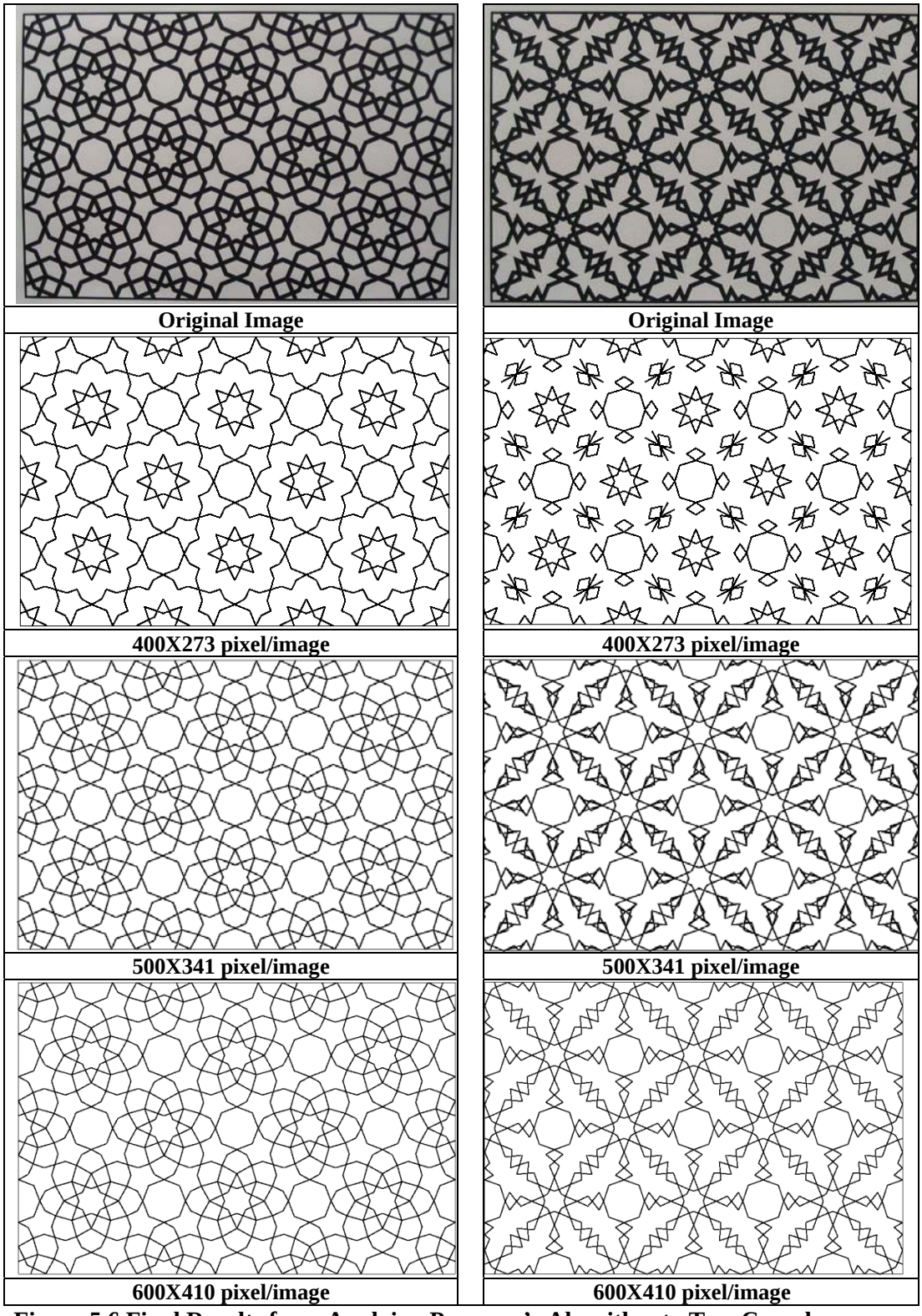
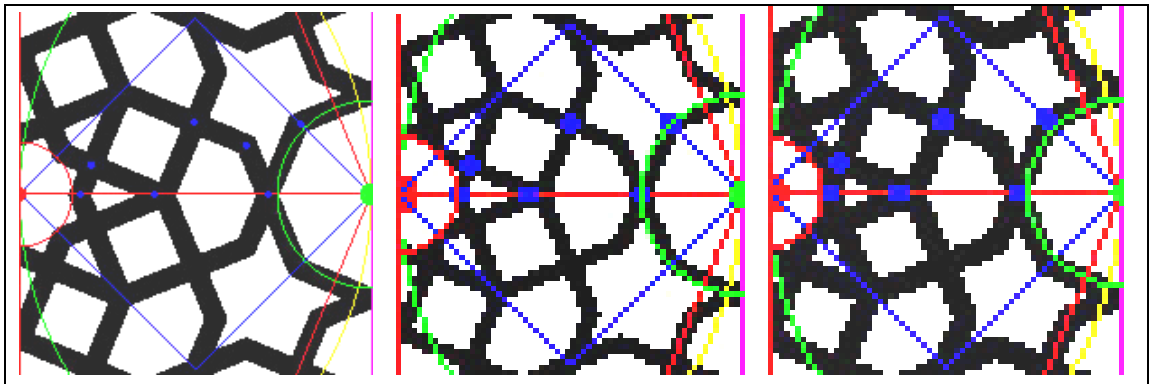


Figure 5.6 Final Results from Applying Program's Algorithm to Two Complex Patterns with Three Different Resolutions.

Results and Findings

1- The lower resolution boundary range for patterns of simple complexity is 200x137 to 400x273 pixel/image.
2- The lower resolution boundary range for patterns of medium complexity is 400x273 to 500x341 pixel/image.
3- The lower resolution boundary range for patterns of high complexity is 400x137 to 600x410 pixel/image.

Figure 5.7.a shows the final results of testing high-resolution image compared to low resolution images of different threshold values (figure 5.7.b, and 5.7.c). The result from testing the high resolution seems much more accurate in capturing all the intersection points. The results from low-resolution image captured almost all the intersection points except for one. The program in this case failed to define the conversion point due to angle distortion. This problem can be overcome by modifying the code to accept a larger threshold value, and by following more sensitive approaches in relation to defining the conversion points.

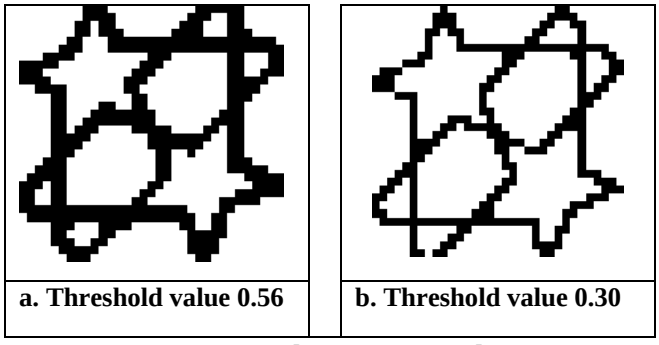


a. 200x200 pixel/unit, with 0.5 contrast threshold value.	b. 50x50 pixel/unit, with 0.3 contrast threshold value	c. 50x50 pixel/unit, with 0.55 contrast threshold value.

Figure 5.7 Final Results from Applying Program's Algorithm to the Same Geometric Unit of Different Resolutions and Contrast Threshold Values.

Based on the visual results, the problem with low-resolution images is related to the quality of line and intersection regions of the repeated unit. Figure 5.8 shows a low-resolution unit with two different contrast threshold values. As illustrated, many deficiencies are found around intersection areas. These are due to the following reasons: a) Distortion and loss of symmetry in areas of intersections. B) The high sensitivity level of intersections to any missing or additional pixels. C) The small distances between the different intersections. Small distances can affect program's performance by not satisfying some low threshold conditional values within program functions.

As a result of these deficiencies, the program either fails to read all intersection points; therefore producing incomplete data, or it identifies more intersection points than it should. To overcome theses obstacles, we can either employ a more sensitive approach for defining the conditional parameters of the different code functions, or by correcting image line quality through applying some of the morphological operations of binary images such as dilation and closing (Jahne 1991, Russ1992).



a. Threshold value 0.56 **b. Threshold value 0.30**

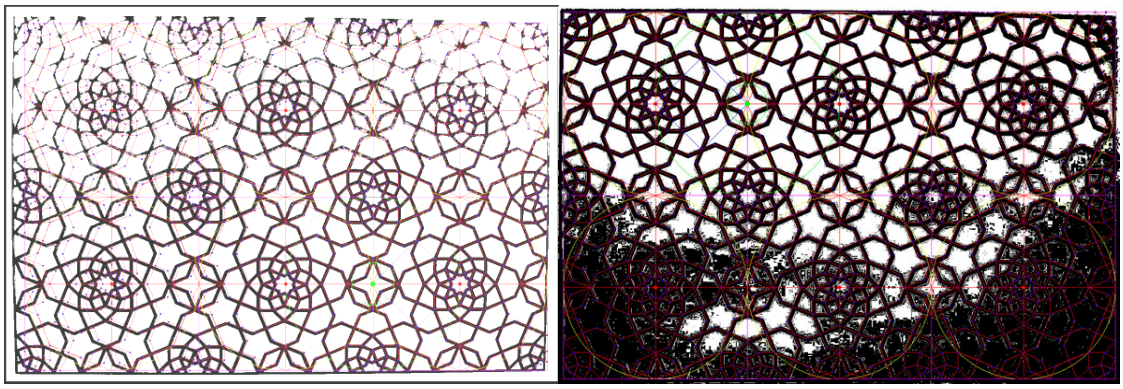
Figure 5.8 A Low-Resolution Unit with Two Different Contrast Thresholds.

It is also important to note that when repeating an experiment within the low-resolution ranges, results were not always consistent. These variations are linked to the different values of the contrast threshold. The problem usually occurs in the low threshold values because the line width is minimized to almost one pixel.

5.1.3 Third Set of Experiments

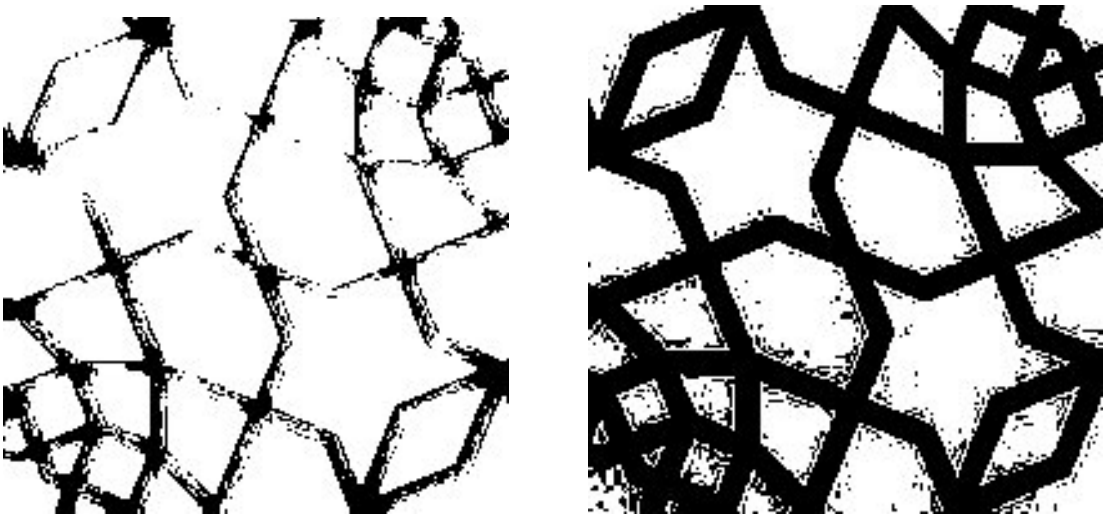
The third set of experiments is designed to test the effect of different slide bar values (segmentation threshold) on the final results. These experiments serve two purposes. The first is to define the best threshold range to w with these images and second to evaluate the program performance when dealing with noise or fainted lines (Figures 5.9, and 5.10). The goal of the second part is to touch on the problems of reconstructing real life objects and to speculate on ways to solve them.

The experiments were conducted using randomly selected complete patterns with good resolutions. Figures 5.11, and 5.12 display some chosen samples of repeat units with different contrast values.



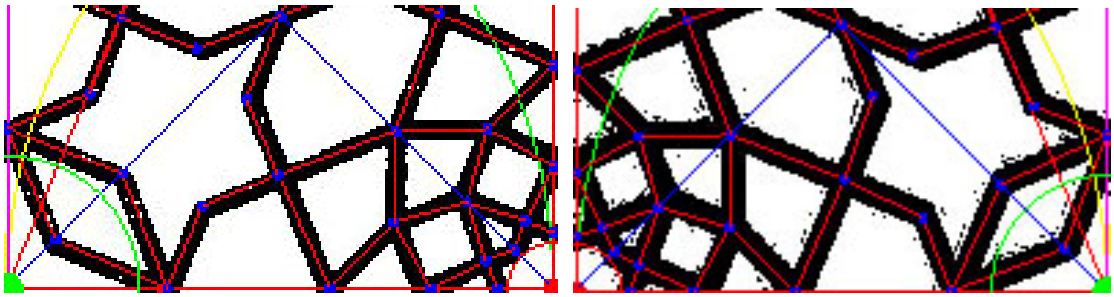
0.185 contrast value **0.670 contrast value**

Figure 5.9 The User Controls the Amount of Missing Data or Noise through Controlling Slide –Bar Contrast Values.



0.175 contrast value **0.670 contrast value**

Figure 5.10 Detailed Geometric Units of the Same Images Segmented Based on Different Contrast Threshold Values.



0.23 contrast value **0.570 contrast value**

Figure 5.11 Results of Applying Program's Algorithm to a Repeat Unit with Two Different Contrast Threshold Values (0.23 and 0.570).

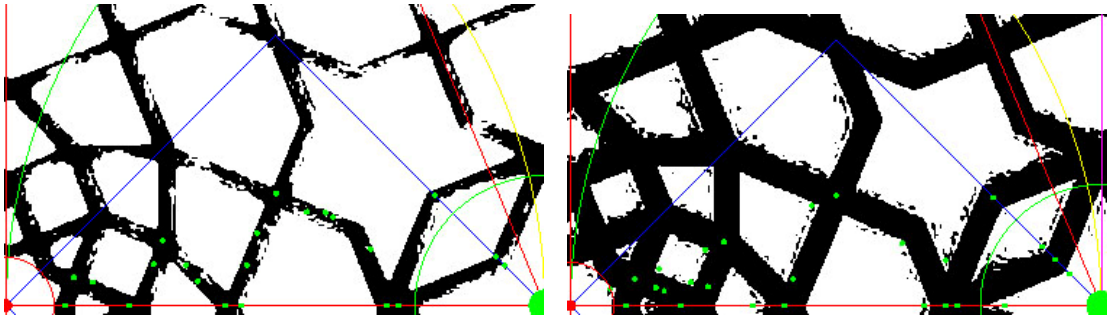
Results and Findings

1- Each pattern has a different ideal range of contrast thresholds, which depends mostly on image lighting conditions. This ideal range is generally between 0.045 and 0.55 for all tested patterns.

2- Pattern complexity did not have any effect on the results.

3- Although the best working values of contrast threshold are around 0.50, the working range can starts from 0.20 to 0,65. Figure 5.11 shows the results of some working values of the tested Pattern.

4- The program failed to produce any acceptable data when the contrast values were outside the working range. On one end, very low contrast threshold values can cause missing line data, and therefore, create a lot of cavities within pattern lines. As a result, the program identifies too many miss-located points as shown in figure 5.12. On the other end, high contrast values can cause too much noise. As a result, the program also identifies too many miss-located points as shown in figure 5.12. In both cases, the program was not able to function correctly. It crashed when trying to process the many identified points. To solve this problem, it is important to preprocess the image to eliminate noise or fixed holes. Also, it is important to modify the code to handle such defections.



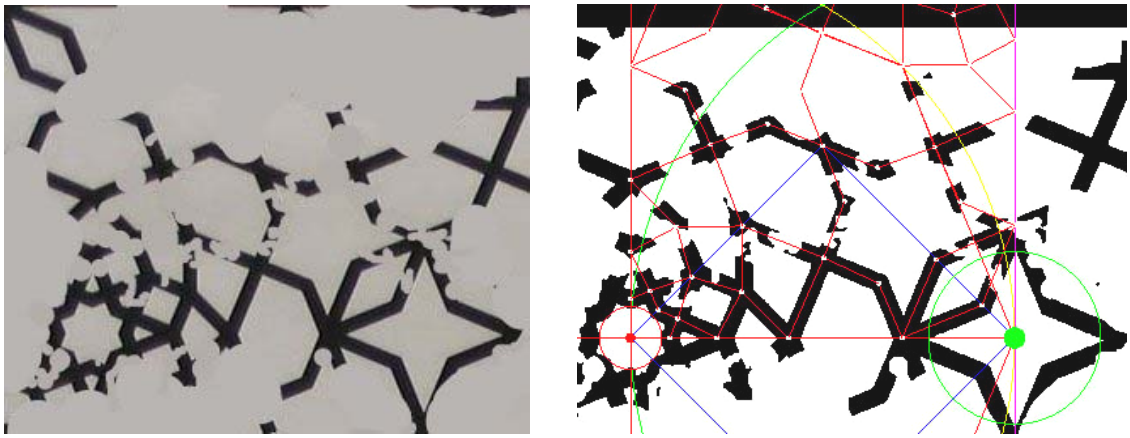
0.185 contrast value **0.70 contrast value**

Figure 5.12 Results of Applying Program's Algorithm to a Repeat Unit with Two Different Contrast Threshold Values (0.185 and 0.70).

5.1.4 Fourth Sets of Experiments

The fourth set of experiments is designed to test the performance of the program in reconstructing the right geometry of deteriorated patterns. It also tests the effects of different complexity on the out-put results. Experiment samples were chosen randomly from the different groups of complexity. The aim of this group of experiments is to test the effect of shape and location of deterioration on the structural accuracy and repeatability.

Figure 5.13 displays the original deteriorated image and the results of applying program's algorithms to the repeat unit. The main conditions for accurately defining the right intersection points are: smooth lines, clean intersections, and enough data of the repeat unit to generate the whole structure. Figure 5.14 displays the original deteriorated images and the reconstructed results of simple complexity patterns. Figure 5.15 displays the resulted reconstructed patterns of complex patterns. Figure 5.16 displays the effect of repeated unit location on the final reconstructions.



Original Image **0.50 threshold value**

Figure 5.13 The Original Input Image vs. the Results of Applying Program's Algorithm to a Deteriorated Repeat Unit.

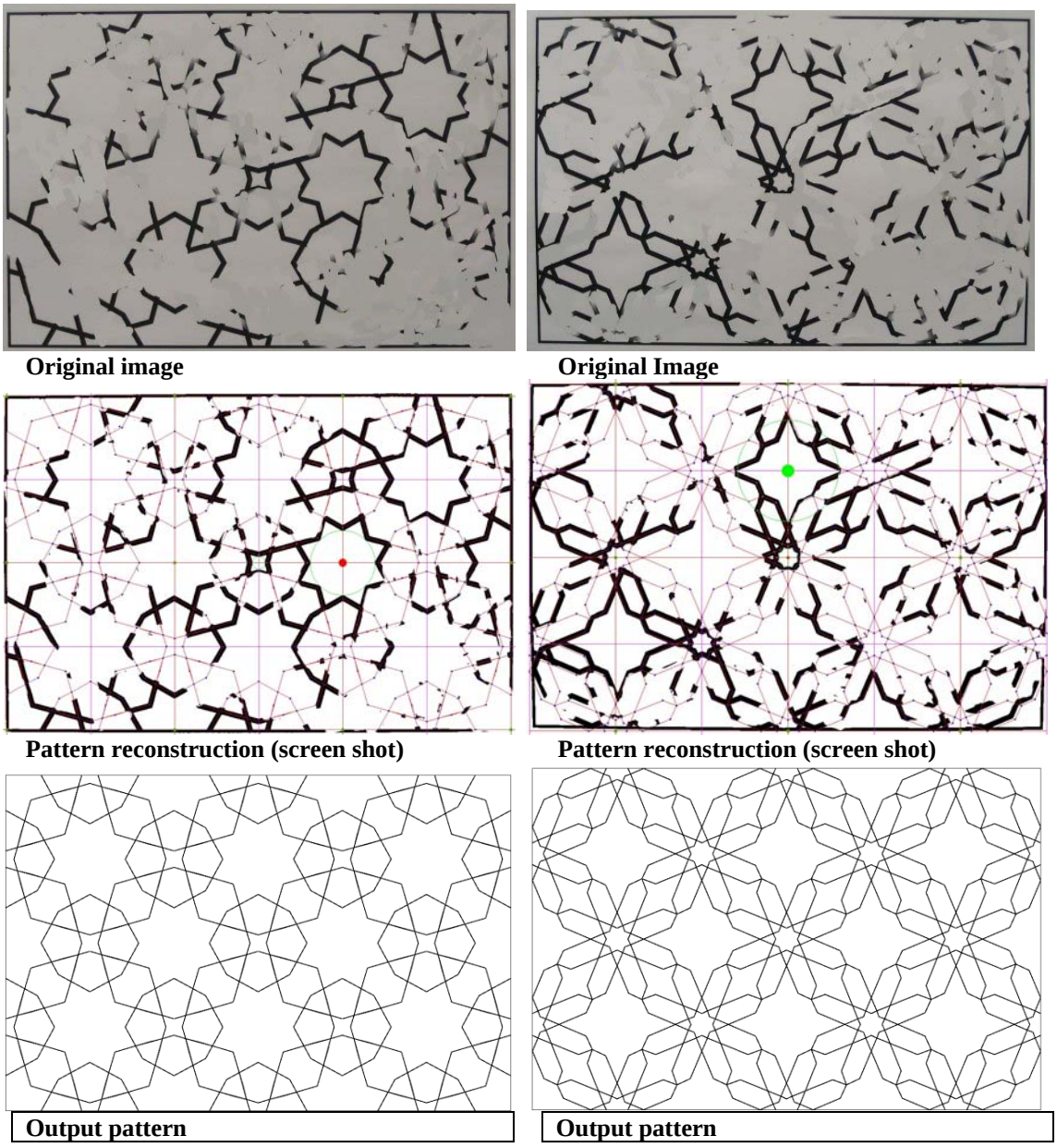
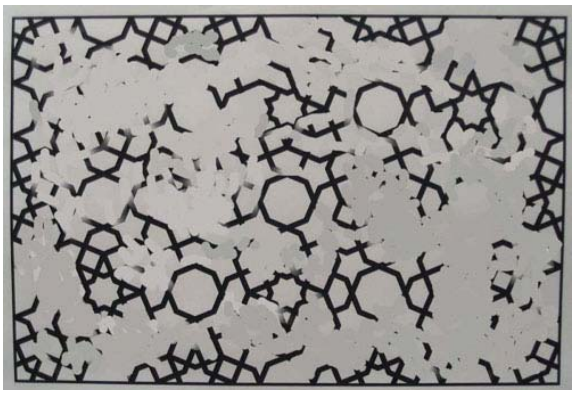
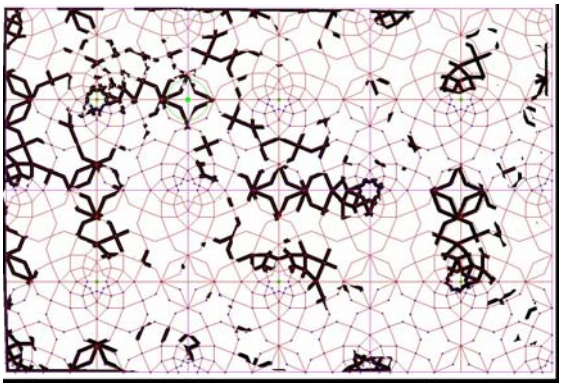


Figure 5.14 Two Original Deteriorated Patterns of Simple Complexity Level, Screen Shots of the Reconstruction Process and the Final Output Vector Line Data.

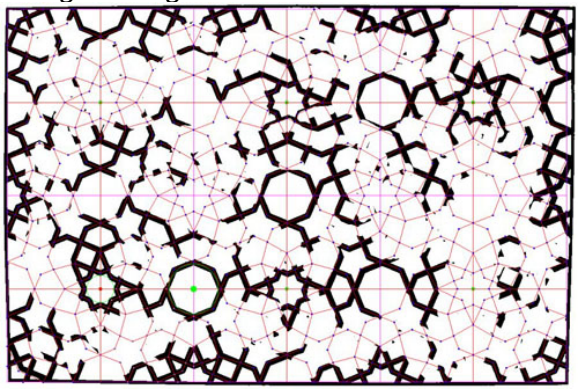
Original image



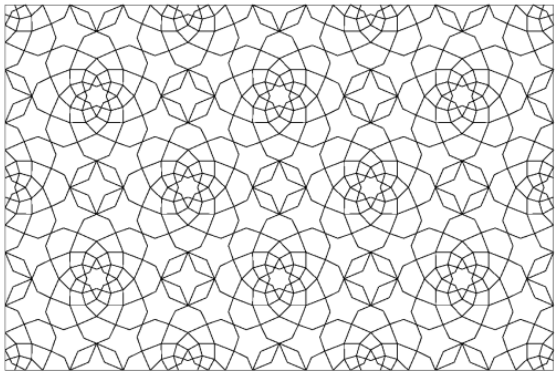
Original image



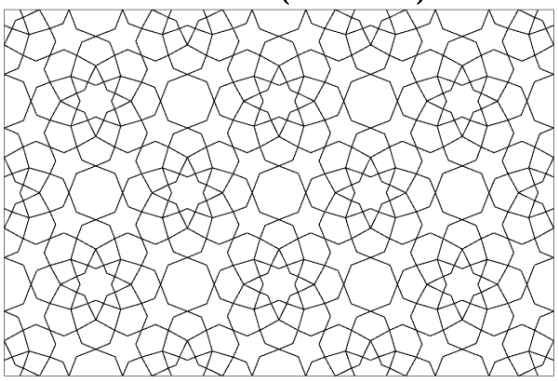
Pattern reconstruction (screen shot)



Pattern reconstruction (screen shot)

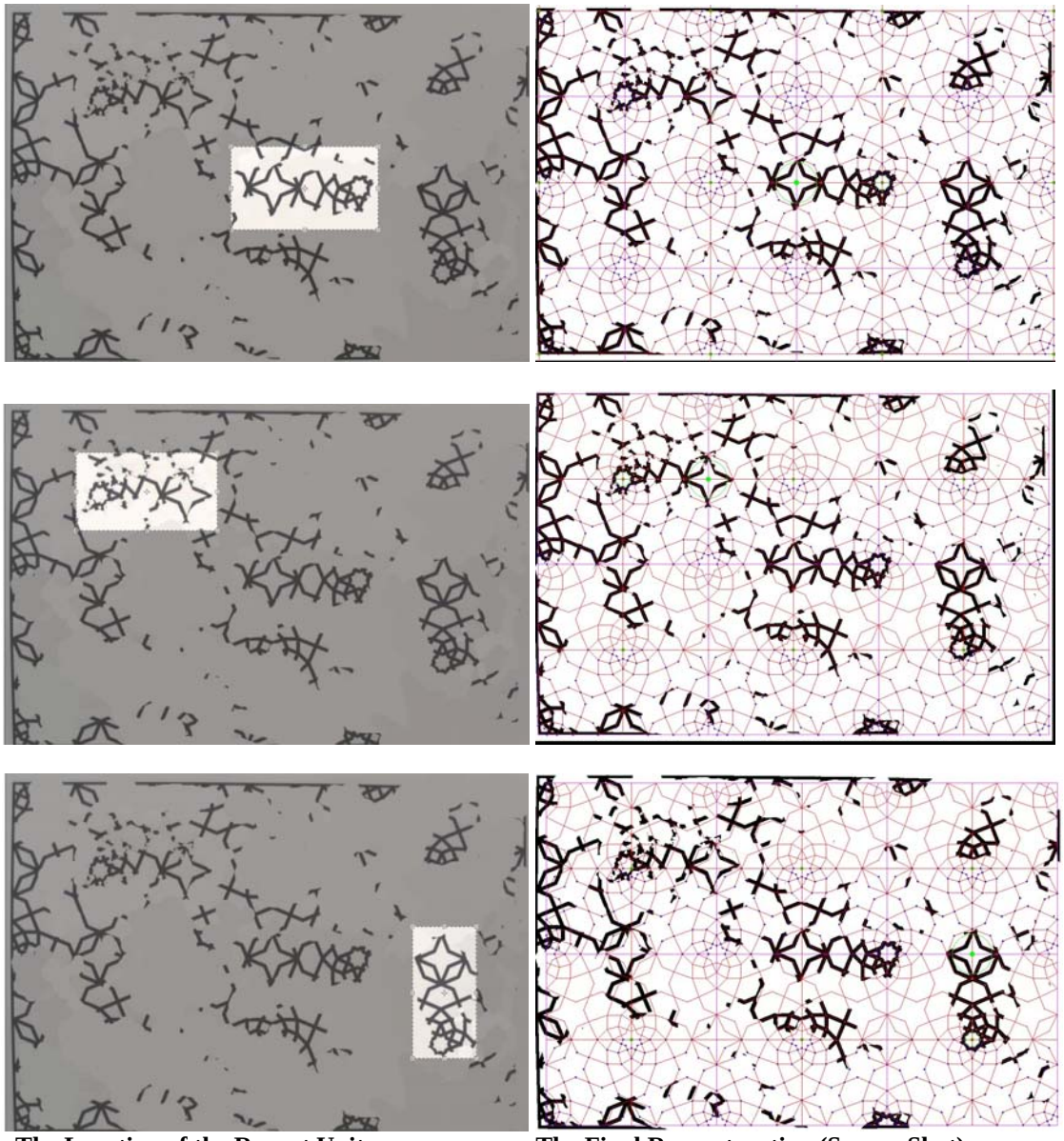


Output pattern



Output pattern

Figure 5.15 Two Original Deteriorated Patterns of High Complexity Level, Screen Shots of the Reconstruction Process and the Final Output Vector Line Data.



The Location of the Repeat Unit **The Final Reconstruction (Screen Shot)**

Figure 5.16 The Location of the Repeat Unit vs. the Final Reconstruction in Relation to the Original Data.

Results and Findings

1- No matter the degree of deterioration, the program is able to produce the whole pattern structure if enough information of one repeated unit is available. The performance of the program relies solely on finding one intact repeat unit.

2- Pattern complexity has no effect on the result.

3- According to the visual results, it seems that different shapes and locations of the repeated unit do not affect the structural accuracy of the final reconstructions. Although, the reconstructed lines are shifted from the original image (Figure 5.17), this shift is related to image distortion. Moreover, processing data located on the edges of the image might affect metric accuracy; theoretically the best location for producing the best metric accuracy should be in the middle. However, it is not within the scope of this research to study this variable.

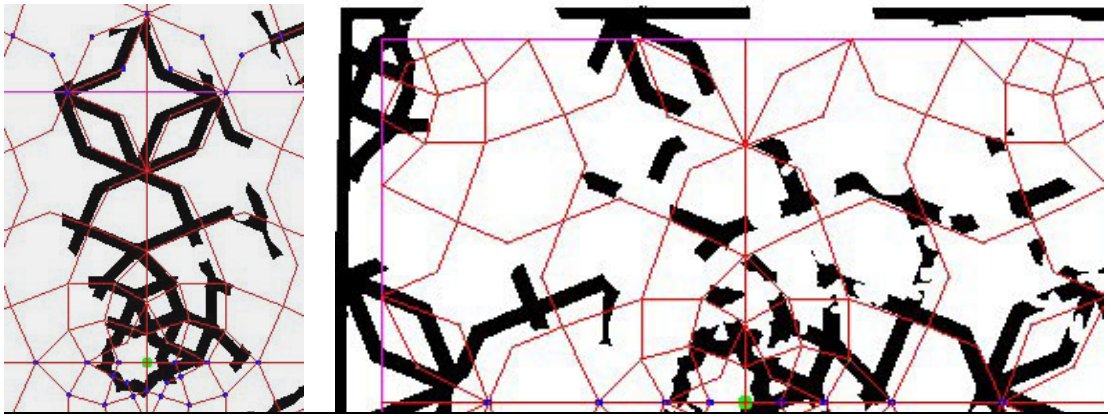


Table 5.17 Visual Result of the Shift Between the Original Image Data and the Reconstructed Lines. This shift is due to the Image Geometric Distortions.

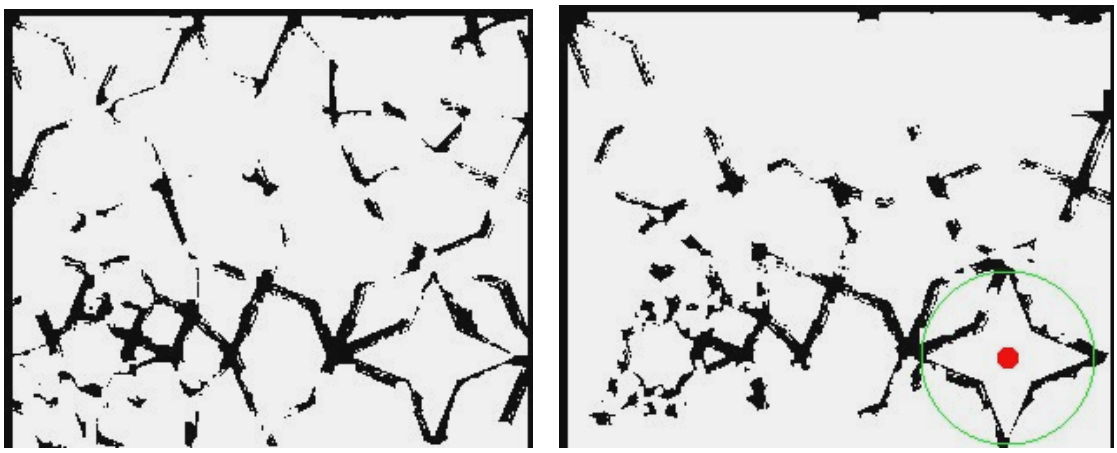
5.1.5 Fifth Set of Experiments

The fifth set of experiments is designed to test the effect of the different degrees of deterioration with slid bar variations (contrast threshold value) on the reconstructed repeat unit results. One sample from the high complexity group was chosen, and two degrees of deterioration were produced (figure 5.18.a, and 5.18.b). Lines of the repeated units were distorted by deleting or adding some noise using Photoshop or Gimp. The idea is to test the effect of the different contrast threshold values on the two images and define the boundary of that threshold. Moreover, it tests the ability of the program to

handle distorted line patterns, in order to define code limitations. Figures 5.19, 5.20 and 5.21 show the visual processing result of the two images with five different contrast values (0.200, 0.300, 0.500, 0.600, and 0.675).

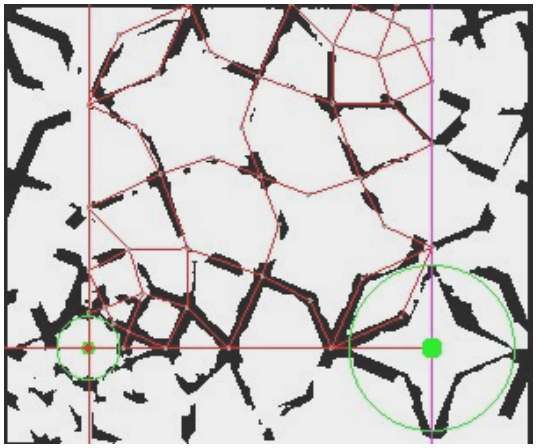
a. Deterioration degree _1 **b. Deterioration degree _2**

Figure 5.18 Two Degrees of Deterioration in the Input Image.

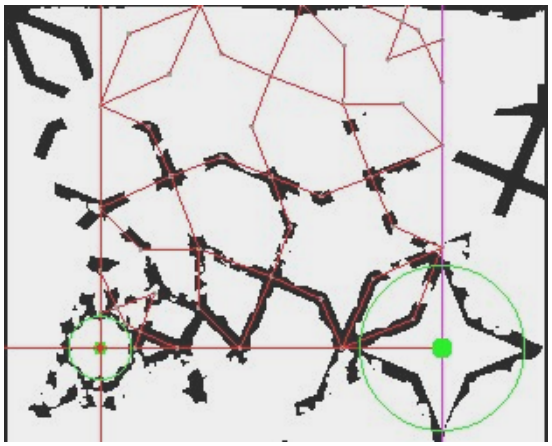


a. Contrast Value = 0.20 **b. Contrast Value = 0.20**

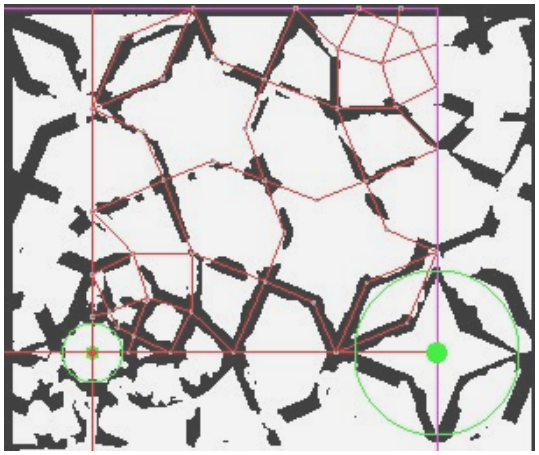
Figure 5.19 Visual Results of Applying Program's Algorithms to the Two Degrees of Deterioration and a Low Contrast Threshold Value of 0.20.



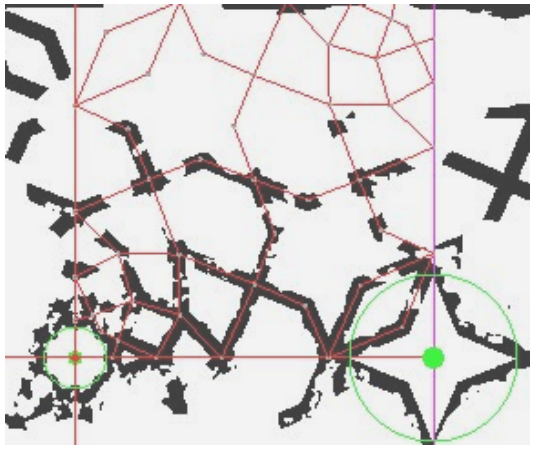
a. Contrast Value= 0.30



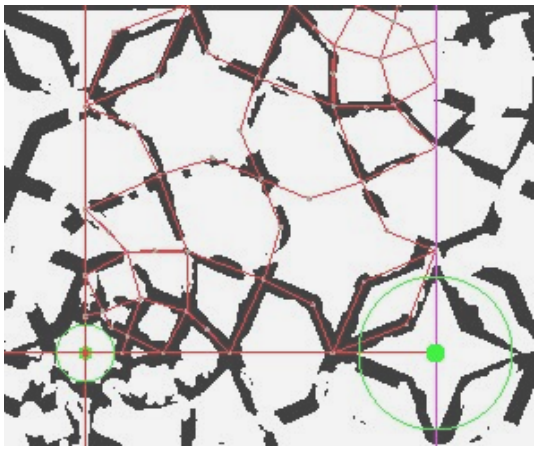
b. Contrast Value = 0.30



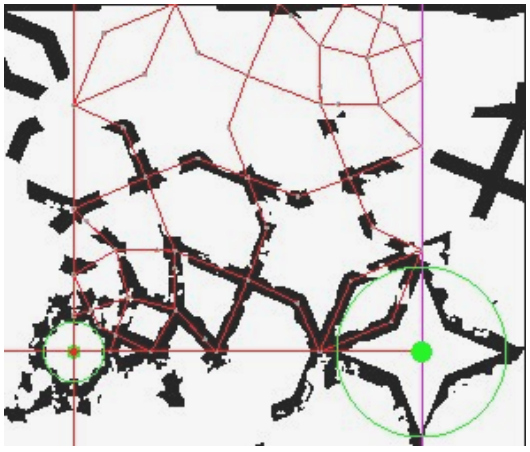
a. Contrast Value = 0.50



b. Contrast Value = 0.50

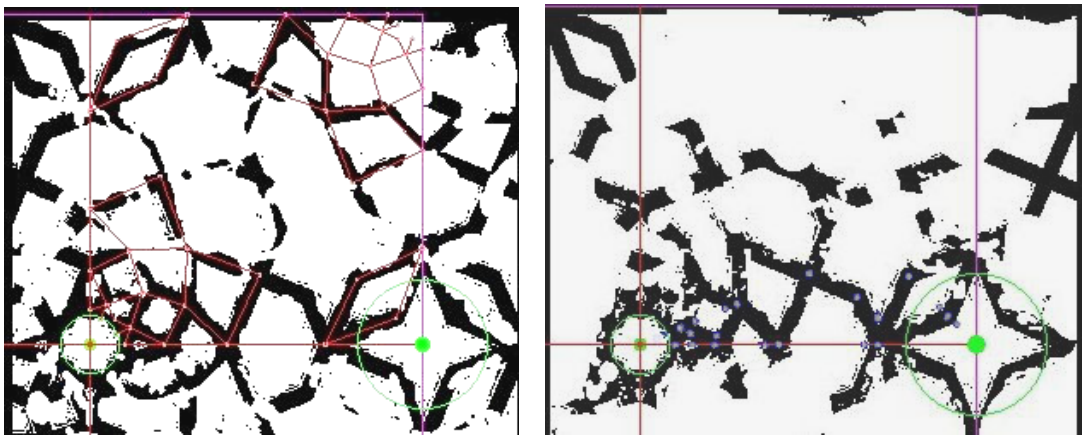


a. Contrast Value = 0.60



b. Contrast Value = 0.60

Figure 5.20 Visual Results of Applying Program's Algorithms to the Two Degrees of Deterioration and with Three Different Contrast Threshold Values (0.30, 0.50 and 0.60).



a. Contrast Value = 0.675 **b. Contrast Value= 0.675**

Figure 5.21 Visual Results of Applying Program's Algorithms to the Two Degrees of Deterioration and a High Contrast Threshold Value of 0.675.

Results and Findings

1- At the 0.200 contrast value, the program failed to define the two center points of the repeat unit, which stopped the process from moving to the next level (Figure 5.19). This failure was due to the symmetry distortion and missing data of the boundary around the center area.

2- The working values for both images were within the range of 0.30 to 0.60. The program was able to handle some degree of noise, but it failed to produce the right results outside this acceptable range (Figure 5.20).

3- At the 0.675 contrast value, the program was able to define the two center points but failed to define all the right intersection points. In the first case (Figure 5.21.a), the program was able to define most of the intersection points except for one. This was mostly due to angle distortion, and the problem with defining the conversion point at that intersection. In the second case (Figure 5.21.b), the program located more points than it needs to do, and as a result failed to generate the pattern. Moreover, it was not able to process these points and crashed while trying to do so.

4- Improving the program performance will require modifying the code to handle noise, angle distortion and missing pixels.

5.1.6 Sixth Set of Experiments

In all the previous experiments, the operator was assumed knowledgeable enough to operate the program and make the right selections. This set of experiment is designed to test the performance of the program when operated by a naive user. The aim of these tests is to define the boundaries and limitations related to the operation process.

The tasks required by the user are listed next:

1- Select the right contrast threshold value to produce a balanced black and white image. The program will not be able to perform properly if images are too faint or too noisy.
2- Select the two centers of the repeat unit. This is done by picking a point any where within the boundaries of the central region. The user needs to select the main center first, and then the secondary center. If the selection is not done in that sequence, the program will be unable to define the right intersection points as shown in Figure 5.22.

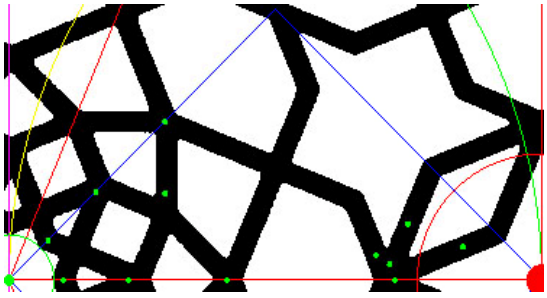


Figure 5.22 Visual Result of a Wrong User Selection Sequence.

3- The user needs to use logical judgment to select the right repeat unit (triangle area). It must contain enough information to regenerate the whole pattern. If the user picks a side that does not contain enough information, the program will only process what is there and will be unable to generate the whole pattern. This case is shown in Figure 3.23.

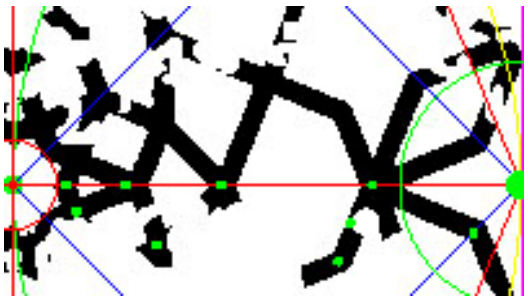
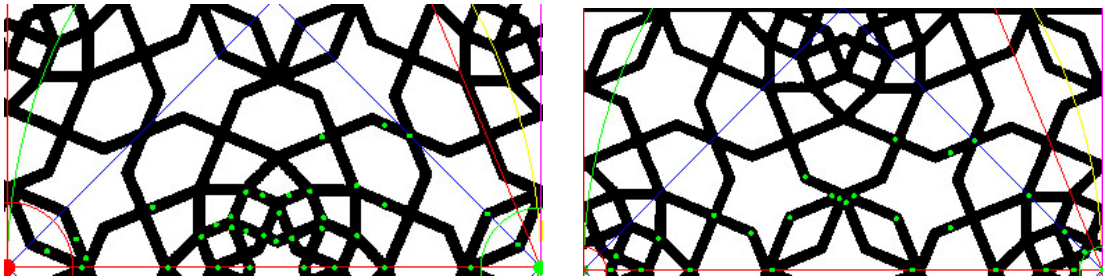


Figure 5.23 Visual Result of Choosing the Wrong Repeat Unit.

4- The operator needs to select a two adjusting centers, otherwise, the program will be unable to define the right repeat unit and as a result will not locate the required intersection points. Figures 5.24.a and 5.24.b display results of choosing the wrong centers.



a. **b.**

Figure 5.24 Visual Results of Choosing the Wrong Centers of the Repeat Unit.

CHAPTER VI
SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

6.1 Summary

For the purpose of cultural heritage preservation, the task of recording and reconstructing visually complicated architectural geometrical details is facing many practical challenges. Existing traditional technologies rely heavily on the subjective nature of our perceptual power in understanding its complexity and depicting its color differences. Such subjective input can greatly affect time, effort, and accuracy. Another difficulty is posed by the fact that archaeological reality often deals with patterns that are broken, incomplete or hidden. Computer vision is challenged to provide an interpretation of the available data, to simulate what is not there or cannot be seen.

Challenged by these tasks, this study proposes one possible solution, through utilizing digital techniques for recognizing; reconstructing and documenting detailed historical geometric patterns. Its main hypothesis is that digital techniques offer many advantages over the human eye in terms of recognizing subtle differences in light and color. It also has a great capacity for testing many scenarios quick, accurate and easy compared to the limitation of the human power.

The aim of this research is to explore the possibilities of using digital techniques for substituting human power in accomplishing tasks related to historic preservation. Specifically, it investigates the use of algorithmic mathematics and computer vision to develop new ways of digitally analyzing, reconstructing and documenting geometric patterns. This is done by addressing the question of how to incorporate the mathematical rules of pattern structure with digital image techniques using computer tools to develop a program capable of reading and interpreting partially deteriorated Islamic patterns represented in digital image format, to simulate the missing parts and produce an accurate virtual reconstruction. Although this research focuses specifically on Islamic

patterns, the underlying goal is to develop general ideas and principles that might then be applied to other ornamental styles.

An experimental approach is used to develop, test and evaluate specialized software for identifying deteriorated or incomplete archaeological geometrical patterns captured in digital images, and then restoring them digitally, for the purpose of producing accurate two-Dimensional reconstructed models. This is done through incorporating mathematical pattern rules into programming algorithms. The study is performed within the context of exploring new photogrammetric methods for historic preservation and cultural recording. Moreover, It is designed to tackle a very specific and idealized problem away from the complexities of exploring real life objects.

Research findings have proven the validity of this line of research and provided proof of the concept proposed in the hypothesis. This research concludes that digital techniques are a very powerful tool, which can be utilized to substitute the human perception in performing some tasks. This research hopes to draw more attention to the importance of utilizing new technologies -developed by other disciplines- in tackling practical problems related to historic preservation. It belongs to a broad line of research with an ambitious goal of developing tools that can take over some of the difficult and subjective tasks still performed by man. By incorporating some of the built-in intelligence into visualization technologies, we can create powerful tools that are able to perform tasks easily, quick and more efficient than the human operator.

6.2 Conclusions

Based on research results and findings, here is a list of conclusions that can be drawn from the study:

1- The proposed method is proven to be successful in capturing the accurate structural geometry of the deteriorated straight-line patterns generated based on the octagon-square basic grid. Nevertheless, a lot needs to be done to improve its performance to deal with real life objects. The developed tool is not assumed to be professional software; rather, it only provides a solution to the idealized problem. This represents

a first step in solving real life challenges. Dealing with real life objects requires applying more sophisticated image processing techniques in order to prepare it for pattern recognition.

2- The general goal for this research is to develop a method that can be applied to reconstruct all types of Islamic Geometric Patterns. Therefore, based on the fact that all Islamic patterns share the same general properties (El-Said 1993, Jones 1978, Gonzalez 2001), it is possible to apply the same conceptual method to reconstruct all two-dimensional straight-line Islamic geometric patterns. Nevertheless, to be able to apply the method to work on patterns generated based on different base grids, algorithms need to be modified to identify the exact pattern' repeated unit in relation to its basic grid properties.

3- The proposed method can also be modified to work with other types of Islamic geometric patterns that have special characteristics. Here I present some of these patterns and ways to modify the method.

- Geometric patterns that have curved lines or free forms.

  To work with these patterns, we need to modify the way to link the defined intersections. After locating intersection points we need to map the curved lines. This can be done by using available skeletonization techniques. The Medial Axis Transform (MAT) method, which was proposed by Blum (1964) (Tomita and Tsuji1990) represents a good candidate. The medial axis for an area is the locus of the centers of the largest circles that can be located at a given point in that area (Snyder and Qi 2004).

- Geometric patterns covering three-Dimensional surfaces (domes, niches, etc.,).

  To work with these patterns we need to incorporate the idea of 3D projection metrics, which correct the geometric distortions of the pattern. Repeated unit line recognition techniques are kept the same.

- Geometric patterns which are rendered as colored areas.

  To identify geometry of this rendering style we need to apply different approaches for segmentation. Region-based segmentation (Jahne 1991) can be used first, and then mapping the boundaries of different regions as the base for extracting line information.
- In the case of geometric patterns that are extremely deteriorated and do not have enough information to generate the rest of the pattern, the proposed method simply will not work.

4- Many pattern systems can be good candidates to develope methods for automatically recreating their structures by applying the same methodology. The automation of any pattern system requires a clear structured hierarchy that can be translated into mathematical formulas, a clear visual appearance of the patterns, to make it possible to digitally identify pattern information, and a defined vocabulary of shapes.

5- The designed digital tool proved to be efficient in capturing the correct geometry of low-resolution digital images. What this means for real life objects, is that we don't need to get too close to the target to get a close-up image. Resolution starting form (200 pixels/image) to (600 pixel/image) is enough to process almost all patterns. Moreover, this also means that digital techniques are more capable of recognizing complex geometry than the human eye. For the task of recognizing structured scenes, image shapes and distances, computer vision is considered best for quantitative measurements (Zuech 2000). Moreover, it is proven that computer vision is capable of dealing with two-Dimensional interpretations of well-defined patterns than three-Dimensional, in which the human eye is highly developed (Zuech 2000). In terms of performance; computer vision is best for working with high redundancy patterns. When dealing with geometrical patterns the human vision can be easily subject to many errors and optical illusions (Russ 1992).

6- Image perspective distortion does not affect the structural accuracy of the final results. On the contrary, geometric distortion can be corrected using output reconstruction. This is accomplished by stretching the edges of the image to meet the corners of the reconstructed pattern. Nevertheless, extreme rotation can affect the accuracy of the final results.

7- For dealing with Islamic Geometric patterns, methods such as the one proposed by this study can prove to be more efficient than any existing documentation method; it is fast, accurate, and an efficient tool for understanding pattern structure and analysis.

6.3 Recommendations

1- This study is designed to deal with very controlled samples. It does not lend itself to examine problems related to real life objects. Images of real world objects absorb many external effects when sensed through images. The challenge for computer algorithms is to extract the essence of objects. Problems related to photographing real life objects can mostly be attributed to the effect of the environmental factors on the targeted object. These effects include shade, shadow, noise, lighting, geometric distortions, etc. To be able to process such images using the proposed method; future research should be directed towards examining effective techniques of image processing of structural patterns. Image processing is used for improving the visual appearance of images and preparing the image for feature management and pattern interpretation.

2- The effort of this research is not directed toward developing professional software. Nevertheless, it represents the first step in that direction. The conceptual framework proposed by this study can serve as a platform for developing professional softwares related to historic documentation. It also can help to modify available softwares by incorporating a more problem-oriented approach.

3- This research is concerned with the qualitative testing of structural accuracy and repeatability. Future research should test the metric accuracy of similar tools. The aim of testing the metric accuracy is to define the margin of error that occurs when using these tools. It also helps to define areas of defects in the code, which can be modified to get better results.

4- The semi-automated tool developed by the research provides a first step into developing a fully automated process. Full automation is possible based on the results of this study. Here I propose a conceptual framework for fully automate this process:

a) Read image and automatically segment it using the 0.50 contrast threshold value. Based on this study, 0.50 is the ideal value for segmenting all patterns.

b) Scan the black and white image to do the following:

- Locate all symmetrical shapes in the image. Categorize all these shapes based on the number of symmetrical sides and the radius of internal area. Finally, based on patterns' rules, define and mark the main centers and the secondary one.
- Save line information using a run-length encoding. Use this encoding to find the line width using image histogram.

c) Find the repeat unit:

- For each main center, find all the adjusting secondary centers that satisfy patterns' rules.
- For each pair of centers, define the boundaries of the repeat unit and define the two rectangular areas as candidates.
- Scan the two rectangular areas and compare the number of run-length segments that have line width.

- The repeat units with the highest number of run-length segments are marked as the main candidates for intersection processing.

d) Find intersection points. Follow the same steps taken by this research.

e) Generate the rest of the structure also by following the same steps used in this research.

5- Current applications in the field of Historic recording are mainly focused on using visualization and image processing techniques. Nevertheless, not much has been done on utilizing machine vision technologies. Future research should be directed more towards developing artificial intelligence and pattern recognition techniques that have the ability to substitute human power in accomplishing difficult tasks. Today, the advancement in the filed of digital pattern recognition techniques make it possible to recognize virtually any pattern structure.

BIBLIOGRAPHY

Abas, S. J. “Islamic Geometrical Patterns for the Teaching of Mathematics of Symmetry.” *Symmetry in Ethnomathematics* 12, no.1-2 (2001): 53-65. Budapest, Hungary: International Symmetry Foundation. Available online at: http://www.ethnomath.org/resources/abas2001.pdf.

Abas, S. J. and A. Salman. “Geometric and Group-theoretic Methods for Computer Graphic Studies of Islamic Symmetric Patterns.” *Computer Graphics Forum* 11, no. 1 (1992): 43-53. Available online at: http://www.blackwell-synergy.com/links/doi/10.1111/1467-8659.1110043/abs/.

Al-Ajlouni, R. *Islamic Geometry*. (Three courses on Islamic Geometry). Al Balqa University, Jordan, 1999.

Al-Bayati, B. *Process and Pattern: Theory and Practice for Architectural Design in the Arab World*. London: Flexiprint Ltd, 1981.

Alexander, H. “The Computer/Plotter and the Seventeenth Ornamental Design Types.” In *Proceedings of SIGRAPH’75*, 160-167. New York: ACM, 1975.

Aliaga, D. G., D. Yanovsky and L. Carlbom. “Sea of Images.” *Computer Graphics and Applications* 23, no. 6 (2003): 22-30.

Al-Sayyed, H. “Decoration, the Main Art of Islamic Civilization.” In *Islam Online,*1999. Available online at:: http://www.islamonline.net/IOL-English/dowalia/art-2000-july-27/art2.asp.

Amenta, A. and M. Phillips. “Kali.” 1996. Available online at: http://www.geom..umn.edu/java/Kali/.

Arcelli, C. and M. Frucci. “Reversable Skeletonization by (5,7,11)-Erosion.” In *Visual form : Analysis and Recognition*, 21 - 28. Edited by C. Arcelli, L. Cordella and G. Baja. New York: Plenum Press, 1992.

Arcelli, C., B. Sanniti and P. Kwok. “Parallel Pattern Compression by Octagonal Propagation.” In *Thinning Methodologies for Pattern Recognition*, 113-137. Edited by C. Y. Suen and P. S. P. Wang. London: World Scientific Publishing Ltd., 1994.

Argialas, D.P. and O. D. Mavrantza. “Comparison of Edge Detection and Hough Transform Techniques for the Extraction of Geologic Features.” In *Geo-Imagery Bridging Continents. Proceedings Volume, Vol.XXXV Congress***,** 790. Edited by O.

Altan. Istanbul: IAPRS, 2004. Available online at: http://www.isprs.org/istanbul2004/comm3/papers/376.pdf

Atkinson, K. B. *Development in Close Range Photogrammetry.* (1st ed.). London: Applied Science Publisher Ltd, 1980.

Avern, J. "A New Technique for Recording Archaeological Excavations: Research Progress Report." In *Computing Archaeology for Understanding the Past. CAA 2000,* 3-7. Edited by Z. Stancic and T. Veljanovski. Oxford: Archaeopress, 2001.

Barcelo, J. A."Virtual Archaeology and Artificial Intelligence." In *Virtual Archaeology: Proceedings of the VAST Euroconference, Arezzo 24-25 November2000,* 21-28. Edited by F. Niccolucci. Oxford: Archaeopress Press, 2002.

Beals, S. (2001). "The Vocabulary of Color." 2001. Available online at: http://ep.pennnet.com/Articles/Article_Display.cfm?Section=Articles&SubSection=Display&ARTICLE_ID=95244&VERSION_NUM=1.

Beraldin, J., F. Blais, L. Cournoyer, G. Godin, M. Rioux and J. Taylor. "Active 3D Sensing for Heritage Applications." In *[Enter the Past]The E-way into the Four Dimensions of Cultural Heritage. Computer Applications and Quantitative Methods in Archaeology. Proceedings of the 31st Conference,* Vienna, Austria, April 3003, 340-343. Edited by M. Wien, R. Erbe and S. Wien. Hampshire, England: Basingstoke Press, 2004.

Bose, T. *Digital Signal and Image Processing*. New York: John Wiley & Sons, Inc., 2004.

Bowen, B. and H. Weisberg. *An Introduction to Data Analysis*. San Francisco: W. H. Freeman and Company, 1980.

Bowkett, L., S. Hill, D. Wardle and K. A.Wardle. *Classical Archaeology in the Field: Approaches*. Bristol, England: Bristol Classical Press, 2001.

Campo, P. "Computer…What for? Virtuality vs Reality in Archaeology." In *[Enter the Past]The E-way into the Four Dimensions of Cultural Heritage. Computer Applications and Quantitative Methods in Archaeology,.Proceedings of the 31st Conference,* Vienna, Austria, April 3003, 192-195. Edited by M. Wien, R. Erbe and S. Wien. Hampshire, England: Basingstoke Press, 2004.

Castle, D. "Critical Assessment and Future Directions." In *Experimental Software Engineering Issues: Proceeding of International Workshop,* Dagstuhl Castle, Germany, September 14-18, 1992. Edited by H. Dieter Rombach, Victor R. B. and Richard W. S. Berlin, New York: Springer-Verlag, 1992.

Castleman, K. *Digital Image Processing*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1996.

Chundy, L. and V. Chundy. "Neural-Like Thinning Processing." In *Lecture Notes in Computer Science: Computer Analysis of Images and Patterns. Proceedings of the 7th International Conference, CAIP '97,* Kiel, Germany, September 10-07, 243 – 550. Edited by G. Sommer, K. Daniilidis and J. Pauli. Berlin: Springer-Verlag, 1997.

Clevenot, D. *Splendors of Islam: Architecture, Decoration and Design*. New York: The Vendom Press, 2000.

Dallas, R. " Architecture and Archaeological Rcording." In *Development in Close Range Photogrammetry,* 81-116. Edited by K. Atkinson. London: Applied Science Publishers Ltd., 1980.

Danby, M. *Moorish Style.* London: Phaidon Press Limited, 1995.

Dispot, F. *Arabeske 2.1.1.* 2004, Available online at: www.wozzeck.net/arabeske/index.html.

Doneus, M and W. Neubauer. "Digital Recording of Stratigraphic Exacavations." In *[Enter the Past]The E-way into the Four Dimensions of Cultural Heritage. Computer Applications and Quantitative Methods in Archaeology. Proceedings of the 31st Conference,* Vienna, Austria, April 3003, 113-116. Edited by M.Wien, R. Erbe and S. Wien. Hampshire, England: Basingstoke Press, 2004.

Drap, P., A.Virnich and P. Grussenmeyer. "Photogrammetric Stone-by-Stone Survey and Archaeological Knowledge: An Application on the Romanesque Priory Church Notre-Dame d'Aleyrac." In *Virtual Archaeology. Proceeding of the VAST Euroconference,* Arezzo 24-25 November,139-145. Edited by F. Niccolucci. Hampshire, England: Basingstoke Press, 2002.

Earl, G. and D. Wheatley. "Virtual Reconstruction and the Interpretive Process: a Case-Study from Avebuty." In *Contemporary Themes in Archaeological Computing*, 5-15. Edited by D. Wheatley, G. Earl and S. Poppy. Oxford: Oxbow Books Limited, 2002.

El-Said, E. *Islamic Art and Architecture: The System of Geometric Design.* (1st ed.). Reading, England: Garnet Publishing Limited, 1993.

Forte, M. *Introduction to Virtual Archaeology: Great Discoveries Brought to Life Through Virtual Reality*, 8 -13 . Edited by M. Forte, M. & S. AlbSiliotti. London: Thames and Hodson, 1997.

Gauch, J. “Segmentation and Edge Detection.” In *The Colour Image Processing Handbook*, 163 -187. Edited by S. Sangwine and R. Horne. London: Chapman & Hall, 1998.

Ghosh, Sanjib K. *Analytical Photogrammetry*. New York: Pergamon Press, 1979.

Gibilisco, S. *Optical Illusions: Puzzles, Paradoxes and Brain Teasers # 4*. Blue Ridge Summit, Pennsylvania: TAB Books, 1990.

Glassner, A. “Frieze Groups.” In *Andrew Glassner's Notebook. IEEE Computer Graphics & Applications*, 1996. Available online at: http://www.glassner.com/andrew/cg/cga/cga1996.htm.

Glassner, A. “Upon Reflection.” In *Andrew Glassner's Notebook. IEEE Computer Graphics & Applications*, 1998. Available online at: http://www.glassner.com/andrew/cg/cga/cga1998.htm.

Glassner, A. “Texturing With Symmetry.” In *Andrew Glassner's Notebook. IEEE Computer Graphics & Applications*, 2000. Available online at: http://www.glassner.com/andrew/cg/cga/cga2000.htm.

Goncalves, A. and A. Mendes. “The Rebirth of a Roman Forum- the Case Study of the Flavian Forum of Conimbirga.” In *[Enter the Past]The E-way into the Four Dimensions of Cultural Heritage. Computer Applications and Quantitative Methods in Archaeology. Proceedings of the 31st Conference*, Vienna, Austria, April 3003, 510-513. Edited by M. Wien, R. Erbe and S. Wien. Hampshire, England: Basingstoke Press, 2004.

Gonzalez, V. *Beauty and Islam: Aesthetic in Islamic Art and Architecture*. London: Islamic Publications Ltd, 2001.

Gool, L., M. Pollefeys, M. Proesmans and A. Zalesny. “The Murale Project:Image-Based 3D Modeling for Archaeology.” In *Virtual Archaeology. Proceeding of the VAST Euroconference, Arezzo 24-25 November 2000*, 53-63. Edited by F. Niccolucci. Oxford, England: Archaeopress Press, 2002.

Groat, L. and D. Wang. *Architectural Research Methods*. New York: John Wiley & Sons, Inc., 2002.

Grube, E. “What is Islamic Architecture?.” In *Architecture of the Islamic World, Its History and Social Meaning*, 10-14. Edited by G. Michell. London: Thames & Hudson Ltd., 1978.

Grunbaum, B. and G. Shephard. *Tilings and Patterns*. New York: W. H. Freeman and Company, 1987.

Grussenmeyer, P., K. Hanke and A. Streilein. "Architectural Photogrammetry: Basic Theory, Procedures, Tools**."** In *Digital Photogrammetry*, 300-339. Edited by M. Kasserand, Y. Egels. Corfu, Greece: Taylor & Francis, 2002. Available online at: http://www.isprs.org/commission5/tutorial/gruss/tut_gruss.pdf.

Guidazzoli, A., M. Mauri, R. Salvi and M. C. Liguori. "Augmented Culture: Historical Environments for Captivating Educational TV Programs." In *[Enter the Past]The E-way into the Four Dimensions of Cultural Heritage. Computer Applications and Quantitative Methods in Archaeology. Proceedings of the 31st Conference,* Vienna, Austria, April 3003, 183-186. Edited by M. Wien, R. Erbe and S. Wien. Hampshire, England: Basingstoke Press, 2004.

Guo, H., Y. Xiao and H. Yan. "Text String Extraction in Delaunay Triangulation." In *Digital Image Computing: Techniques and Application. Proceedings of the VIIth conference,* 319-327. Edited by C. Sun, H. Talbot, S. Qurselin and T. Adriaansen. Sydney: The University of Queensland, 2003. Available online at: http://www.cmis.csiro.au/Hugues.Talbot/dicta2003/cdrom/pdf/0319.pdf.

Hanke, K. "Accuracy Study Project of Eos Systems' PhotoModeler." *Architectural Photogrammetry (CIPA),* 2000. Available online at: http://www.photomodeler.com/pdf/hanke.pdf

Harrison. B. and D. Jupp. *Introduction to Image Processing*. Melbourne: CSIRO Print Advisory Service, 1990.

House, D. H. "The Digital Image." *VIZA 654/Cpsc 646 Course Notes*. Visualization Laboratory, College of Architecture. Texas A&M University, 2002. Available online at: http://www-viz.tamu.edu/courses/viza654/03fall/book.pdf.

Hu, G. and Z. Li. "An X-Crossing Preserving Skeletonizatiom Algorithm." In *Thinning Methodologies for Pattern Recognition,* 67-89. Edited by C. Y. Suen and P. S. P. Wang. London: World Scientific Publishing Ltd., 1994.

Hu, J., S. You and U. Neumann. "Approaches to Large-Scale Urban Modeling." *Computer Graphics and Applications* 23, no. 6 (2003): 62-67.

Iakovlva, L. "Computerized Methods for Palaeolithic Art Studies." In *[Enter the Past]The E-way into the Four Dimensions of Cultural Heritage. Computer Applications and Quantitative Methods in Archaeology. Proceedings of the 31st Conference*, Vienna, Austria, April 3003, 348-352. Edited by M. Wien, R. Erbe and S. Wien. Hampshire, England: Basingstoke Press, 2004.

Jablonka, P. “Reconstructing Sites and Archives: Information and Presentation Systems.” In *[Enter the Past]The E-way into the Four Dimensions of Cultural Heritage. Computer Applications and Quantitative Methods in Archaeology. Proceedings of the 31st Conference*, Vienna, Austria, April 3003, 281-285. Edited by M. Wien, R. Erbe and S. Wien. Hampshire, England: Basingstoke Press, 2004.

Jahne, B. *Digital Image Processing: Concepts, Algorithms and Scientific Applications.* Berlin: Springer-Verlag, 1991.

Jairazbhoy, R. A. *Islamic Architecture*. Lahore, Pakistan: Ferozons (Pvt.) Ltd, 2000.

Jimenez, D. and D. Chapman. “An Application of Proximity Graphs in Archaeological Spatial Analysis.” In *Contemporary Themes in Archaeological Computing*, 90-99. Edited by D. Wheatley, G. Earl and S. Poppy. Oxford, England: Oxbow Books Limited, 2002.

Jones, D. “The Elements of Decoration: Surface, Pattern and Light.” In *Architecture of the Islamic World. Its History and Social Meaning*, 144-175. Edited by G. Michell. London: Thames & Hudson Ltd., 1978.

Kampel, M. and R. Sablatnig. “Computer Aided Classification of Ceramics.” In *Virtual Archaeology. Proceeding of the VAST Euroconference, Arezzo 24-25 November,* 77-82. Edited by F. Niccolucci. Oxford, England: Archaeopress Press, 2002.

Kaplan, C. “Computer Generated Islamic Star Patterns**.**” 2000. Available online at: http://www.cgl.uwaterloo.ca/~csk/washington/taprats/bridges2000.pdf.

Kaplan, C. “Islamic Star Patterns from Polygons in Contrast.” In *GI '05: Proceedings of the 2005 Conference on Graphics Interface*, 2005. Available online at: http://www.cgl.uwaterloo.ca/~csk/papers/kaplan_gi2005.pdf

Kaplan, C. and D. Salesin, D. *Islamic Star Patterns in Absolute Geometry*. New York: ACM Press, 2004 . Available online at: http://portal.acm.org/citation.cfm?id=990002.990003.

Kasturi, R., R. Raman, C. Chennubhotla and L. O’Gorman. “An Overview of Techniques for Graphics Recognition.” In *Structured Document Image Analysis*, 285- 324. Edited by H. Baird, H. Bunke and K. Yamamoto. Berlin: Springer-Verlag, 1992.

Kovesi, P. “Phase Congruency Detects Corners and Edges.” In *Digital Image Computing: Techniques and Application. Proceedings of the VIIth Conference,* 10-12 Dec. 2003, 309- 318. Edited by C. Sun, H. Talbot, S. Qurselin and T.

Adriaansen. Sydney: The University of Queensland, 2003. Available online at: http://www.csse.uwa.edu.au/~pk/research/pkpapers/phasecorners.pdf.

Kritchlow, K. *Islamic Patterns: An Analytical and Cosmological Approach*. New York: Thames & Hudson Inc, 1976.

Latecki, L. "Sliding Window Filters and Edge Detection." In *Computer Graphics and Image Processing, CIS 601, Department of Computer and Information Sciences.* Temple: Temple University, 2003. Available online at: http://www.cis.temple.edu/~latecki/CIS601-03/Lectures/EdgeDetection03.ppt.

Little, J. "Edge Detection Methods." In *UBC course CPSC 505 Image Understanding I: Image Analysis.* Department of Computer Science. University of British Columbia, 2004. Available online at: http://www.cs.ubc.ca/spider/little/505/lectures/subsection3_10_5.html.

Lock, G. and T. Harris. "Return to Ravello." In *Beyond the Map: Archaeology and Spatial Technologies*. Edited by G. Lock. Amsterdam: IOS Press, 2000.

Luckiesh, M. *Visual Illusions: Their Causes, Characteristics and Applications*. New York: Dover Publications, Inc., 1965.

Matkovic, K., L. Neumann, J. Siglaer, M. Kompast and W. Purgathofer. "Visual Image Query." In *ACM International Conference Proceeding Series. Proceedings of the 2nd International Symposium on Smart Graphics, June 11-13,2002*,116-123. Hawthorne, New York: ACM Press, 2002. Available online at: http://portal.acm.org/citation.cfm?id=569022.

Mikhail, E. M. *Modern Photogrammetry*. (1st ed.). New York: John Wiley & Sons, Inc., 2001.

Mikhail, E., J. Bethel and J. McGlone. *Introduction to Modern Photogrammetry*. New York: John Wiley & Sons, Inc, 2001.

Montesinos, M. *Classical Tessellations and Three Manifolds*. Berlin: Springer-Verlag, 1985.

Mostafa, M. *The Museum of Islamic Art*. (1st ed.). Cairo: Ministry of Education Press, 1955.

Mutz, A. "Digital Photography Fundamentals and Ternds". Codesta, 2003. Available online at: http://www.codesta.com/knowledge/technical/digital_photography/printable_version.aspx.

Nodelman, S. “Structural Analysis in Art and Anthropology.” In *Structuralism*, 79-93. Edited by J. Ehrmann. New York: Anchor Books, 1970.

Oh, W., S. Han, H. Yoon, Y. Bae and S. Song. “3d Digital Modeling of Modern Times Building for Preservation and Restoration.” In *Geo-Imagery Bridging Continents. Proceedings Volume: IAPRS, Vol.XXXV Congress*, 12-23 July 2004, 884. Edited by O. Altan. Istanbul, Turkey: ISPRS. Available online at: http://www.isprs.org/istanbul2004/comm5/papers/674.pdf.

Ostromoukhov, V. “Mathematical Tools for Computer-Generated Ornamental Patterns.” In *Proceedings of the 7th International Conference on Electronic Publishing, Held Jointly with the 4th International Conference on Raster Imaging and Digital Typography 1998,193—223.* London: Springer-Verlag, 1998. Available online at: http://portal.acm.org/citation.cfm?id=725890&dl=GUIDE&coll=GUIDE&CFID=42324566&CFTOKEN=30166319#.

Pedagoguery Software Inc. “Tess.”. 2002, Available online at: http://www.peda.com/tess/Welcome.html.

Reilly, P. “Visualizing the Problem: Advancing Graphic Systems in Archeological Analysis.” In *Computing for Archaeologists*, 127-151. Edited by S. Ross, J. Moffett and J. Henderson. Oxford, England: Oxford University Committee for Archaeology, 1991.

Richards, J. D. and N. Ryan. *Data Processing in Archaeology*. Cambridge: Cambridge University Press, 1985.

Russ, J. *The Image Processing Handbook*. Boca Raton, Florida: CRC Press, 1992.

Shapiro, L. G. and G. C. Stockman. *Computer Vision*. Upper Saddle River, New Jersey: Prentice Hall, Inc., 2001.

Shmidek, E. “Application of Earth Photogrammetry for Photographing Monuments of Cultural and Architectural Interest in the People’s Republic of Bulgaria.” In *Photogrammetric Surveys in Monuments and Sites*, 149 -155. Edited by J. Badekas. Amsterdam, New York: North-Holland Publishing Company and American Elsevier Publishing Company, 1974.

Smith, S. W. “The Inner Light Theory of Consciousness.” 373-396. California Technical Publishing, 2001. Available online at: http://www.innerlighttheory.com.

Snyder, W. and H. Qi. *Machine Vision*. Cambridge: Cambridge University Press, 2004.

Srisuk, S., M. Tamsri,R. Fooprateepsiri, P. Sookavatana and K. Sunat. "A New Shape Matching Measure for Nonlinear Distorted Object Recognition." *Digital Image Computing: Techniques and Application. Proceedings of the VIIth conference, 10-12 Dec. 2003*, 339-348. Edited by C. Sun, H. Talbot, S. Qurselin and T. Adriaansen. Sydney: The University of Queensland, 2003. Available online at: http://www.cmis.csiro.au/Hugues.Talbot/dicta2003/cdrom/pdf/0339.pdf.

Stevens, M. R. and J. R. Beveridge. *Integrating Graphics and Vision for Object Recognition*. London: Kluwer Academic Publishers, 2001.

Stirlin, H. *Islam: Early Architecture from Baghdad to Cordoba*. London: Taschen, 1996.

Tennant, R. "Mathematical Connection." Zayed University, 2004. Available online at: http://home.earthlink.net/~mayathelma/sitebuildercontent/sitebuilderfiles/journey.article.tennant.pdf.

Tombre K. and S. Tabbone. "Vectorization in Graphics Recognition: To Thin or Not to Thin." In *International Conference on Pattern Recognition (ICPR'00)-Volume 2. 09 03 - 09, 2000, Barcelona, Spain*, 2000. Available online at: http://csdl.computer.org/comp/proceedings/icpr/2000/0750/02/07502091abs.htm.

Tomita, F. and S. Tsuji. *Computer Analysis of Visual Textures*. London: Kluwer Academic Publishers, 1990.

Toriwaki, J. and K. Mori. "Distance Transformation and Skeletonization of 3D Pictures and their Applications to Medical Images." In *Digital and Image Geometry: Advanced Lectures,* 412 -428. Edited by G. Bertrand, A. Imiya and R. Klette. Berlin: Springer-Verlag, 2001.

Vasaros, Z. and G. Divinyi. "Progressive 3D Modeling of the Theban Tomb 32." In *[Enter the Past]The E-way into the Four Dimensions of Cultural Heritage. Computer Applications and Quantitative Methods in Archaeology. Proceedings of the 31st Conference*, Vienna, Austria, April 3003, 555-557. Edited by M. Wien, R. Erbe and S. Wien. Hampshire, England: Basingstoke Press, 2004.

Viti, S. "Between Reconstruction and Reproduction: the Role of Virtual Models in Archaeological Research." In *[Enter the Past]The E-way into the Four Dimensions of Cultural Heritage. Computer Applications and Quantitative Methods in Archaeology. Proceedings of the 31st Conference*, Vienna, Austria, April 3003, 525-528. Edited by M. Wien, R. Erbe and S. Wien. Hampshire, England: Basingstoke Press, 2004.

Wade, N. *The Art and Science of Visual Illusions*. London: Routledge & Kagan Paul, 1982.

Whiting, D. and S. Nickerson. "Computer Aided Recording Tools Automate the Creation of a Cite Information System." In *The CIPA Papers, 1997*. Available online at: http:// cipa.icomos.org/papers/97s121-c.htm.

Wieting, T. *The Mathematical Theory of Chromatic Plane Ornaments*. New York: Marcel Dekker, Inc., 1982.

Williamson, R.A. "The Opportunities and Challenges of Preservation Technologies." In *Science and Technology in Historic Preservation: Advances in Archaeological and Museum Science*, 3 - 17. Edited by R. Williamson and P. Nickens. New York: Kluwer Academic/Plenum Publishers, 2000.

Wohlin, C., P. Runeson, M. Host, M. Ohlsson, B. Regnell and A. Wasslen. *Experimentation in Software Engineering: An Introduction*. London: Kluwer Academic Publishers, 2000.

Wu, Q. and Y. Yu. "Two-Level Image Segmentation Based on Region and Edge Integration." In *Proceeding of the VIIth Digital Image Computing: Techniques and Application*, 10-12 Dec, 2003. Edited by C. Sun, S. Qurselin and T. Adeiaansen. Sydney: The University of Queensland, 2003. Available online at: http://www.tip.csiro.au/dicta2003/.

Yamada, O., Y. Takase, I. Shimoda, and T. Nakagawa. "Significance of Digital Reconstruction of Historic Buildings Using 3D Laser Scanner. Case Study: Prasat Suor Prat N1 Tower, Angkor, Cambodia." In *Vision Techniques for Digital Architectural and Archaeological Archives. Proceedings of the Workshop XXXIV-6/ W12*, 1 - 3 July 2003, Italy, 342-346. Edited by G. Fangi and E. Malinverni, 2003. Available online at: http://www.commission5.isprs.org/wg4/workshop_ancona/proceedings/83.pdf.

Yanamura, Y and H. Saji. "Automatic Registration of Aerial Image and Digital Map for Detection of Earthquake Damaged Areas." In *Digital Image Computing: Techniques and Application. Proceedings of the VIIth Conference*,10-12 Dec. 2003, 117-126. Edited by C. Sun, H. Talbot, S. Qurselin and T. Adriaansen. Sydney: The University of Queensland, 2003. Available on line at: http://www.cmis.csiro.au/Hugues.Talbot/dicta2003/cdrom/pdf/0117.pdf.

Zemanek, H. "Archaeological Information: An Information Scientist Looks on Archaeology." In *[Enter the Past]The E-way into the Four Dimensions of Cultural Heritage. Computer Applications and Quantitative Methods in Archaeology.*

Proceedings of the 31st Conference, Vienna, Austria, April 3003, 16-25. Edited by M. Wien, R. Erbe and S. Wien. Hampshire, England: Basingstoke Press, 2004.

Zhong, D. and H. Yan. "Pattern Skeletonization Using Run-Length-Wise Processing for Intersection Distortion Problem." *Pattern Recognition Letters* 20, (1999): 833-846. Elsevier Science. Available online at: www.doc.ic.ac.uk/~xh1/Referece/ feature-filterdesign/run%20length%20processing.pdf.

Zou, J. J. "A Fast Skeletonization Method." In *Proceeding of the VIIth Digital Image Computing: Techniques and Application*, 10-12 Dec. Edited by C. Sun, S. Qurselin and T. Adeiaansen. Sydney: The University of Queensland, 2003. Available online at: http://www.tip.csiro.au/dicta2003/.

Zuech, N. *Understanding and Applying Machine Vision*. (2nd ed.). New York: Marcel Dekker, Inc., 2000.

APPENDIX A

PROGRAM'S ALGORITHM FOR RECONSTRUCTING ISLAMIC GEOMETRIC PATTERNS

```
/*
Program Owner: Rima Ahmad Al Ajlouni.
Date: May 2005.
Program for Virtual Reconstruction of Islamic Geometric Patterns.
/u/rajlouni/research/final% prepare_image complex_1_1200_det.ppm out.ppm out.eps
*/
#include <math.h>
#include <stdio.h>                  /* define standard I/O routines */
#include <stdlib.h>                 /* define standard library routines */
#include <GL/glut.h>                /* define GLUT window and device routines */
#include <string.h>

#define RGBWHITE                    1, 1, 1 /* WHITE for screen background */
#define RGBRED                      1, 0, 0
#define RGBGREEN                    0, 1, 0
#define PI                           3.1415926536
#define CIRC_INC                     (2 * PI / 200)
#define  POLYGON                    (2 * PI / 8)
#define RADIUS                      7          /*  icon small radius */
#define WIDTH                       500
#define HEIGHT                      500

#define maximum(x, y, z) ((x) > (y)? ((x) > (z)? (x) : (z)) : ((y) > (z)? (y):(z)))
#define minimum(x, y, z) ((x) < (y)? ((x) < (z)? (x) : (z)) : ((y) < (z)? (y):(z)))

/*****************************************************************************/
/* enumerate the tools*/
static int              triangle;
enum     triangle{UPPER_1,UPPER_2, LOWER_1,LOWER_2, RIGHT_1,RIGHT_2 ,
LEFT_1,LEFT_2 };

/*** define a structure ***/
typedef struct header {
          char magicn[3];
          char coment[2];
          char w[5], h[5];
          char cvalue[4];
          int  width,height,value;
}ELEMENT;
ELEMENT ppm;

/*** define a structure ***/
typedef struct circle {
          float  x, y;
          float rad;
          float color[3];
}CIRCLE;
CIRCLE  array[5];

/*** define a structure ***/
typedef struct center {
```

```
        float  x, y;
}CENTER;
 CENTER centr[20][20];

/*** define a run length structure ***/
typedef struct  stack {
        int row,col,length;
        struct stack *next;
}RUN/*, VERTICAL*/;

RUN *head=NULL,*tail,*head_1=NULL, *tail_1, *head_2=NULL, *tail_2,
*head_3=NULL, *tail_3 ;
/*VERTICAL *head_1=NULL, *tail_1;*/

/*** define a intersection arrays of dtructures ***/
typedef struct  intersection {
        float x,y;
        int L,R;/* R is the right run length,l the left run length *relative to the
conversion point*/
        float Rx1, Ry1, Rx2, Ry2, Rangle;
        float Lx1, Ly1, Lx2, Ly2, Langle;
}INTER;

INTER conv[100], divr[100], conv_1[100], divr_1[100],conv_2[100],
divr_2[100],conv_3[100], divr_3[100];

typedef struct  point {
        float x,y;

}POINT;

POINT inter[100];

typedef struct  Tow_points {
        float x0,y0,x1,y1 ;
}LINE;

LINE line[100];

/***** Global Variables*****************************************************************/
int Max=0, Min=100, MAX=0;
unsigned int pixel;
int rows, cols, count=0, count_1=0, count_2=0,count_3=0 ,inter_count=0;
unsigned int  **pixma, **bitmap;
char n;
float xjc, yic, Cxj, Cyi ;
float Fdis, Bdis, Udis, Ddis, FUdis, BUdis, BDdis, FDdis,
                        F2U1dis, F1U2dis, B2U1dis, B1U2dis,
                        B2D1dis, B1D2dis, F2D1dis, F1D2dis,
                        rad, radd;
float midxdis, midydis, final_dis;
int flag=0, Lflag=0, Mouse=0, sym=0, MX;
float angle;
```

```
float value=0.5;
static int imageflag=0, inimagespace=0, Center=0, save_flag=0, edit_flag=0,
process_flag=0, triangle_flag=0,
            Pick_triangle_flag=0,finish_flag=0;
static int tracking_M=0;
float start_x, start_y;
int  indx_x, indx_y;
int C_indx=0,D_indx=0,C_indx_1=0, D_indx_1=0, C_indx_2=0, D_indx_2=0,
C_indx_3=0, D_indx_3=0, INDEX=0;/****conversion and diversion inexes***/
int line_count=0;
FILE *out;
char *ppmout;

/*****************************value menu*********************************************/
/* Returns true if point (x, y) is in the value menu */

int inValueMenu(int x, int y){

        return (x >= 0 && x <= 200 && y >= -40 && y <= -20);
}
/*****************************image boundaries*********************************************/
/* Returns true if point (x, y) is in the image booundaries  */

int inImageBoundaries(int x, int y){

        return (x >= 0 && x <= ppm.width  && y >= 0 && y <= ppm.height);
}
/*****************************PICK SIDE button*********************************************/
/* Returns true if point (x, y) is in the PROCESS BUTTON  */

int inPickTriangleButton(int x, int y){

        return (x >= 210 && x <= 310  && y >= -43 && y <= -23);
}
/************************** ****Processbutton*********************************************/
/* Returns true if point (x, y) is in the PROCESS BUTTON  */

int inProcessButton(int x, int y){

        return (x >= 210 && x <= 310  && y >= -73 && y <= -53);
}
/*****************************Process button*********************************************/
/* Returns true if point (x, y) is in the FINISH BUTTON  */

int inFinishButton(int x, int y){

        return (x >= 210 && x <= 310  && y >= -103 && y <= -83);
}
/*****************************SAVEbutton*********************************************/
/* Returns true if point (x, y) is in the SAVE BUTTON  */

int inSaveButton(int x, int y){
```

```
        return (x >= 119 && x <= 172  && y >= -103 && y <= -83);
}
/*****************************************************************************/
void Circlef(float X, float Y, float rad)
{
 float theta;

 glBegin(GL_POLYGON);
   for(theta=0.0; theta < 2 * PI; theta += CIRC_INC)
     glVertex2f(X+rad*cos(theta), Y+rad*sin(theta));
 glEnd();
}
/*****************************************************************************/
void polygon(float X, float Y, float rad )
{
 float theta;
glBegin(GL_LINE_LOOP);
   for(theta=0.0; theta < 2 * PI; theta += POLYGON)
     glVertex2f(X+rad*cos(theta), Y+rad*sin(theta));
 glEnd();
}
/*****************************************************************************/
void F_Circlef_rgb(float X, float Y, float rad, float r, float g, float b )
{
 float theta;
 glBegin(GL_POLYGON);
        glColor3f (r,g,b);
   for(theta=0.0; theta < 2 * PI; theta += CIRC_INC)
     glVertex2f(X+rad*cos(theta), Y+rad*sin(theta));
 glEnd();
}

/*****************************************************************************/
/* Draw a outlined circle with center at position (x, y) and radius rad*/
void Circlef_rgb(float x, float y, float rad, float r, float g, float b)
{
 float theta;
 glBegin(GL_LINE_LOOP);
        glColor3f (r,g,b);
   for(theta=0.0; theta < 2 * PI; theta += CIRC_INC)
     glVertex2f(x+rad*cos(theta), y+rad*sin(theta));
 glEnd();
}

/*****************************************************************************/
/*Draw a line between two points*/
void Draw_line(float x0, float y0, float x1, float y1)
{
 glBegin(GL_LINES);
        glColor3f(1,0,0);
                glVertex2f( x0 , y0);
                glVertex2f( x1 , y1);
        glEnd();
```

```
}
/*****************************Draw value  menu control shape*************************/
void DrawPoint(float x, float y){
glBegin(GL_LINES);
                    glColor3f(0,0,0);
                    glLineWidth(1.0);

                    glVertex2f( x , y+13);
                    glVertex2f( x , y-16);
          glEnd();
glPushMatrix();/* save current transformation matrix   */
          glColor3f (0,0,0);              /* dark gray*/
          Circlef (x-2, y-2, RADIUS);           /* draw the shadow*/
          glColor3f (.9,.2,.2);           /* */
          Circlef (x, y, RADIUS);
          glColor3f (.9,.9,.9);           /* dark gray*/
          Circlef (x+2, y+2, RADIUS-4);
 glPopMatrix ();
}
/*********************Draw center & menue lines*************************************/
void DrawC_lines(void)
{
          glBegin(GL_LINES);
                    glColor3f(.3,.3,.3);
                    glLineWidth(1.0);
                    glVertex2f( 0.0  , -28.0);
                    glVertex2f( 200.0, -28.0);

                    glVertex2f( 0.0  , -27.0);
                    glVertex2f( 200.0, -27.0);


                    glColor3f(.6,.6,.6);
                    glVertex2f( 0.0  , -31.0);
                    glVertex2f( 200.0, -31.0);

                    glVertex2f( 0.0  , -30.0);
                    glVertex2f( 200.0, -30.0);

                    glVertex2f( 0.0  , -29.0);
                    glVertex2f( 200.0, -29.0);

          glEnd();
}
/*************************************Draw the menue*********************************/
void Draw_Menue(void){

          glBegin(GL_POLYGON);/*draw black background of the image*/
                    glColor3f(0,0,0);

                    glVertex2f( -20, -20 );
                    glVertex2f( ppm.width+20, -20);
```

```
                glVertex2f( ppm.width+20, -20);
                glVertex2f( ppm.width+20, ppm.height+20);

                glVertex2f( ppm.width+20, ppm.height+20);
                glVertex2f( -20, ppm.height+20);
        glEnd();

/*draw gray background of the menue in the right side*/
        glBegin(GL_POLYGON);
                glColor3f(.8,.8,.8);
                glLineWidth(1.0);
                glVertex2f( 205, -115.0);
                glVertex2f( ppm.width, -115.0);
                glVertex2f( ppm.width, -115.0);
                glVertex2f( ppm.width, -20);
                glVertex2f( ppm.width, -20 );
                glVertex2f( 205, -20 );
        glEnd();

        glBegin(GL_POLYGON);/*draw black button for the PICK SIDE. button*/
                glColor3f(0,0,0);
                glLineWidth(1.0);
                glVertex2f( 210, -43);
                glVertex2f( 310, -43);
                glVertex2f( 310, -23);
                glVertex2f( 210, -23);
        glEnd();

glBegin(GL_POLYGON);/*draw black button for the PROCESS button*/
                glColor3f(0,0,0);
                glLineWidth(1.0);
                glVertex2f( 210, -73);
                glVertex2f( 310, -73);
                glVertex2f( 310, -53);
                glVertex2f( 210, -53);
        glEnd();

glBegin(GL_POLYGON);/*draw black button for the FINISH button*/
                glColor3f(0,0,0);
                glLineWidth(1.0);
                glVertex2f( 210, -103);
                glVertex2f( 310, -103);
                glVertex2f( 310, -83);
                glVertex2f( 210, -83);
        glEnd();

        glBegin(GL_POLYGON);/*draw dark shade of the background of the menu*/
                glColor3f(.4,.4,.4);
                glLineWidth(1.0);
                glVertex2f(-3.0, -118.0);
                glVertex2f( 197, -118.0);
                glVertex2f( 197 ,-118.0 );
                glVertex2f( 197 , -53);
```

```
                glVertex2f( 197 , -53 );
                glVertex2f(-3.0 , -53 );
        glEnd();
        glBegin(GL_POLYGON);/*draw gray background of the menue*/
                glColor3f(.8,.8,.8);
                glLineWidth(1.0);

                glVertex2f( 0.0, -115.0);
                glVertex2f( 200, -115.0);
                glVertex2f( 200, -115.0);
                glVertex2f( 200, -50);
                glVertex2f( 200, -50 );
                glVertex2f( 0.0, -50 );
        glEnd();

        glBegin(GL_POLYGON);/*draw white background of the value number button*/
                glColor3f(1,1,1);
                glLineWidth(1.0);
                glVertex2f( 147, -72.0);
                glVertex2f( 196, -72.0);
                glVertex2f( 196, -72.0);
                glVertex2f( 196, -55);
                glVertex2f( 196, -55 );
                glVertex2f( 147, -55 );
        glEnd();



glBegin(GL_POLYGON);
                glColor3f(.4,.4,.4);/*draw dark shade of the background of the value bar*/
                glLineWidth(1.0);
                glVertex2f( -3.0, -43);
                glVertex2f( 197, -43);
                glVertex2f( 197, -43);
                glVertex2f( 197, -23);
                glVertex2f( 197, -23 );
                glVertex2f( -3.0, -23 );
        glEnd();

        glBegin(GL_POLYGON);/*draw gray background of the value bar*/

                glColor3f(.8,.8,.8);
                glLineWidth(1.0);
                glVertex2f( 0.0, -40);
                glVertex2f( 200, -40);
                glVertex2f( 200, -40);
                glVertex2f( 200, -20);
                glVertex2f( 200, -20 );
                glVertex2f( 0.0, -20 );
        glEnd();
        glBegin(GL_POLYGON);/*draw white button for the SAVE button*/
                glColor3f(0,0,0);
                glLineWidth(1.0);
```

```
                glVertex2f( 119, -103);
                glVertex2f( 172, -103);
                glVertex2f( 172, -83);
                glVertex2f( 119, -83);
        glEnd();
        DrawC_lines();
}
/**********************extend this line in one direction and return x**************************/
float extendx(float x1, float y1, float theta, float dis){
float x2, y2, xdis=0, ydis=0;
float theta1;
/*if(theta <0)theta= 180+theta;*/
theta1= theta*PI/180;
/*printf("theta1 =%f\n",theta1 );*/
xdis=(cos(theta1) * dis);
ydis=(sin(theta1) * dis);
/*printf(" cos(%f)  =%f\n",theta1, cos(theta1) );
printf(" sin(%f)  =%f\n",theta1,sin(theta1) );
printf(" xdis =%d\n",xdis  );
printf(" ydis =%d\n",ydis  );
printf(" x1 =%d\n",x1  );
printf(" y1 =%d\n",y1  );*/
x2=  x1+xdis;
y2=  y1+ydis;
/*printf("  x2 =%d\n", x2  );
printf("  y2 =%d\n", y2  );*/

/*line( x1 , y1 , x2, y2);*/
return(x2);
}
/************************extend this line in one direction and return y**********************/
float extendy(float x1, float y1, float theta, float dis){
float x2, y2, xdis=0, ydis=0;
float theta1;
theta1= theta*PI/180;
xdis=(cos(theta1) * dis);
ydis=(sin(theta1) * dis);
x2=  x1+xdis;
y2=  y1+ydis;
return(y2);
}
/*********************************draw the text*******************************/
void draw_string_bitmap(void *font, const char* string){
        while (*string)
        glutBitmapCharacter(font, *string++);
}

/************************************display text*********************************/
/* to draw the text */
void writeValue(){
        char a[10];
sprintf( a, "%1.3f" ,value );
        glLineWidth(0.3f);
```

```
	glColor3f(0,0,0);   /* set color to white */

	glPushMatrix();
		glRasterPos2f(1.0,-70);
		draw_string_bitmap(GLUT_BITMAP_HELVETICA_18, "Contrast Value=" );
	glPopMatrix();

	glColor3f(1,0,0);
	glPushMatrix();
		glRasterPos2f(149,-70);
		draw_string_bitmap(GLUT_BITMAP_HELVETICA_18,  a );
	glPopMatrix();
	glPushMatrix();
		glRasterPos2f(20,-100);
		draw_string_bitmap(GLUT_BITMAP_HELVETICA_18, "RESET" );
	glPopMatrix();
	glPushMatrix();
		glRasterPos2f(120,-100);
		draw_string_bitmap(GLUT_BITMAP_HELVETICA_18, "SAVE" );
	glPopMatrix();
	glPushMatrix();
		glRasterPos2f(215,-40);
		draw_string_bitmap(GLUT_BITMAP_HELVETICA_18, "PICK SIDE" );
	glPopMatrix();
	glPushMatrix();
		glRasterPos2f(215,-70);
		draw_string_bitmap(GLUT_BITMAP_HELVETICA_18, "PROCESS" );
	glPopMatrix();
	glPushMatrix();
		glRasterPos2f(230,-100);
		draw_string_bitmap(GLUT_BITMAP_HELVETICA_18, "FINISH" );
	glPopMatrix();

}
/**********************draw square based on the center point*********************************/
void square(float x, float y, float rad)
{
	glBegin(GL_LINE_LOOP);
		glColor3f(0,0,0);
		glLineWidth(1.0);
		glVertex2f( x-rad, y-rad);
		glVertex2f( x+rad, y-rad);
		glVertex2f( x+rad, y+rad);
		glVertex2f( x-rad, y+rad);
	glEnd();
}
/**********************************draw star Area*************************************/
void star_area(float x, float y, float rad, float r, float g, float b)
{
	glBegin(GL_LINE_LOOP);
		glColor3f(r,g,b);
		glLineWidth(1.0);
		glVertex2f( x-rad, y-rad);
```

```
                glVertex2f( x+rad, y-rad);
                glVertex2f( x+rad, y+rad);
                glVertex2f( x-rad, y+rad);
        glEnd();
        glBegin(GL_LINES);
                glColor3f(1,0,0);
                glLineWidth(3.0);
                glVertex2f( x-rad, y);
                glVertex2f( x+rad, y);
                glVertex2f( x, y+rad);
                glVertex2f( x, y-rad);
        glEnd();
}
/************************************draw two triangles, repeated unit**********************/
void draw_two_triangles(float x0, float y0, float x1, float y1 ,float rad)
{
        if (y0==y1)/*if horisontal*/
                if(x1>x0)
                        {
                        glBegin(GL_LINE_LOOP);
                          glColor3f(0,0,1);
                          glVertex2f( x0, y0);
                          glVertex2f( x0+(rad/2),y0-(rad/2));
                          glVertex2f( x1, y1);
                          glVertex2f( x0+(rad/2),y0+(rad/2));
                        glEnd();
                        }
                if(x0>x1)
                        {
                        glBegin(GL_LINE_LOOP);
                          glColor3f(0,0,1);
                          glVertex2f( x1, y1);
                          glVertex2f( x1+(rad/2),y1-(rad/2));
                          glVertex2f( x0, y0);
                          glVertex2f( x1+(rad/2),y1+(rad/2));
                        glEnd();
                }
        if (x0==x1)/*if vertical*/
                if(y1>y0)
                        {
                        glBegin(GL_LINE_LOOP);
                        glColor3f(0,0,1);
                        glVertex2f( x1, y1);
                        glVertex2f( x0+(rad/2),y0+(rad/2));
                        glVertex2f( x0, y0);
                        glVertex2f( x0-(rad/2),y0+(rad/2));
                        glEnd();
                        }
                if(y0>y1)
                        {
                        glBegin(GL_LINE_LOOP);
                        glColor3f(0,0,1);
                        glVertex2f( x0, y0);
```

```
                        glVertex2f( x1+(rad/2),y1+(rad/2));
                        glVertex2f( x1, y1);
                        glVertex2f( x1-(rad/2),y1+(rad/2));
                        glEnd();
                        }
}

/************************************highlight rectangle*******************************/

void highlight_rectangle(float x0, float y0, float x1, float y1, float x2,
float y2)
{
        glBegin(GL_LINE_LOOP);
                glColor3f(0,1,0);
                 glVertex2f( x0, y0);
                 glVertex2f( x1, y1);
                 glVertex2f( x2, y2);
        glEnd();
}
/*******************************find the distance between two points**********************/
/* This function takes a center point and a tangent point*/
float findDis(float xc, float yc, float x, float y){
float Dis;
Dis=sqrt (((x-xc) *(x-xc)) + ((y-yc) *(y-yc)) );
eturn Dis;
}
/*********************************find out angle theta************************************/
/* find out what theta is according to two given points */
float Theta(float x1, float y1, float x2, float y2){
        float theta,theta1, z;
        float x, y;
        x=x2-x1;
        y=y2-y1;
        /*printf(" x=%f\n", x );
        printf(" y=%f\n", y );*/

        theta = atan(y/x);
        /*printf(" y/x =%f\n", y/x  );
        printf("theta =%f\n",theta );*/
        theta1=(theta * (180/PI));
        printf("theta1 =%3.2f\n",theta1 );
        return (theta1);
}
/*********************************find out angle degree**********************************/
/* find out what theta is according to two given points */
float degrees(float x1, float y1, float x2, float y2){
        float theta,theta1,degree, z;
        float x, y;
        x=fabs((x2-x1));
        y=fabs((y2-y1));
        /*printf(" x=%f\n", x );
        printf(" y=%f\n", y );*/
        theta = atan(y/x);
```

```
        /*printf(" y/x =%f\n", y/x  );*/
        /*printf("theta =%f\n",theta );*/
        theta1=(theta * (180/PI));
        /*printf("theta1 =%3.2f\n",theta1 );*/
        if(x2>= x1 && y2>=y1)degree=theta1;
  else if(x2< x1 && y2>y1)degree= 180-theta1;
  else if(x2<= x1 && y2<=y1)degree= 180+theta1;
  else if(x2> x1 && y2<y1)degree= 360-theta1;
        /*printf("  degree=%3.2f\n",degree );*/
        return (degree);
}
/**************************Read ppm image file from the file*****************************/

void read_ppm(char *filename)
{
        FILE *input;
        int w, h, i, j, x;
        char c;
        unsigned int red, green, blue, alpha;
        if((input=fopen(filename ,"r"))==NULL) /*open the file if it is there*/
                {
                printf("There is no file to read from.\n");
                return;
                }
        else{
                /************read header information************/
                /*read the magic number*/
                fscanf(input, "%c%c", &ppm.magicn[0],&ppm.magicn[1]);
                if ( ppm.magicn[0]!= 'P' || ppm.magicn[1]!='6')
                        {
                        printf("The magic number of this file is not P6.\n");
                        return;
                        }
                c=fgetc(input);/*read end on line chracter*/
                c=fgetc(input);/*read the first chracter in the next line*/
                /*check if there is a comment and skip it*/
                while(c == '#')
                        {
                        while(c != '\n')
                                {
                                c=fgetc(input);
                                }
                        c=fgetc(input);
                        }
                ungetc(c,input);
                /*read the width and height and the maximum value*/
                fscanf(input, "%s",ppm.w);
                        ppm.width= atoi( ppm.w);
                fscanf(input, "%s", ppm.h );
                        ppm.height= atoi( ppm.h);
                fscanf(input, "%s", ppm.cvalue);
                        ppm.value= atoi( ppm.cvalue);
                        if(ppm.value > 255 || ppm.value <0) return;
```

```
                n=fgetc(input);/*read the last character in the header*/

                /*print out width, height and the maximum value*/
                printf(" magicn= %c%c\n",ppm.magicn[0],ppm.magicn[1]);
                printf(" width= %d \n", ppm.width);
                printf(" height=%d \n",ppm.height);
                printf(" cvalue=%d \n",ppm.value);
                /*=initiate 2D arrayes for red, green, blue, and alpha==*/
                rows = ppm.height;
        cols = ppm.width;
                pixmap = (unsigned int**)malloc (rows*sizeof(int*));
                for(x=0;x<rows; x++)
                        {
                pixmap[x]   =(unsigned int*)malloc(cols*sizeof(int));
                }
/***Reading pixels and saving it into a pixmap and into different 2d  array for channels*****/
                for(i=rows-1 ;i>=0;i--)
                        for(j=0;j<cols;j++)
                        {
                        red= fgetc(input);
                        green= fgetc(input);
                        blue= fgetc(input);
                        alpha= 255;
                        pixel =  alpha << 24 | blue  << 16 | green  << 8 | red ;
                        pixmap[i][j] = pixel;
                        /*printf("  pixmap[%d][%d]=%d\n", i,j,pixmap[i][j] );*/
                        }
        }
        fclose(input);
}
/*****************************************RGB to HSV****************************/
/*
  Input RGB color primary values: r, g, and b on scale 0 - 255
  Output HSV colors: h on scale 0-360, s and v on scale 0-1
*/
void RGBtoHSV(int r, int g, int b, double &h, double &s, double &v){
  double red, green, blue;
  double max, min, delta;
  red = r / 255.0; green = g / 255.0; blue = b / 255.0;  /* r, g, b to 0 – 1  scale */
  max = maximum(red, green, blue);
  min = minimum(red, green, blue);
  v = max;        /* value is maximum of r, g, b */
  if(max == 0){    /* saturation and hue 0 if value is 0 */
    s = 0;
    h = 0;
  }
  else{
    s = (max - min) / max; /* saturation is color purity on scale 0 - 1 */
    delta = max - min;
      if(delta == 0)                /* hue doesn't matter if saturation
is 0 */
     h = 0;
```

```
   else{
     if(red == max)              /* otherwise, determine hue on scale 0 -
360 */
       h = (green - blue) / delta;
     else if(green == max)
       h = 2.0 + (blue - red) / delta;
     else /* (blue == max) */
       h = 4.0 + (red - green) / delta;
     h = h * 60.0;
     if(h < 0)
       h = h + 360.0;
   }
  }
}

/************************************ HSV to RGB *********************************/
/*
  Input RGB HSV colors: h on scale 0-360, s and v on scale 0-1 color
  Output RGB primary values: r, g, and b on scale 0 - 255
*/
void HSVtoRGB( double H, double S, double V,   double  &RR,   double  &GG,
double  &BB){
/*HSV to RGB (Foley and VanDam)*/
double R,G,B;
int i;
double f,p,q,t;
  if (S == 0) R = G = B = V;
  if (H == 360) H = 0;
  H = H / 60;
  i = int(floor(H));
  f = H - i;
  p = V*(1-S);
  q = V*(1-(S*f));
  t = V*(1 - (S * (1-f)));

            if (i == 0){ R = V; G = t; B = p;}
            if (i == 1){ R = q; G = V; B = p;}
            if (i == 2){ R = p; G = V; B = t;}
            if (i == 3){ R = p; G = q; B = V;}
            if (i == 4){ R = t; G = p; B = V;}
            if (i == 5){ R = V; G = p; B = q;}

            RR= (R*255.0);
            GG= (G*255.0);
            BB= (B*255.0);
}
/*****************************Find the center point for this space*****************************/
void findCenter(int jx, int iy){
unsigned char red;
double value;
int I,J;
unsigned int pix;
float xdif, ydif;
```

```
/*********Reading pixels and saving it into a pixmap and into different 2d array for channels********/
                pixel = bitmap[iy][jx];
                red= pixel;
                printf(" red=%d\n", red );
                if (red==0)printf(" you are on a line. Please pick inside a white space\n");
                else    {

                        /*========Forward=======*/
                        J=jx;
                        J++;
                        while(J<cols)
                                {
                                pix = bitmap[iy][J];
                                red= pix;
                                J++;
                                if(red==0.0)break;
                                }

                        Fdis=findDis( jx, iy, J, iy);
                        printf(" Fdis=%4.2f\n", Fdis );

                        /*========Backward=======*/
                        J=jx;
                        while(J>0)
                                {
                                J--;
                                pix = bitmap[iy][J];
                                red= pix;
                                if(red==0.0)break;
                                }


                        Bdis=findDis( jx, iy, J, iy);
                        printf(" Bdis=%4.2f\n", Bdis );

                        /*========Up=======*/
                        I=iy;
                        I++;
                        while(I<rows)
                                {
                                pix = bitmap[I][jx];
                                red= pix;
                                I++;
                                if(red==0.0)break;
                                }


                        Udis=findDis( jx, iy, jx, I);
                        printf(" Udis=%4.2f\n", Udis );

                        /*========Down=======*/
                        I=iy;
```

```
while(I>0)
        {
        I--;
        pix = bitmap[I][jx];
        red= pix;
        if(red==0.0)break;
        }

Ddis=findDis( jx, iy, jx, I);
printf(" Ddis=%4.2f\n", Ddis );

/*========Diagonal Forward & UP=======*/
J=jx;
I=iy;
J++;
I++;

while(J<cols && I<rows)
        {
        pix = bitmap[I][J];
        red= pix;
        J++;
        I++;
        if(red==0.0)break;
        }

FUdis=findDis( jx, iy, J, I);
printf(" FUdis=%4.2f\n", FUdis );

/*========Diagonal Backward & Down =======*/
I=iy;
J=jx;
while(I>0 && J>0)
        {
        I--;
        J--;
        pix = bitmap[I][J];
        red= pix;
        if(red==0.0)break;
        }

BDdis=findDis( jx, iy, J, I);
printf(" BDdis=%4.2f\n", BDdis );

/*========Diagonal Backward & UP =======*/
J=jx;
I=iy;
I++;
while(J>0 && I<rows)
        {
        J--;
        pix = bitmap[I][J];
        red= pix;
```

```
                I++;
                if(red==0.0)break;
                }

        BUdis=findDis( jx, iy, J, I);
        printf(" BUdis=%4.2f\n", BUdis );


        /*========Diagonal Forward & Down =======*/
        J=jx;
        I=iy;
        J++;
        while(J<cols && I>0)
                {
                I--;
                pix = bitmap[I][J];
                red= pix;
                J++;
                if(red==0.0)break;
                }

        FDdis=findDis( jx, iy, J, I);
        printf(" FDdis=%4.2f\n", FDdis );
        /*========Diagonal Forward 2 & UP 1=======*/
        J=jx;
        I=iy;
        J+=2;
        I++;

        while(J<cols && I<rows)
                {
                pix = bitmap[I][J];
                red= pix;
                J+=2;
                I++;
                if(red==0.0)break;
                }

        F2U1dis=findDis( jx, iy, J, I);
        printf("\n F2U1dis=%4.2f\n", F2U1dis );

/*========Diagonal Forward 1 & UP 2=======*/
        J=jx;
        I=iy;
        J++;
        I+=2;
        while(J<cols && I<rows)
                {
                pix = bitmap[I][J];
                red= pix;
                J++;
                I+=2;
                if(red==0.0)break;
```

```
            }

F1U2dis=findDis( jx, iy, J, I);
printf(" F1U2dis=%4.2f\n", F1U2dis );

/*========Diagonal Backward 2  & UP 1 =======*/
J=jx;
I=iy;
I++;
while(J>0 && I<rows)
            {
            J-=2;
            pix = bitmap[I][J];
            red= pix;
            I++;
            if(red==0.0)break;
            }

B2U1dis=findDis( jx, iy, J, I);
printf(" B2U1dis=%4.2f\n", B2U1dis );

/*========Diagonal Backward 1  & UP 2 =======*/
J=jx;
I=iy;
I+=2;
while(J>0 && I<rows)
            {
            J--;
            pix = bitmap[I][J];
            red= pix;
            I+=2;
            if(red==0.0)break;
            }

B1U2dis=findDis( jx, iy, J, I);
printf(" B1U2dis=%4.2f\n", B1U2dis );

/*========Diagonal Backward 2 & Down 1=======*/

I=iy;
J=jx;
while(I>0 && J>0)
            {
            I--;
            J-=2;
            pix = bitmap[I][J];
            red= pix;
            if(red==0.0)break;
            }

B2D1dis=findDis( jx, iy, J, I);
printf(" B2D1dis=%4.2f\n", B2D1dis );
```

```
/*========Diagonal Backward 1 & Down 2=======*/

I=iy;
J=jx;
while(I>0 && J>0)
        {
        I-=2;
        J--;
        pix = bitmap[I][J];
        red= pix;
        if(red==0.0)break;
        }

B1D2dis=findDis( jx, iy, J, I);
printf(" B1D2dis=%4.2f\n", B1D2dis );

/*========Diagonal Forward2 & Down 1=======*/
J=jx;
I=iy;
J+=2;
while(J<cols && I>0)
        {
        I--;
        pix = bitmap[I][J];
        red= pix;
        J+=2;
        if(red==0.0)break;
        }

F2D1dis=findDis( jx, iy, J, I);
printf(" F2D1dis=%4.2f\n", F2D1dis );

/*========Diagonal Forward 1 & Down 2=======*/
J=jx;
I=iy;
J++;
while(J<cols && I>0)
        {
        I-=2;
        pix = bitmap[I][J];
        red= pix;
        J++;
        if(red==0.0)break;
        }

F1D2dis=findDis( jx, iy, J, I);
printf(" F1D2dis=%4.2f\n", F1D2dis );

/****************find vertical center x************************/
                                        if(Fdis>Bdis)
                {
                midxdis=((Fdis+Bdis)/2);
                /*printf("midxdis=%d\n",midxdis);*/
```

```
                                        xdif=(Fdis-midxdis);
                                        Cxj=jx+xdif;
                                        }
                                else if(Fdis<Bdis)
                                        {
                                        midxdis=((Fdis+Bdis)/2);
                                        /*printf("midxdis=%d\n",midxdis);*/
                                        xdif=(Bdis-midxdis);
                                        Cxj=jx-xdif;
                                        }
                                else if(Fdis==Bdis)Cxj=jx;

                        /****************find horizontal center y*********************/
                        /*************************************************************/
                                if(Udis>Ddis)
                                        {
                                        midydis=((Udis+Ddis)/2);
                                        /*printf("midydis=%d\n",midydis);*/
                                        ydif=(Udis-midydis);
                                        Cyi=iy+ydif;
                                        }
                                else if(Udis<Ddis)
                                        {
                                        midydis=((Udis+Ddis)/2);
                                        /*printf("midydis=%d\n",midydis);*/
                                        ydif=(Ddis-midydis);
                                        Cyi=iy-ydif;
                                        }
                                else if(Udis==Ddis)Cyi=iy;

                        printf("Cyi =%4.2f\n",Cyi);
                        printf("Cxj =%4.2f\n",Cxj);

                        }

}
/**********************check if it is a symmetrical shape*********************************/

void check_if_symetrical(int xx, int yy){

float test;
                findCenter(int(Cxj),int (Cyi));
                if((fabs((Fdis-Bdis))<=6)&&
                  (fabs((Fdis-Ddis))<=6)&&
                   (fabs((Fdis-Udis))<=6)&&

                   (fabs((FUdis-BUdis))<=6)&&
                   (fabs((FUdis-FDdis))<=6)&&
                   (fabs((FUdis-BDdis))<=6)&&

                   (fabs((F2U1dis-B2D1dis))<=6)&&
                   (fabs((F1U2dis-B1D2dis))<=6)&&
                   (fabs((B2U1dis-F2D1dis))<=6)&&
```

```
                (fabs((B1U2dis-F1D2dis))<=6))
                        {
                        test=Fdis;
                        if (Center==0)
                                {
                                array[0].x=Cxj;
                                array[0].y=Cyi;
                                array[0].rad=test;
                                array[0].color[0]=1;
                                array[0].color[1]=0;
                                array[0].color[2]=0;
                                save_flag=1;
                                }

                        /*DrawC_lines(array[0].x ,
                        array[0].y,array[0].color[0],array[0].color[1],array[0].color[2] );*/

                        if(Center==1)
                                {
                                array[1].x=Cxj;
                                array[1].y=Cyi;
                                array[1].rad=test;
                                array[1].color[0]=0;
                                array[1].color[1]=1;
                                array[1].color[2]=0;
                                }
                        }
}

/*********************************Locate other centers*************************/
void find_other_centers(float x,float y,float rad){
int  indx;
float spacing;
        spacing= (2 * rad);         /*==spacing between tow stars, measurements taken
                                    between centers ==*/
        indx=0;
        while (((y-(indx *spacing)))>=-rad/2)
                {
                start_y=((y-(indx *spacing)));
                indx++;
                printf("start_y =%f\n",start_y );
                }

        indx=0;
        while (((x-(indx *spacing)))>=-rad/2)
                {
                start_x=((x-(indx *spacing)));
                indx++;
                printf("start_x =%f\n",start_x );
                }
        for (indx_y=0; (start_y + (indx_y*spacing))<=ppm.height+rad/2; indx_y++)
                for (indx_x=0; (start_x + (indx_x*spacing))<=ppm.width+rad/2; indx_x++)
                        {
```

```
                        centr[indx_y][indx_x].y=(start_y + (float(indx_y)*spacing));
                        centr[indx_y][indx_x].x=(start_x + (float(indx_x)*spacing));
                        printf(" centr[%d ][%d ].x =%4.2f\n",indx_y,indx_x,centr[indx_y][indx_x].x );
                        printf(" centr[%d ][%d ].y =%4.2f\n",indx_y,indx_x,centr[indx_y][indx_x].y );
                        }
        }


/**********SCAN TRIANGLE NOUNDARIES FOR POINTS*******************************/
/*********AND SAVE IT INTO AN INTERSECTION ARRAY*******************************/

void scan_line(float x0, float y0,float x1,float y1)
{
int i, j, pt, x=0;
float ii, jj;
unsigned char red,current,previous;
float angle;
int pix_count=0, start_x, start_y, start=0;
float DX, DY, dx, dy;
        pixel=bitmap[int(y1)][int(x1)];
        red = pixel;
        if(red==0)
                {
                if (inter_count!=0)
                        {
                        for(pt=0; pt<inter_count;pt++)
                                {

                                if(inter[pt].y==y1 && inter[pt].x==x1)
                                        {
                                        x++;
                                        }
                                }
                        if(x==0){
                                printf(" saved\n" );
                                inter[inter_count].y =y1;
                                inter[inter_count].x =x1;
                                inter_count++;
                                }
                        }
                else if(inter_count==0)/**if first time***/
                        {
                        inter[inter_count].y =y1;
                        inter[inter_count].x =x1;
                        inter_count++;
                        }
                }

        angle=degrees( x0, y0, x1, y1);
        printf("angle=%f\n",angle  );
        if(angle>=0 && angle<45)
                {
                previous=255;
```

```
        for(j=int (x0);j<=x1;j++)
                {
                DX= x1-x0;
                DY= y1-y0;
                /*printf("DX =%4.2f,DY=%4.2f \n",DX, DY );*/
                dx= j-x0;
                dy=(DY *dx)/DX;
                i=int (dy+y0);
                /*printf("i =%d \n",i );*/
                pixel=bitmap[i][j];
                red = pixel;
                current=red;
                if( previous==255&& current == 0 && (j!=int (x0) || i!=int (y0)))
                        {
                        start_y=i;
                        start_x=j;
                        pix_count=1;
                        }

                /*black-> black*/
                else if( previous==0&& current == 0 &&( j!=int(x1) || i!=int (y1)))
                        {
                         pix_count++;
                        }
                /*black-> white*/
                else if( previous==0&& current == 255 )
                        {
                        inter[inter_count].x =start_x+(pix_count/2);
                        dx=inter[inter_count].x-x0;
                        dy=(DY *dx)/DX;
                        ii=dy+y0;
                        inter[inter_count].y = ii;

                        inter_count++;
                        }
                previous=current;
                }
        }
else if(angle>=45 && angle<90)
        {
        previous=255;
        for(i=int (y0);i<=y1;i++)
                {
                DX= x1-x0;
                DY= y1-y0;
                /*printf("DX =%4.2f,DY=%4.2f \n",DX, DY );*/
                dy= i-y0;
                dx=(DX *dy)/DY;
                j=int (x0+dx);
                /*printf("i =%d \n",i );*/

                pixel=bitmap[i][j];
                red = pixel;
```

```
                current=red;
                if( previous==255&& current == 0 && (j!=int (x0) || i!=int (y0)))
                        {
                        start_y=i;
                        start_x=j;
                        pix_count=1;
                        }

                /*black-> black*/
                else if( previous==0&& current == 0 &&( j!=int(x1) || i!=int (y1)))
                        {
                         pix_count++;
                        }
                /*black-> white*/
                else if( previous==0&& current == 255 )
                        {
                        inter[inter_count].y =start_y+(pix_count/2);
                        dy=inter[inter_count].y-y0;
                        dx=(DX *dy)/DY;
                        jj=dx+x0;
                        inter[inter_count].x = jj;
                        inter_count++;
                        }
                else if( previous==0 && current == 0 && j==int(x1) && i==int(y1))
                        {
                        inter[inter_count].y =y1;
                        inter[inter_count].x =x1;
                        inter_count++;
                        }

                previous=current;
                }
        }
else if(angle>=90 && angle<135)
        {
        previous=255;
        for(i=int (y0);i<=y1;i++)
                {
                DX=x0-x1;
                DY=y1-y0;
                /*printf("DX =%4.2f,DY=%4.2f \n",DX, DY );*/
                dy=i-y0;
                dx=(DX *dy)/DY;
                j=int(x0-dx);
                /*printf("i =%d \n",i );
                printf("j =%d \n",j );
                printf(" y1 =%d \n",int(y1) );
                printf(" x1 =%d \n",int(x1) );*/
                pixel=bitmap[i][j];
                red = pixel;
                current=red;
                if( previous==255&& current == 0 && (j!=int (x0) || i!=int (y0)))
                        {
```

```
                        start_y=i;
                        start_x=j;
                        pix_count=1;
                        }
                /*black-> black*/
                else if( previous==0&& current == 0 &&( j!=int(x1) || i!=int (y1)))
                        {
                         pix_count++;
                        }
                /*black-> white*/
                else if( previous==0&& current == 255 )
                        {
                        inter[inter_count].y =start_y+(pix_count/2);
                        dy=inter[inter_count].y-y0;
                        dx=(DX *dy)/DY;
                        jj=x0-dx;
                        inter[inter_count].x = jj;
                        inter_count++;
                        }
                else if( previous==0 && current == 0 && (j-int(x1))<=1 && (i-int(y1))<=1 )
                        {
                        inter[inter_count].y =y1;
                        inter[inter_count].x =x1;
                        inter_count++;
                        }

                previous=current;
                }
        }
else if(angle>=135 && angle<180)
        {
        previous=255;
        for(j=int(x0);j>=x1;j--)
                {
                DX=x0-x1;
                DY=y1-y0;
                /*printf("DX =%4.2f,DY=%4.2f \n",DX, DY );*/
                dx=x0-j;
                dy=(DY *dx)/DX;
                i=int(y0+dy);
                /*printf("i =%d \n",i );
                printf("j =%d \n",j );
                printf(" y1 =%d \n",int(y1) );
                printf(" x1 =%d \n",int(x1) );*/
                pixel=bitmap[i][j];
                red = pixel;
                current=red;
                if( previous==255&& current == 0 && (j!=int (x0) || i!=int (y0)))
                        {
                        start_y=i;
                        start_x=j;
                        pix_count=1;
                        }
```

```
                    /*black-> black*/
                    else if( previous==0&& current == 0 &&( j!=int(x1) || i!=int (y1)))
                            {
                             pix_count++;
                            }
                    /*black-> white*/
                    else if( previous==0&& current == 255 )
                            {
                            inter[inter_count].x =start_x-(pix_count/2);
                            dx=x0-inter[inter_count].x;
                            dy=(DY *dx)/DX;
                            ii=dy+y0;
                            inter[inter_count].y =ii;
                            inter_count++;
                            }
                    else if( previous==0 && current == 0 && abs(j-int(x1))<=1
                            && abs(i-int(y1))<=1 )
                            {
                            inter[inter_count].y =y1;
                            inter[inter_count].x =x1;
                            inter_count++;
                            }
                    previous=current;
                    }
            }
else if(angle>=180 && angle<225)
            {
            previous=255;
            for(j=int(x0);j>=x1;j--)
                    {
                    DX=x0-x1;
                    DY=y0-y1;
                    /*printf("DX =%4.2f,DY=%4.2f \n",DX, DY );*/
                    dx=x0-j;
                    dy=(DY *dx)/DX;
                    i=int(y0-dy);
                    /*printf("i =%d \n",i );
                    printf("j =%d \n",j );
                    printf(" y1 =%d \n",int(y1) );
                    printf(" x1 =%d \n",int(x1) );*/
                    pixel=bitmap[i][j];
                    red = pixel;
                    current=red;
                    if( previous==255&& current == 0 && (j!=int (x0) || i!=int (y0)))
                            {
                            start_y=i;
                            start_x=j;
                            pix_count=1;
                            }

                    /*black-> black*/
                    else if( previous==0&& current == 0 &&( j!=int(x1) || i!=int (y1)))
```

```
                    {
                     pix_count++;
                    }
                /*black-> white*/
                else if( previous==0&& current == 255 )
                    {
                    inter[inter_count].x =start_x-(pix_count/2);
                    dx=x0-inter[inter_count].x;
                    dy=(DY *dx)/DX;
                    ii=y0-dy;
                    inter[inter_count].y =ii;
                    inter_count++;
                    }
                else if( previous==0 && current == 0 )
                    {
                    pixel=bitmap[int(y1)][int(x1)];
                    red = pixel;
                    if(red==0)
                        {
                        inter[inter_count].y =y1;
                        inter[inter_count].x =x1;
                        inter_count++;
                        }
                    }

                previous=current;
                }
        }
else if(angle>=225 && angle<270)
        {
        previous=255;
        for(i=int(y0);i>=y1;i--)
                {
                DX=x0-x1;
                DY=y0-y1;
                /*printf("DX =%4.2f,DY=%4.2f \n",DX, DY );*/
                dy=y0-i;
                dx=(DX *dy)/DY;
                j=int(x0-dx);
                /*printf("i =%d \n",i );
                printf("j =%d \n",j );
                printf(" y1 =%d \n",int(y1) );
                printf(" x1 =%d \n",int(x1) );*/
                pixel=bitmap[i][j];
                red = pixel;
                current=red;
                if( previous==255&& current == 0 && (j!=int (x0) || i!=int (y0)))
                    {
                    start_y=i;
                    start_x=j;
                    pix_count=1;
                    }
                /*black-> black*/
```

```
                else if( previous==0&& current == 0 &&( j!=int(x1) || i!=int (y1)))
                        {
                         pix_count++;
                        }
                /*black-> white*/
                else if( previous==0&& current == 255 )
                        {
                        inter[inter_count].y =start_y-(pix_count/2);
                        dy=y0-inter[inter_count].y;
                        dx=(DX *dy)/DY;
                        jj=x0-dx;
                        inter[inter_count].x =jj;
                        inter_count++;
                        }
                previous=current;
                }
        }
else if(angle>=270 && angle<315)
        {
        previous=255;
        for(i=int(y0);i>=y1;i--)
                {
                DX=x1-x0;
                DY=y0-y1;
                /*printf("DX =%4.2f,DY=%4.2f \n",DX, DY );*/
                dy=y0-i;
                dx=(DX *dy)/DY;
                j=int(x0+dx);
                /*printf("i =%d \n",i );
                printf("j =%d \n",j );
                printf(" y1 =%d \n",int(y1) );
                printf(" x1 =%d \n",int(x1) );*/
                pixel=bitmap[i][j];
                red = pixel;
                current=red;
                if( previous==255&& current == 0 && (j!=int (x0) || i!=int (y0)))
                        {
                        start_y=i;
                        start_x=j;
                        pix_count=1;
                        }

                /*black-> black*/
                else if( previous==0&& current == 0 &&( j!=int(x1) || i!=int (y1)))
                        {
                         pix_count++;

                        }
                /*black-> white*/
                else if( previous==0&& current == 255 )
                        {
                        inter[inter_count].y =start_y-(pix_count/2);
                        dy=y0-inter[inter_count].y;
```

```
                                        dx=(DX *dy)/DY;
                                        jj=x0+dx;
                                        inter[inter_count].x =jj;
                                        inter_count++;
                                        }


                                previous=current;
                                }
                        }
                else if(angle>=315 && angle<360)
                        {
                        previous=255;
                        for(j=int (x0);j<=x1;j++)
                                {
                                DX= x1-x0;
                                DY= y0-y1;
                                /*printf("DX =%4.2f,DY=%4.2f \n",DX, DY );*/

                                dx= j-x0;
                                dy=(DY *dx)/DX;
                                i=int (y0-dy);
                                /*printf("i =%d \n",i );*/
                                pixel=bitmap[i][j];
                                red = pixel;
                                current=red;
                                if( previous==255&& current == 0 && (j!=int (x0) || i!=int (y0)))
                                        {
                                        start_y=i;
                                        start_x=j;
                                        pix_count=1;
                                        }
                                /*black-> black*/
                                else if( previous==0&& current == 0 &&( j!=int(x1) || i!=int (y1)))
                                        {
                                         pix_count++;
                                        }
                                /*black-> white*/
                                else if( previous==0&& current == 255 )
                                        {
                                        inter[inter_count].x =start_x+(pix_count/2);
                                        dx=inter[inter_count].x-x0;
                                        dy=(DY *dx)/DX;
                                        ii=y0-dy;
                                        inter[inter_count].y = ii;
                                        inter_count++;
                                        }
                                previous=current;
                                }

                        }

}
```

```
/************************************find the other point of the line*********************/
/************************************sing a starting point and an angle*******************/
void findPOint(float x0, float y0, float x2, float y2, float angle){
float DX, DY, Dist, theta;
        Dist=findDis( x0,y0 , x2, y2);


                if (angle >=0 && angle<=45)
                        {
                        theta= (angle *PI/180);
                        DX=Dist;
                        DY=(tan(theta))* DX;
                        array[4].x=x0+Dist;
                        array[4].y=DY+y0;
                        }
                else if (angle >45 && angle<=90)
                        {
                        theta= (angle *PI/180);
                        DY=Dist;
                        DX=DY/(tan(theta)) ;
                        array[4].x=x0+DX;
                        array[4].y=y0+Dist;
                        }
                 else if (angle >90 && angle<=135)
                        {
                        angle= 180-angle;
                        theta= (angle *PI/180);
                        DY=Dist;
                        DX=DY/(tan(theta)) ;
                        array[4].x=x0-DX;
                        array[4].y=y0+Dist;
                        }
                else if (angle >135 && angle<=180)
                        {
                        angle= 180-angle;
                        theta= (angle *PI/180);
                        DX=Dist;
                        DY=(tan(theta))* DX;
                        array[4].x=x0-Dist;
                        array[4].y=DY+y0;
                        }
                else if (angle >180 && angle<=225)
                        {
                        angle= angle-180 ;
                        theta= (angle *PI/180);
                        DX=Dist;
                        DY=(tan(theta))* DX;
                        array[4].x=x0-Dist;
                        array[4].y=y0-DY;
                        }
                else if (angle >225 && angle<=270)
                        {
```

```
                    angle= angle-180 ;
                    theta= (angle *PI/180);
                    DY=Dist;
                    DX=DY/(tan(theta)) ;
                    array[4].x=x0-DX;
                    array[4].y=y0-Dist;
                    }
                else if (angle >270 && angle<315)
                    {
                    angle= 360-angle ;
                    theta= (angle *PI/180);
                    DY=Dist;
                    DX=DY/(tan(theta)) ;
                    array[4].x=x0+DX;
                    array[4].y=y0-Dist;
                    }
                else if (angle >315 && angle<360)
                    {
                    angle= 360-angle ;
                    theta= (angle *PI/180);
                    DX=Dist;
                    DY=(tan(theta))* DX;
                    array[4].x=x0+Dist;
                    array[4].y=y0-DY;
                    }

}
/*****************************check which triangle*********************************/
/*****************************************************************************/
int inTriangle (float x, float y){
float Angle;
angle=degrees(array[0].x,array[0].y,x,y);
if(triangle==UPPER_1)return (x > array[0].x+5 && x <array[4].x-5 && angle >0.0+5 &&  angle < 22.5-
5&& y > array[0].y+5 );
if(triangle==UPPER_2)return (x > array[4].x+5 && x <array[0].x-5 && angle >157.5+5 &&  angle <
180-5&& y > array[0].y+5 );
if(triangle==LOWER_1)return (x > array[0].x+5 && x <array[4].x-5 && angle >337.5+5 &&  angle <
360-5&& y < array[0].y-5);
if(triangle==LOWER_2)return (x > array[4].x+5 && x <array[0].x-5 && angle >180+5 &&  angle <
202.5-5 && y < array[0].y-5);
if(triangle==RIGHT_1)return (y > array[0].y+5 && y <array[4].y-5 && angle >67.5+5 &&  angle < 90-5
&& x > array[0].x+3);
if(triangle==RIGHT_2)return (y > array[4].y+5 && y <array[0].y-5 && angle >270+5 &&  angle <292.5-
5&& x > array[0].x+5);
if(triangle==LEFT_1) return (y > array[0].y+5 && y <array[4].y-5 && angle >90+5 &&  angle < 112.5-5
&& x < array[0].x-5);
if(triangle==LEFT_2) return (y > array[4].y+5 && y <array[0].y-5 && angle >247.5+5 &&  angle <270-5
&& x < array[0].x-5 );
}
/*****************************************final point*********************************/
void Final_point(float x0, float y0, float X_R,float Y_R ,float X_L ,float Y_L)
{
int skip_point=0, White=0 ;
```

```
float theta,theta1,theta2, degree, distance,distance1,distance2,
final_x,final_y, final_xx,final_yy;
float x, y,xx, yy, r ;
unsigned char red,current;

                theta=degrees(X_R, Y_R ,X_L , Y_L);
                distance=(findDis(X_R , Y_R ,X_L , Y_L ));
                final_x= extendx(X_R , Y_R, theta, distance/2 );
                final_y= extendy(X_R , Y_R, theta, distance/2 );
                F_Circlef_rgb( final_x ,final_y, 1, 0,1,0);
                degree=degrees(x0,y0, final_x, final_y );
                theta1=degree * PI/180;
                distance1=(findDis( x0,y0, final_x, final_y ));
                for ( r=0; r<=(4*distance1);r++)
                        {
                        x= final_x+ r *cos(theta1);
                        y= final_y+ r *sin(theta1);
                        pixel=bitmap[int(y)][int(x)];
                        red=pixel;
                        current=red;
                        if(r==int(4*distance1) && White==0)
                                {
                                current=255;
                                printf(" hi\n");
                                }
                        if(current==255 && White==0)
                                {
                                White=1;
                                xx=x;
                                yy=y;
                                r=(4*distance1+2);
                                }
                        }
                if(White==1)
                        {
                        distance2=(findDis( x0,y0,xx ,yy ));
                        Draw_line( x0,y0, xx, yy);
                        theta2=degrees( x0,y0, xx, yy);
                        final_xx= extendx(x0, y0, theta2, distance2/2 );
                        final_yy= extendy(x0, y0, theta2, distance2/2 );
                        F_Circlef_rgb( final_xx ,final_yy, 2, 1,0,1);
                        }
                if (inTriangle(final_xx ,final_yy ))
                        {
                        /*printf("final_x= %4.2f\n",final_x );
                        printf("final_y= %4.2f\n",final_y );*/
                        for (int pt=0; pt<inter_count;pt++)
                                {
                                /*printf("inter[%d].x= %4.2f\n", pt,inter[pt].x );
                                printf("inter[%d].y= %4.2f\n", pt,inter[pt].y );
                                printf("fabs(inter[%d].x-final_x)= %4.2f\n", pt,fabs(inter[pt].x –
                                final_x));
                                printf("fabs(inter[%d].y-final_y)= %4.2f\n", pt,fabs(inter[pt].y
```

```
                                        final_y));*/

                                        if( (fabs(inter[pt].x- final_xx))<=5 &&
                                           (fabs(inter[pt].y- final_yy))<=5)
                                                {
                                                skip_point=1;
                                                }

                                        }
                                if (skip_point==0)
                                        {
                                        inter[inter_count].x = final_xx;
                                        inter[inter_count].y = final_yy;
                                        F_Circlef_rgb( inter[inter_count].x , inter[inter_count].y,2, 1,1,1);
                                        inter_count++;
                                        }
                                }

}
/*********************************find final intersection*************************************/
/*************************************Vertical*************************************************/
void Final_intersection_C(float x0,float y0, float Dis)
{
int  flag_R=0,flag_L=0;
unsigned char red;
float theta,distance, x, y, D, X_R, Y_R, X_L, Y_L, r,theta1, final_x, final_y ;
        for( r=4; r<=3*Dis; r+=0.3)
                {
                for(theta=PI*35/18; theta >=(PI*30/18) ; theta -= CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_R==0)
                                {
                                X_R=x;
                                Y_R=y;
                                flag_R=1;
                                }
                        }
                for(theta=PI*19/18; theta <=(PI*25/18) ; theta += CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_L==0)
                                {
                                X_L=x;
                                Y_L=y;
```

```
                                        flag_L=1;
                                        }
                                }
                        }
                        if(flag_R==1&& flag_L==1)
                                {
                                Final_point(x0, y0, X_R,Y_R ,X_L ,Y_L);
                                F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
                                F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);
                                Draw_line( X_R, Y_R ,X_L , Y_L);
                                }
}

/*******************************find final intersection*************************************/
/*********************************Vertical*********************************************/
void Final_intersection_D(float x0,float y0, float Dis)
{
int  flag_R=0,flag_L=0;
unsigned char red;
float theta,distance, x, y, D, X_R, Y_R, X_L, Y_L, r,theta1, final_x, final_y ;
        for( r=4; r<=3*Dis; r+=0.3)
                {
                for(theta=PI*1/18; theta <=(PI*6/18) ; theta += CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_R==0)
                                {
                                X_R=x;
                                Y_R=y;
                                flag_R=1;
                                }
                        }
                for(theta=PI*17/18; theta >=(PI*12/18) ; theta -= CIRC_INC)
                        {
                        x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_L==0)
                                {
                                X_L=x;
                                Y_L=y;
                                flag_L=1;
                                }
                        }
                }
                if(flag_R==1&& flag_L==1)
                        {
```

```
                        Final_point(x0, y0, X_R,Y_R ,X_L , Y_L);
                        F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
                        F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);
                        Draw_line( X_R, Y_R ,X_L , Y_L);
                        }

}
/**********************************find final intersection*****************************/
/**********************************horisontal*************************************/

void Final_intersection_C_1(float x0,float y0, float Dis)
{
int  flag_R=0,flag_L=0;
unsigned char red;
float theta,distance, x, y, D, X_R, Y_R, X_L, Y_L, r,theta1, final_x, final_y ;
        for( r=4; r<=3*Dis; r+=0.3)
                {
                for(theta=PI*8/18; theta >=(PI*3/18) ; theta -= CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_R==0)
                                {
                                X_R=x;
                                Y_R=y;
                                flag_R=1;
                                }
                        }
                for(theta=PI*28/18; theta <=(PI*33/18) ; theta += CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_L==0)
                                {
                                X_L=x;
                                Y_L=y;
                                flag_L=1;
                                }
                        }
                }
                if(flag_R==1&& flag_L==1)
                        {
                        Final_point(x0, y0, X_R,Y_R ,X_L , Y_L);

                        F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
                        F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);
                        Draw_line( X_R, Y_R ,X_L , Y_L);
```

```
                        }
}

/**************************find final intersection*************************************/
/*****************************Horisontal*********************************************/
void Final_intersection_D_1(float x0,float y0, float Dis)
{
int  flag_R=0,flag_L=0;
unsigned char red;
float theta,distance, x, y, D, X_R, Y_R, X_L, Y_L, r,theta1, final_x, final_y ;
        for( r=4; r<=3*Dis; r+=0.3)
                {
                for(theta=PI*10/18; theta <=(PI*15/18) ; theta += CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_R==0)
                                {
                                X_R=x;
                                Y_R=y;
                                flag_R=1;
                                }
                        }

                for(theta=PI*26/18; theta >=(PI*21/18) ; theta -= CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_L==0)
                                {
                                X_L=x;
                                Y_L=y;
                                flag_L=1;
                                }
                        }

                }
                if(flag_R==1&& flag_L==1)
                        {
                        Final_point(x0, y0, X_R,Y_R ,X_L , Y_L);
                        F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
                        F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);
                        Draw_line( X_R, Y_R ,X_L , Y_L);
                        }

}
```

```
/************************find final intersection*********************************/
/*****************************diagonal*****************************************/
void Final_intersection_C_2(float x0,float y0, float Dis)
{
int  flag_R=0,flag_L=0;
unsigned char red;
float theta,distance, x, y, D, X_R, Y_R, X_L, Y_L, r,theta1, final_x, final_y ;
        for( r=4; r<=6*Dis; r+=0.3)
                {
                for(theta=PI*3.5/18; theta >=0.0-PI*2/18; theta -= CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_R==0)
                                {
                                X_R=x;
                                Y_R=y;
                                flag_R=1;
                                }
                        }
                for(theta=PI*23.5/18; theta <=(PI*29/18) ; theta += CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_L==0)
                                {
                                X_L=x;
                                Y_L=y;
                                flag_L=1;
                                }
                        }
                }
                /*F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
                F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);*/

                if(flag_R==1&& flag_L==1)
                        {
                        Final_point(x0, y0, X_R,Y_R ,X_L , Y_L);
                        F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
                        F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);
                        Draw_line( X_R, Y_R ,X_L , Y_L);
                        }
}
/************************find final intersection*********************************/
/*****************************diagonal*****************************************/
void Final_intersection_D_2(float x0,float y0, float Dis)
{
```

```
int  flag_R=0,flag_L=0;
unsigned char red;
float theta,distance, x, y, D, X_R, Y_R, X_L, Y_L, r,theta1, final_x, final_y ;
        for( r=4; r<=6*Dis; r+=0.3)
                {
                for(theta=PI*5.5/18; theta <=(PI*11/18) ; theta += CIRC_INC)
                        {


                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_R==0)
                                {
                                X_R=x;
                                Y_R=y;
                                flag_R=1;
                                }
                        }
                for(theta=PI*21.5/18; theta >=(PI*16/18) ; theta -= CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_L==0)
                                {
                                X_L=x;
                                Y_L=y;
                                flag_L=1;
                                }
                        }

                }
                if(flag_R==1&& flag_L==1)
                        {
                        Final_point(x0, y0, X_R,Y_R ,X_L , Y_L);
                        F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
                        F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);
                        Draw_line( X_R, Y_R ,X_L , Y_L);
                        }
}
/************************find final intersection*************************************/
/*****************************diagonal*************************************************/
void Final_intersection_C_3(float x0,float y0, float Dis)
{
int  flag_R=0,flag_L=0;
unsigned char red;
float theta,distance, x, y, D, X_R, Y_R, X_L, Y_L, r,theta1, final_x, final_y ;
        for( r=4; r<=6*Dis; r+=0.3)
```

```
		{
		for(theta=PI*30.5/18; theta >=PI*25/18 ; theta -= CIRC_INC)
			{
			x= x0+ (r*cos(theta));
			y= y0+ (r*sin(theta));
			/*printf(" x=%f, y=%f\n\n", x, y );*/
			pixel = bitmap[int (y)] [int(x)];
			red= pixel;
			if(red==255 &&flag_R==0)
				{
				X_R=x;
				Y_R=y;
				flag_R=1;
				}
			}

		for(theta=PI*14.5/18; theta <=(PI*20/18) ; theta += CIRC_INC)
			{
			x= x0+ (r*cos(theta));
			y= y0+ (r*sin(theta));
			/*printf(" x=%f, y=%f\n\n", x, y );*/
			pixel = bitmap[int (y)] [int(x)];
			red= pixel;
			if(red==255 &&flag_L==0)
				{
				X_L=x;
				Y_L=y;
				flag_L=1;
				}
			}
		}
		/*F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
		F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);*/
		if(flag_R==1&& flag_L==1)
			{
			Final_point(x0, y0, X_R,Y_R ,X_L , Y_L);
			F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
			F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);
			Draw_line( X_R, Y_R ,X_L , Y_L);
			}
}
/************************find final intersection*******************/
/*****************************diagonal*********************************************/
void Final_intersection_D_3(float x0,float y0, float Dis)
{
int  flag_R=0,flag_L=0;
unsigned char red;
float theta,distance, x, y, D, X_R, Y_R, X_L, Y_L, r,theta1, final_x, final_y ;
	for( r=4; r<=6*Dis; r+=0.3)
		{
		for(theta=PI*32.5/18; theta <=(PI*36/18)+PI*2/18 ; theta += CIRC_INC)
			{
			x= x0+ (r*cos(theta));
```

```
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_R==0)
                                {
                                X_R=x;
                                Y_R=y;
                                flag_R=1;
                                }
                        }

                for(theta=PI*12.5/18; theta >=(PI*7/18) ; theta -= CIRC_INC)
                        {
                        x= x0+ (r*cos(theta));
                        y= y0+ (r*sin(theta));
                        /*printf(" x=%f, y=%f\n\n", x, y );*/
                        pixel = bitmap[int (y)] [int(x)];
                        red= pixel;
                        if(red==255 &&flag_L==0)
                                {
                                X_L=x;
                                Y_L=y;
                                flag_L=1;
                                }
                        }
                }

                /*F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
                F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);*/
                if(flag_R==1&& flag_L==1)
                        {
                        Final_point(x0, y0, X_R,Y_R ,X_L , Y_L);
                        F_Circlef_rgb(X_R, Y_R ,2, 1,0,0);
                        F_Circlef_rgb(X_L, Y_L ,2, 1,0,0);
                        Draw_line( X_R, Y_R ,X_L , Y_L);
                        }
}
/*************************calculate the exact intersection point*****************************/
/*****************************************vertival*************************************/
void Conv_intersection (RUN *change, int indx)
{
unsigned char red,current, prev=0;
float  Rx,Lx ;
float angle, theta,X,Y,xx,yy,x1,y1,x2,y2,rad, S_x, S_y, M_x=2, M_y=2, dis;
int k=0, r, l,i;
POINT p_1[100],p_2[100], p[100]  ;
        Rx=change->next->col;
        Lx=(change->col + (change->length));
        for(theta=0.0; theta < PI/2 ; theta += CIRC_INC)
                {
                rad=change->next->length-5;
                xx=(conv[indx].x+ rad*cos(theta));
```

```
		yy=(conv[indx].y+ rad*sin(theta));
		pixel = bitmap[int(yy)][int(xx)];
		red= pixel;
		current=red;
			if(prev==0 && current==255 && k==0)
				{
				x1=xx;
				y1=yy;
				theta=degrees(Rx-1,conv[indx].y  ,x1,y1);
				S_x=(Rx) +(change->next->length/2);
				S_y=conv[indx].y;
				r= Save_extend_points(S_x,S_y,theta, p_1);
				for (i=0; i<r; i++)F_Circlef_rgb(p_1[i].x, p_1[i].y ,1, 0,1,1);
				k=1;
				}
		prev=current;
		}
prev=0;
for( theta=PI; theta >= PI/2 ; theta -= CIRC_INC)
		{
		rad=change->length-5;
		xx=(conv[indx].x+ rad*cos(theta));
		yy=(conv[indx].y+ rad*sin(theta));
		pixel = bitmap[int(yy)][int(xx)];
		red= pixel;
		current=red;
			if(prev==0 && current==255 && k==1)
				{
				x2=xx;
				y2=yy;
				angle=degrees(Lx,conv[indx].y ,x2,y2);
				printf(" angle=%4.1f \n",angle  );
				S_x=(Lx)-(change->length/2);
				S_y=conv[indx].y;
				l= Save_extend_points(S_x,S_y,angle, p_2);
				for (i=0; i<l; i++)F_Circlef_rgb(p_2[i].x, p_2[i].y ,1, 0,1,1);
				k=2;
				}
		prev=current;
		}
if (k==2)
		{
		printf(" r= %d\n", r);
		printf(" l= %d\n", l);
		for (int rr=0; rr<r; rr++)
			for (int ll=0; ll<l; ll++)
				{
				if( fabs(p_1[rr].x - p_2[ll].x)<2 && fabs(p_1[rr].y-p_2[ll].y)<2)
					{
					if (fabs (p_1[rr].x - p_2[ll].x) < M_x) M_x=p_1[rr].x ;
					if (fabs (p_1[rr].y - p_2[ll].y) < M_y) M_y=p_2[ll].y;
					}
				}
```

```
                    if (inTriangle(M_x,M_y))
                            {
                            printf(" found one.....Yeahhh...\n" );
                            F_Circlef_rgb(  M_x , M_y , 2,1,0,0);
                            p[INDEX].x=M_x;
                            p[INDEX].y=M_y;
                            INDEX++;
                            }

                    }
}
/*****************************calculate the exact intersectionpoint*********************/
/****************************** Diversion***vertival*******************************/
/
void Divr_intersection (RUN *change, int indx)
{
unsigned char red,current, prev=0;
float  Rx,Lx ;
float angle,angle1, theta,X,Y,xx,yy,x1,y1,x2,y2,rad, S_x, S_y, M_x=2, M_y=2,
f_dis;
int k=0, r, l,i;
POINT p_1[100],p_2[100], p[100]  ;
        Rx=change->next->col;
        Lx=(change->col + (change->length));
        for(theta=PI*2; theta > PI*3/2 ; theta -= CIRC_INC)
                {
                rad=change->next->length-5;
                xx=(divr[indx].x+ rad*cos(theta));
                yy=(divr[indx].y+ rad*sin(theta));
                pixel = bitmap[int(yy)][int(xx)];
                red= pixel;
                current=red;
                        if(prev==0 && current==255 && k==0)
                                {
                                x1=xx;
                                y1=yy;
                                theta=degrees(Rx-1,divr[indx].y  ,x1,y1);
                                S_x=(Rx) + change->next->length/2;
                                S_y=divr[indx].y;
                                r= Save_extend_points(S_x,S_y,theta, p_1);
                                for (i=0; i<r; i++)F_Circlef_rgb(p_1[i].x, p_1[i].y ,1, 0,1,1);
                                k=1;
                                }
                prev=current;
                }
        prev=0;
        for( theta=PI; theta <= PI*3/2 ; theta += CIRC_INC)
                {
                rad=change->length-5;
                xx=(divr[indx].x+ rad*cos(theta));
                yy=(divr[indx].y+ rad*sin(theta));
                pixel = bitmap[int(yy)][int(xx)];
                red= pixel;
```

```
                current=red;
                        if(prev==0 && current==255 && k==1)
                                {
                                x2=xx;
                                y2=yy;
                                angle=degrees(Lx,divr[indx].y ,x2,y2);
                                printf(" angle=%4.1f \n",angle  );
                                S_x=(Lx)-(change->length/2);
                                S_y=divr[indx].y;
                                l= Save_extend_points(S_x,S_y,angle, p_2);
                                for (i=0; i<l; i++)F_Circlef_rgb(p_2[i].x, p_2[i].y ,1, 0,1,1);
                                k=2;
                                }
                prev=current;
                }
        if (k==2)
                {
                printf(" r= %d\n", r);
                printf(" l= %d\n", l);
                for (int rr=0; rr<r; rr++)
                        for (int ll=0; ll<l; ll++)
                                {

                                if( fabs(p_1[rr].x - p_2[ll].x)<2 && fabs(p_1[rr].y-p_2[ll].y)<2)
                                        {
                                        if (fabs (p_1[rr].x - p_2[ll].x) < M_x) M_x=p_1[rr].x ;
                                        if (fabs (p_1[rr].y - p_2[ll].y) < M_y) M_y=p_2[ll].y;
                                        }
                                }
                if (inTriangle(M_x,M_y))
                        {
                        printf(" found one.....Yeahhh...\n" );
                        F_Circlef_rgb(  M_x , M_y , 2,1,0,0);
                        p[INDEX].x=M_x;
                        p[INDEX].y=M_y;
                        INDEX++;
                        }
                }
}
/*****************************calculate the exact conversion point*************************/
/*****************************************horisontal*************************************/
void conversion(RUN *change, int indx)
{
float end_col, conversion_x, dis;
unsigned char red, red_1,current, prev=0,;
int y,Rx, Lx,rad,xx,yy ;
        end_col=(change->col + (change->length-1));
        conversion_x=(end_col + (change->next->col-end_col)/2);
        pixel = bitmap[change->row+1][change->col+((change->length)/2)];
        red= pixel;
        pixel = bitmap[change->next->row+1][change->next->col+(change->length/2)];
        red_1= pixel;
        if (change->next->col-end_col <=10 && inTriangle(conversion_x,change->row)
```

```
		&& red==0 && red_1==0*/)
			{
			conv[indx].x= conversion_x;
			conv[indx].y= change->row;
			F_Circlef_rgb(	conv[indx].x, conv[indx].y ,3, 0,.5,.5);
			if (change->next->length <change->length) dis=change->next->length/2;
			else dis=change->length/2;
			Final_intersection_C(conv[indx].x,conv[indx].y, dis );
			C_indx++;
			}
}
/*****************************calculate the exact conversion point*************************/
/*****************************************vertival*************************************/
void conversion_1(RUN *change, int indx)
{
float end_row, conversion_y, dis;
		end_row=(change->row + (change->length-1));
		conversion_y=(end_row + (change->next->row-end_row)/2);
		if (change->next->row-end_row <=10 && inTriangle(change->col,conversion_y))
			{
			conv_1[indx].x= change->col;
			conv_1[indx].y= conversion_y;

			F_Circlef_rgb(	conv_1[indx].x, conv_1[indx].y ,3, 0,.5,.5);
			if (change->next->length <change->length) dis=change->next->length/2;
			else dis=change->length/2;
			Final_intersection_C_1(conv_1[indx].x,conv_1[indx].y, dis );
			C_indx_1++;
			}
}
/*****************************calculate the exact conversion point*************************/
/*********************************Diagonal (down to up)*********************************/

void conversion_2(RUN *change, int indx)
{
float end_row, end_col, conversion_y, conversion_x, dis;
int end_y, end_x;
unsigned char red, red_1;
		end_row=(change->row + (change->length-1));
		end_col=(change->col + (change->length-1));
		end_y=(change->next->row + (change->next->length/2));
		end_x=(change->next->col + (change->next->length/2));
		pixel = bitmap[change->row+ (change->length/2)+1] [change->col+
		(change->length/2)-1];
		red= pixel;
		pixel = bitmap[end_y+1][end_x-1];
		red_1= pixel;
		conversion_y=(end_row + (change->next->row-end_row)/2);
		conversion_x=(end_col + (change->next->col-end_col)/2);
		if (change->next->row-end_row <=7 && change->next->col-end_col<=7
			&& red==0 && red_1==0 && inTriangle(conversion_x ,conversion_y))
			{
			conv_2[indx].x= conversion_x;
```

```
                conv_2[indx].y= conversion_y;
                F_Circlef_rgb(     conv_2[indx].x, conv_2[indx].y ,3, 0,0,1);
                if (change->next->length <change->length) dis=change->next->length/2;
                else dis=change->length/2;
                Final_intersection_C_2(conv_2[indx].x,conv_2[indx].y, dis );
                C_indx_2++;
                }
}
/*****************************calculate the exact conversion point************************/
/*****************************************Diagonal (up yo down)************************/
void conversion_3(RUN *change, int indx)
{
float end_row, end_col, conversion_y, conversion_x, dis;
int mid_y, mid_x;
unsigned char red, red_1;
        end_row=(change->row - (change->length-1));
        end_col=(change->col + (change->length-1));

        mid_y=(change->next->row - (change->next->length/2));
        mid_x=(change->next->col + (change->next->length/2));
        pixel = bitmap[change->row -(change->length/2)+1] [change->col+
        (change->length/2)+1];
        red= pixel;
        pixel = bitmap[mid_y+1][mid_x+1];
        red_1= pixel;
        conversion_y=(end_row - fabs(change->next->row-end_row)/2);
        conversion_x=(end_col + (change->next->col-end_col)/2);
        if (change->next->row-end_row <=7 && change->next->col-end_col<=7
                && red==0 && red_1==0 && inTriangle(conversion_x ,conversion_y))
                {
                conv_3[indx].x= conversion_x;
                conv_3[indx].y= conversion_y;
                F_Circlef_rgb(     conv_3[indx].x, conv_3[indx].y ,3, 0,0,1);
                if (change->next->length <change->length) dis=change->next->length/2;
                else dis=change->length/2;
                Final_intersection_C_3(conv_3[indx].x,conv_3[indx].y, dis );
                C_indx_3++;
                }
}
/*************************calculate the exact diversion ***point*************************/
/*************************************horisontal*************************************/
void diversion(RUN *element, int indx)
{
float end_col, diversion_x, dis;
        end_col=(element->col + (element->length-1));
        diversion_x=(end_col + (element->next->col - end_col)/2);
        if (element->next->col-end_col <=7 && inTriangle(diversion_x ,element->row ))
                {
                divr[indx].x= diversion_x;
                divr[indx].y= element->row;
                F_Circlef_rgb(divr[indx].x, divr[indx].y ,3, 0,.5,.5);
                if (element->next->length < element->length) dis=element->next->length/2;
                else dis=element->length/2;
```

```
                Final_intersection_D(divr[indx].x,divr[indx].y, dis );
                /*Divr_intersection (element , indx);*/
                D_indx++;
                }
}
/****************************calculate the exact diversion point*************************/
/*************************************vertical*************************************/
void diversion_1(RUN *element, int indx)
{
float end_row, diversion_y, dis;

        end_row=(element->row + (element->length-1));
        diversion_y=(end_row + (element->next->row - end_row)/2);
        if (element->next->row-end_row <=10 && inTriangle(element->col ,diversion_y ))
                {
                divr_1[indx].x= element->col;
                divr_1[indx].y= diversion_y;
                /*F_Circlef_rgb(  divr_1[indx].x, divr_1[indx].y ,3, 0,.5,.5);*/
                if (element->next->length < element->length) dis=element->next->length/2;
                else dis=element->length/2;

                /*Final_intersection_D_1(divr_1[indx].x,divr_1[indx].y, dis );*/

                D_indx_1++;
                }
}
/****************************calculate the exact diversion point*************************/
/*************************************diagonal (down to UP)*************************/
void diversion_2(RUN *element, int indx)
{
float end_row,end_col, diversion_y, diversion_x, dis;
int mid_y, mid_x;
unsigned char red, red_1;
        end_row=(element->row + (element->length-1));
        end_col=(element->col + (element->length-1));
        mid_y=(element->next->row + (element->next->length/2));
        mid_x=(element->next->col + (element->next->length/2));
        pixel = bitmap[element->row+(element->length/2)-1] [element->col+ (element->length/2)+1];
        red= pixel;
        pixel = bitmap[mid_y-1][mid_x+1];
        red_1= pixel;
        diversion_y=(end_row + (element->next->row - end_row)/2);
        diversion_x=(end_col + (element->next->col - end_col)/2);
        if (element->next->row-end_row <=7 && element->next->col-end_col<=7
                && red==0 && red_1==0 && inTriangle( diversion_x , diversion_y ))
                {
                divr_2[indx].x= diversion_x;
                divr_2[indx].y= diversion_y;
                F_Circlef_rgb(divr_2[indx].x, divr_2[indx].y ,3, 0,0,1);
                if (element->next->length < element->length) dis=element->next->length/2;
                else dis=element->length/2;
                Final_intersection_D_2(divr_2[indx].x,divr_2[indx].y, dis );
                D_indx_2++;
```

```
                }
}
/*****************************calculate the exact diversion point*************************/
/*****************************************diagonal**up to down*************************/
void diversion_3(RUN *element, int indx)
{
float end_row,end_col, diversion_y, diversion_x, dis;
int mid_y, mid_x;
unsigned char red, red_1;
        end_row=(element->row - (element->length-1));
        end_col=(element->col + (element->length-1));
        mid_y=(element->next->row - (element->next->length/2));
        mid_x=(element->next->col + (element->next->length/2));
        pixel = bitmap[element->row-(element->length/2)-1] [element->col+(element->length/2)-1];
        red= pixel;
        pixel = bitmap[mid_y-1][mid_x-1];
        red_1= pixel;
        diversion_y=(end_row - fabs(element->next->row - end_row)/2);
        diversion_x=(end_col + (element->next->col - end_col)/2);
        if (fabs(element->next->row-end_row) <=7 && element->next->col-end_col<=7
                && red==0 && red_1==0 && inTriangle( diversion_x , diversion_y ))
                {
                divr_3[indx].x= diversion_x;
                divr_3[indx].y= diversion_y;
                F_Circlef_rgb(divr_3[indx].x, divr_3[indx].y ,3, 0,0,1);
                if (element->next->length < element->length) dis=element->next->length/2;
                else dis=element->length/2;
                Final_intersection_D_3(divr_3[indx].x,divr_3[indx].y, dis );
                D_indx_3++;
                }
}
/************decide which triangle to process, and then save the third point in an array[2].************/
/*************************************************************************************/
void Pick_triangle(float x0, float y0,float x1,float y1, float x, float y,
float rad)
{
int x2, y2;
float Dist;
        printf("array[0].x= %4.2f, array[0].y= %4.2f,\n",array[0].x, array[0].y );
        printf("array[1].x= %4.2f, array[1].y= %4.2f,\n",array[1].x, array[1].y );
        printf("radd= %4.2f\n",rad);
        if (y0==y1)/*if horisontal*/
                {
                if(y>y0)/*upper triangle*/
                        {
                        if (x1>x0)
                                {
                                triangle = UPPER_1;
                                printf("upper triangle_1 \n");
                                array[2].x=x0+(rad/2);
                                array[2].y=y0+(rad/2);
                                array[3].x=x0+(rad/2);
                                array[3].y=y0;
```

```
                findPOint(array[0].x ,array[0].y ,array[2].x, array[2].y , 22.5);
                scan_boundaries(x0,y0,x1,array[2].x ,array[2].y);
                scan_line(x0,y0,x1,y1);
                for(int pt=0; pt<inter_count;pt++)
                        {
                        if(fabs(inter[pt].x-array[4].x)<=5 &&
                          fabs(inter[pt].y-array[0].y)<=5)
                                {
                                inter[pt].x=array[4].x;
                                inter[pt].y=array[0].y;
                                }
                        }
                scan_line(x0,y0,array[2].x,array[2].y);
                scan_line(x0,y0,array[4].x,array[4].y);
                scan_line(x1,y1,array[4].x,array[4].y);
                scan_line(x1,y1, array[2].x ,array[2].y);
                }
        if(x0>x1)
                {
                triangle = UPPER_2;
                printf("upper triangle_2 \n");
                array[2].x=x1+(rad/2);
                array[2].y=y1+(rad/2);
                array[3].x=x1+(rad/2);
                array[3].y=y0;
                findPOint(array[0].x ,array[0].y ,array[2].x, array[2].y , 157.5);
                scan_line(x0,y0,x1,y1);
                for(int pt=0; pt<inter_count;pt++)
                        {
                        if(fabs(inter[pt].x-array[4].x)<=5 &&
                          fabs(inter[pt].y-array[0].y)<=5)
                                {
                                inter[pt].x=array[4].x;
                                inter[pt].y=array[0].y;
                                }
                        }
                scan_line(x0,y0,array[2].x,array[2].y);
                scan_line(x0,y0,array[4].x,array[4].y);
                scan_line(x1,y1,array[4].x,array[4].y);
                scan_line(x1,y1, array[2].x ,array[2].y);
                }
        }
if(y<y0)/*lower triangle*/
        {
        if (x1>x0)
                {
                triangle = LOWER_1;
                printf(" lower triangle_1 \n");
                array[2].x=x0+(rad/2);
                array[2].y=y0-(rad/2);
                array[3].x=x0+(rad/2);
                array[3].y=y0;
                findPOint(array[0].x ,array[0].y ,array[2].x, array[2].y , 337.5);
```

```
                                scan_line(x0,y0,x1,y1);
                                for(int pt=0; pt<inter_count;pt++)
                                        {
                                        if(fabs(inter[pt].x-array[4].x)<=5 &&
                                          fabs(inter[pt].y-array[0].y)<=5)
                                                {
                                                inter[pt].x=array[4].x;
                                                inter[pt].y=array[0].y;
                                                }
                                        }
                                scan_line(x0,y0,array[2].x,array[2].y);
                                scan_line(x0,y0,array[4].x,array[4].y);
                                scan_line(x1,y1,array[4].x,array[4].y);
                                scan_line(x1,y1, array[2].x ,array[2].y);
                                }
                        if(x0>x1)
                                {
                                triangle = LOWER_2;
                                printf(" lower triangle_2 \n");
                                array[2].x=x1+(rad/2);
                                array[2].y=y1-(rad/2);
                                array[3].x=x1+(rad/2);
                                array[3].y=y0;
                                findPOint(array[0].x ,array[0].y ,array[2].x, array[2].y , 202.5);
                                scan_line(x0,y0,x1,y1);
                                for(int pt=0; pt<inter_count;pt++)
                                        {
                                        if(fabs(inter[pt].x-array[4].x)<=5 &&
                                          fabs(inter[pt].y-array[0].y)<=5)
                                                {
                                                inter[pt].x=array[4].x;
                                                inter[pt].y=array[0].y;
                                                }
                                        }
                                scan_line(x0,y0,array[2].x,array[2].y);
                                scan_line(x0,y0,array[4].x,array[4].y);
                                scan_line(x1,y1,array[4].x,array[4].y);
                                scan_line(x1,y1, array[2].x ,array[2].y);
                                }
                        }
                }
else if (x0==x1)/*if vertical*/
                {
                if(x>x0)/*right triangle*/
                        {
                        if (y1>y0)
                                {
                                triangle =RIGHT_1;
                                printf(" right triangle_1 \n");
                                array[2].x=x0+(rad/2);
                                array[2].y=y0+(rad/2);
                                array[3].x=x0;
                                array[3].y=y0+(rad/2);
```

```
                findPOint(array[0].x ,array[0].y ,array[2].x, array[2].y , 67.5);
                scan_line(x0,y0,x1,y1);
                for(int pt=0; pt<inter_count;pt++)
                        {
                        if(fabs(inter[pt].x-array[0].x)<=5 &&
                          fabs(inter[pt].y-array[4].y)<=5)
                                {
                                inter[pt].x=array[0].x;
                                inter[pt].y=array[4].y;
                                }
                        }
                scan_line(x0,y0,array[2].x,array[2].y);
                scan_line(x0,y0,array[4].x,array[4].y);
                scan_line(x1,y1,array[4].x,array[4].y);
                scan_line(x1,y1, array[2].x ,array[2].y);
                }
        if(y0>y1)
                {
                triangle =RIGHT_2;
                printf(" right triangle_2 \n");
                array[2].x=x1+(rad/2);
                array[2].y=y1+(rad/2);
                array[3].x=x0;
                array[3].y=y1+(rad/2);
                findPOint(array[0].x ,array[0].y ,array[2].x, array[2].y , 292.5);
                scan_line(x0,y0,x1,y1);
                for(int pt=0; pt<inter_count;pt++)
                        {
                        if(fabs(inter[pt].x-array[0].x)<=5 &&
                          fabs(inter[pt].y-array[4].y)<=5)
                                {
                                inter[pt].x=array[0].x;
                                inter[pt].y=array[4].y;
                                }
                        }
                scan_line(x0,y0,array[2].x,array[2].y);
                scan_line(x0,y0,array[4].x,array[4].y);
                scan_line(x1,y1,array[4].x,array[4].y);
                scan_line(x1,y1, array[2].x ,array[2].y);
                }
        }
if(x<x0)/*left triangle*/
        {
        if (y1>y0)
                {
                triangle=LEFT_1;
                printf(" left triangle_1 \n");
                array[2].x=x0-(rad/2);
                array[2].y=y0+(rad/2);
                array[3].x=x0;
                array[3].y=y0+(rad/2);
                findPOint(array[0].x ,array[0].y ,array[2].x, array[2].y ,112.5);
                scan_line(x0,y0,x1,y1);
```

```
                                for(int pt=0; pt<inter_count;pt++)
                                        {
                                        if(fabs(inter[pt].x-array[0].x)<=5 &&
                                          fabs(inter[pt].y-array[4].y)<=5)
                                                {
                                                inter[pt].x=array[0].x;
                                                inter[pt].y=array[4].y;
                                                }
                                        }
                                scan_line(x0,y0,array[2].x,array[2].y);
                                scan_line(x0,y0,array[4].x,array[4].y);
                                scan_line(x1,y1,array[4].x,array[4].y);
                                scan_line(x1,y1, array[2].x ,array[2].y);
                                }
                        if(y0>y1)
                                {
                                triangle=LEFT_2;
                                printf(" left triangle_2 \n");
                                array[2].x=x1-(rad/2);
                                array[2].y=y1+(rad/2);
                                array[3].x=x0;
                                array[3].y=y1+(rad/2);
                                findPOint(array[0].x ,array[0].y ,array[2].x, array[2].y , 247.5);
                                scan_line(x0,y0,x1,y1);
                                for(int pt=0; pt<inter_count;pt++)
                                        {
                                        if(fabs(inter[pt].x-array[0].x)<=5 &&
                                          fabs(inter[pt].y-array[4].y)<=5)
                                                {
                                                inter[pt].x=array[0].x;
                                                inter[pt].y=array[4].y;
                                                }
                                        }
                                scan_line(x0,y0,array[2].x,array[2].y);
                                scan_line(x0,y0,array[4].x,array[4].y);
                                scan_line(x1,y1,array[4].x,array[4].y);
                                scan_line(x1,y1, array[2].x ,array[2].y);
                                }
                        }
                }
/*****************************save triangle run length in a link list*****************************/
/*************************************************************************************************/
}
void Link_List(void)
{
unsigned char red,current,previous;
int i,j, iy, jx;
count=0;
        printf("array[0].x= %4.2f, array[0].y= %4.2f,\n",array[0].x, array[0].y );
        printf("array[1].x= %4.2f, array[1].y= %4.2f,\n",array[1].x, array[1].y );
        printf("array[2].x= %4.2f, array[2].y= %4.2f,\n",array[2].x, array[2].y );
        printf("radd= %4.2f\n",radd);
/**************scan horisontally and save in link list**********/
```

```
/*****************************************************/
        for (i=int (array[3].y+radd/2); i>= int (array[3].y-radd/2);i--)
                {
                previous=255;
                for(j= int (array[3].x-radd/2); j<= int (array[3].x+radd/2) ; j++)
                        {
                        pixel = bitmap[i][j];
                        red= pixel;
                        current=red;
                        /*white-> black*/
                        if( previous==255/*white*/&& current == 0 /*black*/)
                                {
                                /*start run length*/
                                count++;
                                /*alocate space for struct ( row, column, length) and save it in link
                                list*/
                                if (head == NULL)
                                        {
                                        head= (RUN*)malloc(sizeof(RUN));
                                        tail=head;
                                        }
                                 else
                                        { /* is not the first time */
                                        tail-> next=(RUN*)malloc(sizeof(RUN));
                                        tail = tail->next;
                                        }

                                tail-> row=i;
                                tail-> col=j;
                                tail-> length=1;
                                }
                        /*black-> black*/
                        else if( previous==0/*black*/&& current == 0 /*black*/)
                                {
                                tail-> length++;
                                }
                        /*black-> white*/
                        else if( previous==0/*black*/&& current == 255 /*white*/)
                                {
                                tail-> next = NULL;
                                }
                        previous= current;
                        }
                }
/*****************scan vertically and save in differenet link list**********/
/***********************RUN *head_1=NULL, *tail_1;***************/
/*****************************************************************/
count_1=0;
        for(j= int (array[3].x-radd/2); j<= int (array[3].x+radd/2) ; j++)
                {
                previous=255;
                for (i=int (array[3].y-radd/2); i<= int (array[3].y+radd/2);i++)
                        {
```

```
                        pixel = bitmap[i][j];
                        red= pixel;

                        current=red;
                        /*white-> black*/
                        if( previous==255/*white*/&& current == 0 /*black*/)
                                {

                                /*start run length*/
                                count_1++;
                                /*alocate space for struct ( row, column, length) and save it in link
                                list*/
                                if (head_1 == NULL)
                                        {
                                        head_1= (RUN*)malloc(sizeof(RUN));
                                        tail_1=head_1;
                                        }
                                 else
                                        { /* is not the first time */
                                        tail_1-> next=(RUN*)malloc(sizeof(RUN));
                                        tail_1 = tail_1->next;
                                        }
                                tail_1-> row=i;
                                tail_1-> col=j;
                                tail_1-> length=1;
                                }
                        /*black-> black*/
                        else if( previous==0/*black*/&& current == 0 /*black*/)
                                {
                                tail_1-> length++;
                                }
                        /*black-> white*/
                        else if( previous==0/*black*/&& current == 255 /*white*/)
                                {
                                /*printf(" tail-> length =%d\n\n",tail_1-> length  );*/
                                tail_1-> next = NULL;
                                }
                        previous= current;
                        }
                }
/**scan diagonally (down to up)and save in differenet link list*************/
/*********************RUN *head_2=NULL, *tail_2;*****************/
/*************************************************************/
count_2=0;
        for (i=int(array[3].y),j=int(array[3].x-radd/2);i>=int(array[3].y-radd/2),j<=
int (array[3].x) ;i--,j++)
                {
                previous=255;
                for(jx=j, iy=i; jx<=j+radd/2 , iy<= i+radd/2; jx++, iy++)
                        {
                        pixel = bitmap[iy][jx];
                        red= pixel;
                        current=red;
```

```
                        /*white-> black*/
                        if( previous==255/*white*/&& current == 0 /*black*/)
                                {
                                /*start run length*/
                                count_2++;
                                /*alocate space for struct ( row, column, length) and save it in link
                                list*/
                                if (head_2 == NULL)
                                        {
                                        head_2= (RUN*)malloc(sizeof(RUN));
                                        tail_2=head_2;
                                        }
                                 else
                                        { /* is not the first time */
                                        tail_2-> next=(RUN*)malloc(sizeof(RUN));
                                        tail_2 = tail_2->next;
                                        }
                                tail_2-> row=iy;
                                tail_2-> col=jx;
                                tail_2-> length=1;
                                }
                        /*black-> black*/
                        else if( previous==0/*black*/&& current == 0 /*black*/)
                                {
                                tail_2-> length++;
                                }
                        /*black-> white*/
                        else if( previous==0/*black*/&& current == 255 /*white*/)
                                {
                                /*printf(" tail-> length =%d\n\n",tail_2-> length  );*/
                                tail_2-> next = NULL;
                                }
                        previous= current;
                        }
                }

/**scan diagonally (up to down)and save in differenet link list*************/
/*********************RUN *head_3=NULL, *tail_3;*****************/
/*****************************************************************/
count_3=0;
        for (i=int(array[3].y+radd/2),j=int (array[3].x); i>=int
(array[3].y),j>=int(array[3].x-radd/2) ;i--,j--)
                {
                previous=255;
                for(jx=j, iy=i; jx<=j+radd/2 , iy>= i-radd/2; jx++, iy--)
                        {
                        pixel = bitmap[iy][jx];
                        red= pixel;

                        current=red;
                        /*white-> black*/
                        if( previous==255/*white*/&& current == 0 /*black*/)
                                {
```

```
                                        /*start run length*/
                                        count_3++;
                                        /*alocate space for struct ( row, column, length) and save it in link
                                        list*/
                                        if (head_3 == NULL)
                                                {
                                                head_3= (RUN*)malloc(sizeof(RUN));
                                                tail_3=head_3;
                                                }
                                         else
                                                { /* is not the first time */
                                                tail_3-> next=(RUN*)malloc(sizeof(RUN));
                                                tail_3 = tail_3->next;
                                                }
                                        tail_3-> row=iy;
                                        tail_3-> col=jx;
                                        tail_3-> length=1;
                                        }
                                /*black-> black*/
                                else if( previous==0/*black*/&& current == 0 /*black*/)
                                        {
                                        tail_3-> length++;
                                        }
                                /*black-> white*/
                                else if( previous==0/*black*/&& current == 255 /*white*/)
                                        {
                                        /*printf(" tail-> length =%d\n\n",tail_2-> length  );*/
                                        tail_3-> next = NULL;
                                        }
                                previous= current;
                                }
                        }
}
/**************************find coversion and diversion points.*****************************/
void find_Conv_Div(void)
{
RUN *change, *element;
int L_C_index=0, R_C_index=0, L_D_index=0 ,R_D_index=0 ;
int e, c, cc,ee ;
int end_col;
int x, y, xx, yy, xxx, yyy;

/****search link list (head)for conversions and diversions**********/
/***************************Horisontal*********************************/
        if(head!=NULL)
                {
                for (change=head; change!=NULL; change=change->next)
                        {
                        /* Start ===find conversion points  */
                        for (element=head;element!=NULL;element=element->next)
                                {
                                if(element->row == change->row-1&& element->length >=4 &&
                                change->length>=4)
```

```
{
L_C_index=0;/* left conversion index  */
L_D_index=0;/* left diversion index  */
for (c=change->col;c<=((change->col+change->length)-1);
c++)
        {
        for(e=element->col;e<=((element->col+element-
        >length)-1); e++)
                {
                if (c==e)
                        {
                        L_C_index++;
                        L_D_index++;
                        }
                }
        }
if(L_C_index >=1 && element->row == change->next-
>row-1&&change->next->length>=4)
        {
        R_C_index=0;/* right conversion index  */
        for (c=change->next->col;c<=((change->next-
        >col+change->next->length)-1);c++)
                {
                for(e=element->col;e<=((element-
                >col+element->length)-1); e++)
                        {
                        if (c==e)R_C_index++;
                        }
                }
        if(R_C_index>=1)
                {
                conversion(change, C_indx);
                }
        }
if(L_D_index >=1 && element->next!= NULL && element-
>next->row ==change->row-1 && element->next-
>length>=4)
        {

        R_D_index=0;
        for (c=change->col; c<=((change->col+change-
        >length)-1); c++)
                {
                for(e=element->next->col;
                  e<=((element->next->col+element->next-
                        >length)-1);e++)
                        {
                        if (c==e)R_D_index++;
                        }
                }
        if(R_D_index>=1)
                {
                diversion(element, D_indx);
```

```
                                                }
                                        }
                                }
                        }
                }
        }

/*************search link list (head_1) for conversions and diversions**********/
/***************************Vertical ************************************/
if(head_1!=NULL)
        {
        for (change=head_1; change!=NULL; change=change->next)
                {
                /* Start ===find conversion points  */
                for (element=head_1;element!=NULL;element=element->next)
                        {
                        if(element->col == change->col+1&& element->length >=4 &&
                        change->length>=4)
                                {
                                L_C_index=0;/* left conversion index  */
                                L_D_index=0;/* left diversion index  */
                                for (c=change->row;c<=((change->row+change->length)-1);
                                c++)
                                        {
                                        for(e=element->row;e<=((element->row+element-
                                        >length)-1); e++)
                                                {
                                                if (c==e)
                                                        {
                                                        L_C_index++;
                                                        L_D_index++;
                                                        }
                                                }
                                        }
                                if(L_C_index >=1 && element->col ==
                                change->next->col+1&&change->next->length>=4)
                                        {
                                        R_C_index=0;/* right conversion index  */
                                        for (c=change->next->row;c<=((change->next-
                                        >row+change->next->length)-1); c++)
                                                {
                                                for(e=element->row;e<=((element-
                                                >row+element->length)-1); e++)
                                                        {
                                                        if (c==e)R_C_index++;
                                                        }
                                                }
                                        if(R_C_index>=1)
                                                {
                                                conversion_1(change, C_indx_1);
                                                }
                                        }
                                if(L_D_index >=1 && element->next!= NULL && element-
```

```
                                        >next->col ==change->col+1 && element->next-
                                        >length>=4)
                                                {
                                                R_D_index=0;
                                                for (c=change->row; c<=((change->row+change-
                                                >length)-1); c++)
                                                        {
                                                        for(e=element->next->row;
                                                        e<=((element->next->row+element->next-
                                                        >length)-1); e++)
                                                                {
                                                                if (c==e)R_D_index++;
                                                                }
                                                        }
                                                if(R_D_index>=1)
                                                        {
                                                        diversion_1(element, D_indx_1);

                                                        }
                                                }
                                        }
                                }
                        }
/**************search link list (head_2) for conversions and diversions**********/
/***************************Diagonal(down to up) ************************/
if(head_2!=NULL)
                        {
                        for (change=head_2; change!=NULL; change=change->next)
                                {
                                /* Start ===find conversion points  */
                                for (element=head_2;element!=NULL;element=element->next)
                                        {
                                        y= change->row-1;
                                        x= change->col+1;
                                        if(abs(element->col-x) == abs(element->row-y) && element->length
                                        >=4 &&change->length>=4)
                                                {
                                                L_C_index=0;/* left conversion index  */
                                                L_D_index=0;/* left diversion index  */

                                                for (c=change->row, cc=change->col;
                                                    c<=((change->row+change->length)-1),
                                                    cc<=((change->col+change->length)-1);
                                                    c++, cc++)

                                                        {
                                                        for(e=element->row, ee=element->col;
                                                           e<=((element->row+element->length)-1),
                                                           ee<=((element->col+element->length)-1);
                                                                e++, ee++)
                                                                {
                                                                if (ee==cc+1 && e==c-1)
```

```
                                {
                                L_C_index++;
                                L_D_index++;
                                }
                        }
                }
if(change->next !=NULL)
                {
                yy= change->next->row-1;
                xx= change->next->col+1;
                }
if(L_C_index >=1 && change->next !=NULL &&
 abs(element->col-xx) == abs(element->row-yy) && change-
>next->length >=4 )
                {
                R_C_index=0;/* right conversion index  */
                for (c=change->next->row, cc=change->next->col;
                   c<=((change->next->row+change->next->length)-
                        1),
                   cc<=((change->next->col+change->next
                        >length)-1);c++, cc++)
                        {
                        for(e=element->row, ee=element->col;
                        e<=((element->row+element->length)-1),
                        ee<=((element->col+element->length)-1
                        e++, ee++)
                                {
                                if (ee==cc+1 && e==c-1)
                                        {
                                        R_C_index++;
                                        }
                                }
                        }
                if(R_C_index>=1)
                        {
                        conversion_2(change, C_indx_2);
                        }
                }
if(L_D_index >=1 && element->next!= NULL &&
  abs(element->next->col-x)== abs(element->next->row-y)
 &&element->next->length >=4)
                {
                R_D_index=0;
                for (c=change->row, cc=change->col;
                   c<=((change->row+change->length)-1),
                   cc<=((change->col+change->length)-1);
                   c++, cc++)

                        {
                        for(e=element->next->row, ee=element-
                        >next->col;
                        e<=((element->next->row+element->next-
                        >length)-1),
```

```
                                                ee<=((element->next->col+element->next-
                                                >length)-1);
                                                e++,ee++)
                                                        {
                                                        if (ee==cc+1 && e==c-1)
                                                                {
                                                                R_D_index++;
                                                                }
                                                        }
                                                }
                                        if(R_D_index>=1)
                                                {
                                                diversion_2(element, D_indx_2);
                                                }
                                        }
                                }
                        }
                }
/*************search link list (head_3) for conversions and diversions**********/
/**************************Diagonal( up to down) ************************/
if(head_3!=NULL)
                {
                for (change=head_3; change!=NULL; change=change->next)
                        {
                        /* Start ===find conversion points  */
                        for (element=head_3;element!=NULL;element=element->next)
                                {
                                y= change->row-1;
                                x= change->col-1;
                                if(abs(element->col-x) == abs(element->row-y) && element->length
                                >=4 &&change->length>=4)
                                        {
                                        L_C_index=0;/* left conversion index  */
                                        L_D_index=0;/* left diversion index  */
                                        for (c=change->row, cc=change->col;
                                            c>=((change->row-change->length)-1),
                                            cc<=((change->col+change->length)-1);
                                            c--, cc++)
                                                {
                                                for(e=element->row, ee=element->col;
                                                   e>=((element->row-element->length)-1),
                                                   ee<=((element->col+element->length)-1);
                                                e--, ee++)
                                                        {
                                                        if (ee==cc-1 && e==c-1)
                                                                {
                                                                L_C_index++;
                                                                L_D_index++;
                                                                }
                                                        }
                                                }
                                        if(change->next !=NULL)
```

```
        {
        yy= change->next->row-1;
        xx= change->next->col-1;
        }
if(L_C_index >=1 && change->next !=NULL &&
 abs(element->col-xx) == abs(element->row-yy) && change-
>next->length >=4 )
        {
        R_C_index=0;/* right conversion index  */
        for (c=change->next->row, cc=change->next->col;
         c>=((change->next->row-change->next->length)-
                1),
           cc<=((change->next->col+change->next-
                >length)-1);
                c--, cc++)
                {
                for(e=element->row, ee=element->col;
                e>=((element->row-element->length)-1),
                ee<=((element->col+element->length)-1);
                 e--, ee++)
                        {
                        if (ee==cc-1 && e==c-1)
                                {
                                R_C_index++;
                                }
                        }
                }
        if(R_C_index>=1)
                {
                conversion_3(change, C_indx_3);
                }
        }
if(L_D_index >=1 && element->next!= NULL &&
  abs(element->next->col-x)== abs(element->next->row-y)
        &&element->next->length >=4)
        {
        R_D_index=0;
        for (c=change->row, cc=change->col;
           c>=((change->row-change->length)-1),
           cc<=((change->col+change->length)-1);
           c--, cc++)
                {
                for(e=element->next->row, ee=element-
                >next->col;
                e>=((element->next->row-element->next-
                >length)-1),
                ee<=((element->next->col+element->next-
                >length)-1);
                e--,ee++)
                        {
                        if (ee==cc-1 && e==c-1)
                                {
                                R_D_index++;
```

```
                                                                                }
                                                                        }
                                                                }
                                                        if(R_D_index>=1)
                                                                {
                                                                diversion_3(element, D_indx_3);

                                                                }
                                                        }
                                                }
                                        }
                                }
                        }
}
/*=========================find symmetrical lines around this line====================*/
float x_point_symmetry(float px,float py, float x0,float y0,float x1,float y1)
{
float X;
float degree,degree_0, final_degree, distance_0, theta;
        degree=degrees(x0,y0, x1 ,y1 );
        degree_0=degrees(x0,y0,px,py);
        distance_0=findDis( x0 ,y0,px,py);
        final_degree= degree+(degree-degree_0);
        X=extendx(x0,y0,final_degree,distance_0 );
return(X);
}
/*=============================find symmetrical lines around this line=================*/
float y_point_symmetry(float px,float py, float x0,float y0,float x1,float y1)
{
float Y;
float degree,degree_0, final_degree, distance_0, theta;
        degree=degrees(x0,y0, x1 ,y1 );
        degree_0=degrees(x0,y0,px,py);
        distance_0=findDis( x0 ,y0,px,py);
        final_degree= degree+(degree-degree_0);
        Y=extendy(x0,y0,final_degree,distance_0 );
return(Y);
}
/*=========================find symmetrical lines around 22.5 degree=================*/
void First_symmetry(void)
{
float x0, y0, x1,y1;
float degree,degree_0,degree_1, final_degree, distance_0,distance_1,  theta;
        /*Draw_line( array[4].x ,array[4].y,array[4].x ,array[0].y);*/
        Draw_line( array[0].x ,array[0].y,array[4].x ,array[4].y);
        for( int l=0; l< line_count; l++)
                {
                Draw_line( line[l].x0 ,line[l].y0,
                        line[l].x1 ,line[l].y1);
                printf( " line[%d].x0= %4.2f ,line[%d].y0= %4.2f\n",
                        l,line[l].x0,
                        l,line[l].y0);
                if(triangle==UPPER_1 && line[l].x0 <= array[4].x+5 && line[l].x1<=array[4].x+5
```

```
                || triangle==LOWER_1 && line[l].x0 <= array[4].x+5 && line[l].x1<=array[4].x+5)
                        {
x0=x_point_symmetry(line[l].x0,line[l].y0,array[0].x,array[0].y,array[4].x,array[4].y);
y0=y_point_symmetry(line[l].x0,line[l].y0,array[0].x,array[0].y,array[4].x,array[4].y);
                        if(fabs(array[2].x-x0)<=7 && fabs(array[2].y-y0)<=7)
                                {
                                x0=array[2].x;
                                y0=array[2].y;
                                }
                        F_Circlef_rgb(x0 , y0 ,2, 1,0,1);
x1=x_point_symmetry(line[l].x1,line[l].y1,array[0].x,array[0].y,array[4].x,array[4].y);
y1=y_point_symmetry(line[l].x1,line[l].y1,array[0].x,array[0].y,array[4].x,array[4].y);
                        if(fabs (array[2].x-x1)<=7 && fabs(array[2].y-y1)<=7)
                                {
                                x1=array[2].x;
                                y1=array[2].y;
                                }
                        F_Circlef_rgb(x1 , y1 ,2, 1,0,1);
                        Draw_line( x0 ,y0,x1 ,y1);
                        }
                else if(triangle==UPPER_2 && line[l].x0 >= array[4].x-5 && line[l].x1>=array[4].x-5
                   || triangle==LOWER_2 && line[l].x0 >= array[4].x-5 && line[l].x1>=array[4].x-5)
                        {
x0=x_point_symmetry(line[l].x0,line[l].y0,array[0].x,array[0].y,array[4].x,array[4].y);
y0=y_point_symmetry(line[l].x0,line[l].y0,array[0].x,array[0].y,array[4].x,array[4].y);
                        if(fabs(array[2].x-x0)<=7 && fabs(array[2].y-y0)<=7)
                                {
                                x0=array[2].x;
                                y0=array[2].y;
                                }
                        F_Circlef_rgb(x0 , y0 ,2, 1,0,1);
x1=x_point_symmetry(line[l].x1,line[l].y1,array[0].x,array[0].y,array[4].x,array[4].y);

y1=y_point_symmetry(line[l].x1,line[l].y1,array[0].x,array[0].y,array[4].x,array[4].y);
                        if(fabs (array[2].x-x1)<=7 && fabs(array[2].y-y1)<=7)
                                {
                                x1=array[2].x;
                                y1=array[2].y;
                                }
                        F_Circlef_rgb(x1 , y1 ,2, 1,0,1);
                        Draw_line( x0 ,y0,x1 ,y1);
                        }
                else if(triangle==RIGHT_1 && line[l].y0 <= array[4].y+5 && line[l].y1<=array[4].y+5
                   || triangle==LEFT_1  && line[l].y0 <= array[4].y+5 && line[l].y1<=array[4].y+5)
                        {
x0=x_point_symmetry(line[l].x0,line[l].y0,array[0].x,array[0].y,array[4].x,array[4].y);
y0=y_point_symmetry(line[l].x0,line[l].y0,array[0].x,array[0].y,array[4].x,array[4].y);
                        if(fabs(array[2].x-x0)<=7 && fabs(array[2].y-y0)<=7)
                                {
                                x0=array[2].x;
                                y0=array[2].y;
                                }
                        F_Circlef_rgb(x0 , y0 ,2, 1,0,1);
```

```
x1=x_point_symmetry(line[l].x1,line[l].y1,array[0].x,array[0].y,array[4].x,array[4].y);
y1=y_point_symmetry(line[l].x1,line[l].y1,array[0].x,array[0].y,array[4].x,array[4].y);
                                if(fabs (array[2].x-x1)<=7 && fabs(array[2].y-y1)<=7)
                                        {
                                        x1=array[2].x;
                                        y1=array[2].y;
                                        }
                                F_Circlef_rgb(x1 , y1 ,2, 1,0,1);
                                Draw_line( x0 ,y0,x1 ,y1);
                                }
                        else if(triangle==RIGHT_2 && line[l].y0 >= array[4].y-5 && line[l].y1>=array[4].y-5
                           || triangle==LEFT_2  && line[l].y0 >= array[4].y-5 && line[l].y1>=array[4].y-5)
                                {
x0=x_point_symmetry(line[l].x0,line[l].y0,array[0].x,array[0].y,array[4].x,array[4].y);
y0=y_point_symmetry(line[l].x0,line[l].y0,array[0].x,array[0].y,array[4].x,array[4].y);
                                if(fabs(array[2].x-x0)<=7 && fabs(array[2].y-y0)<=7)
                                        {
                                        x0=array[2].x;
                                        y0=array[2].y;
                                        }
                                F_Circlef_rgb(x0 , y0 ,2, 1,0,1);
x1=x_point_symmetry(line[l].x1,line[l].y1,array[0].x,array[0].y,array[4].x,array[4].y);
y1=y_point_symmetry(line[l].x1,line[l].y1,array[0].x,array[0].y,array[4].x,array[4].y);
                                if(fabs (array[2].x-x1)<=7 && fabs(array[2].y-y1)<=7)
                                        {
                                        x1=array[2].x;
                                        y1=array[2].y;
                                        }
                                F_Circlef_rgb(x1 , y1 ,2, 1,0,1);
                                Draw_line( x0 ,y0,x1 ,y1);
                                }
                        }
/* Save();
Second_symmetry();
Save();
Thirs_symmetry();
Save();*/
}
/*============================find lines and save them in a
linestructure=======================*/
/*============================**********************************==============
============*/
void Find_lines(void)
{
int No_line=0;
float degree, theta, x,y,distance;
unsigned char red,current;
        for( int pt=0; pt< inter_count; pt++)
                for( int ptt= (pt+1); ptt< inter_count; ptt++)
                        {
                        degree=degrees(inter[pt].x,inter[pt].y,inter[ptt].x,inter[ptt].y);
                        distance=findDis(inter[pt].x,inter[pt].y,inter[ptt].x,inter[ptt].y);
                        theta=degree*PI/180;
```

```
                        No_line=0;
                        for (int r=1; r<=distance; r++)
                                {
                                x=inter[pt].x + r*cos(theta);
                                y=inter[pt].y + r*sin(theta);
                                pixel= bitmap[int(y)][int(x)];
                                red=pixel;
                                current=red;
                                if( current==255 && No_line==0)
                                        {
                                        No_line=1;
                                        }
                                }
                        /*Draw_line( inter[pt].x, inter[pt].y,inter[ptt].x,inter[ptt].y);*/
                        if(No_line==0)
                                {
                                line[line_count].x0=inter[pt].x;
                                line[line_count].y0=inter[pt].y;
                                line[line_count].x1=inter[ptt].x;
                                line[line_count].y1=inter[ptt].y;
                                line_count++;
                                }
                        }
                First_symmetry();

}
/*=============================================================================
==================*/
/*==============================find max and min of the runlength ->length
============================*/
void find_max_min(){
RUN *change;
        if(head != NULL)
                {
                for(change=head; change!=NULL; change=change->next)
                        {
                        if(change->length > Max)Max=change->length;
                        if(change->length < Min)Min=change->length;
                        }
                printf("\n count =%d\n", count);
                printf(" Min =%d\n", Min );
                printf(" Max =%d\n\n", Max );
                }
}

/*****************************************************************************************/
/*==============================Bitmap==================================*
void Bitmap(float valueControl){
unsigned char red, green, blue, alpha;
double hue, sat, val;
double value;
int i,j,x;
double r , g, b;
```

```
int RR , GG, BB;
        bitmap = (unsigned int**)malloc (rows*sizeof(int*));
        for(x=0;x<rows; x++)
        {
          bitmap[x]   =(unsigned int*)malloc(cols*sizeof(int));
        }

              for(i=rows-1 ;i>=0;i--)
                        for(j=0;j< cols;j++)
                        {
                        pixel = pixmap[i][j];
                        red= pixel;
                        green= (pixel >> 8) & 0xff;
                        blue= (pixel >> 16) & 0xff;
                        alpha = (pixel >> 24);
                        RGBtoHSV( red,  green,  blue,  hue,  sat, val);
                        /*printf("  val =%f\n",val);*/
                        if(val < valueControl){red=0;green=0;blue=0; }
                        else {red=255;green=255;blue=255; }
                        pixel =  alpha << 24 | blue  << 16 | green  << 8 | red ;
                        bitmap[i][j]= pixel;
                        /*printf("  bitmap[%d][%d]  =%d\n", i,j,bitmap[i][j] );*/
                        }
}

/*=======================write out the data to another file=======================*/
void write_ppm(){
        int i,j;            /*pixel arranged in pixmap*/
        char red, green, blue, alpha;
        if((out=fopen(ppmout,"w"))==NULL)/*open the file*/
                {
                printf("\n ERROR' could not open a file to write to\n");
                return;
                }
        else       {
/*output the header information*/
                fputc( ppm.magicn[0],out );
                fputc( ppm.magicn[1], out );
                fprintf( out , "\n%s\t", ppm.w );
                fprintf(  out, "%s\n",  ppm.h );
                fprintf(  out, "%s",  ppm.cvalue);
                fputc(n, out);  /*output the the last character in the header*/
/*========writtning pixels to a copy file===========*/
                for(i= rows-1 ;i>=0;i--)
                        for(j=0;j<cols;j++)
                        {
                        pixel = bitmap[i][j];
                        red= pixel;
                        green= (pixel >> 8) & 0xff;
                        blue= (pixel >> 16) & 0xff;
                        alpha = (pixel >> 24);
                        fputc(red, out );
                        fputc(green, out);
```

```
                                fputc(blue, out);
                                }
        fclose(out);
        }
}



/*=======================redisplay the ppm image=======================*/
void display_Pixmap(){
unsigned int *buffer;
/* clear window */
glRasterPos2i(0,0);
        buffer =(unsigned int*)malloc(cols *rows *sizeof(int));
        for(int i=rows-1; i>=0; i--)
                for (int j=0; j<cols ; j++)
                        {
                        buffer[(i*cols+j)]=pixmap[i][j];
                        }
glDrawPixels(ppm.width,ppm.height,GL_RGBA,GL_UNSIGNED_BYTE,buffer );
glFlush();
}

/*=======================redisplay the ppm image=======================*/
void display_Bitmap(){
unsigned int *buffer;
/* clear window */
glRasterPos2i(0,0);
        buffer =(unsigned int*)malloc(cols *rows *sizeof(int));
        for(int i=rows-1; i>=0; i--)
                for (int j=0; j<cols ; j++)
                        {
                        buffer[(i*cols+j)]=bitmap[i][j];
                        }
glDrawPixels(ppm.width,ppm.height,GL_RGBA,GL_UNSIGNED_BYTE,buffer );
}

/*=========================================================================*/
/* takes the values stored on the linked list frees the memory and returns
the head pointer */
RUN  * clear(){
        RUN *change;
        change=head;

        while(change !=NULL)
                {
                head=head->next;
                free(change);
                change=head;
                }
        return head;
}
```

```
/*=========================================================================*/
/* takes the values stored on the linked list frees the memory and returns
the head pointer */
RUN  * clear_1(){
        RUN *change;
        change=head_1;
        while(change !=NULL)
                {
                head_1=head_1->next;
                free(change);
                change=head_1;
                }
        return head_1;
}

/*=========================================================================*/
/* takes the values stored on the linked list frees the memory and returns
the head pointer */
RUN  * clear_2(){
        RUN *change;
        change=head_2;

        while(change !=NULL)
                {
                head_2=head_2->next;
                free(change);
                change=head_2;
                }
        return head_2;
}

/*=========================================================================*/
/* takes the values stored on the linked list frees the memory and returns
the head pointer */
RUN  * clear_3(){
        RUN *change;
        change=head_3;

        while(change !=NULL)
                {
                head_3=head_3->next;
                free(change);
                change=head_3;
                }
        return head_3;
}

/*=========================================================================*/
/*=======================clear all=========================*/
void clear_all(void)
{
RUN *change;
```

```
        imageflag=1;
        edit_flag=1;
        Pick_triangle_flag=0;
        triangle_flag=0;
        process_flag=0;
        inimagespace=1;
        finish_flag=0;
        Center=0;
        save_flag=0;
        array[0].x=0;
        array[0].y=0;
        array[0].rad=0;
        array[0].color[0]=1;
        array[0].color[1]=0;
        array[0].color[2]=0;
        array[1].x=0;
        array[1].y=0;
        array[1].rad=0;
        array[1].color[0]=0;
        array[1].color[1]=1;
        array[1].color[2]=0;
        array[2].x=0;
        array[2].y=0;
        array[2].rad=0;
        array[2].color[0]=0;
        array[2].color[1]=1;
        array[2].color[2]=0;
        array[3].x=0;
        array[3].y=0;
        array[3].rad=0;
        array[3].color[0]=0;
        array[3].color[1]=1;
        array[3].color[2]=0;
        radd=0;
        C_indx=0;
        D_indx=0;
        C_indx_1=0;
        D_indx_1=0;
        C_indx_2=0;
        D_indx_2=0;
        C_indx_3=0;
        D_indx_3=0;
        inter_count=0;
        line_count=0;
        clear();/*free link list*/
        clear_1();/*free vertical link list*/
        clear_2();/*free diagonal (down ti up) link list*/
        clear_3();/*free diagonal (up ti down) link list*/
}

/**********************************mouse, mouse, mousem mouse**********************/
/*****************************************************************************************/
```

```
void clicks(int button, int state, int x, int y){
    y = (ppm.height+20) - y;
    x =  x-20;
  if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
          {
          if(inValueMenu(x, y)&& process_flag!=1)
                  {
                  tracking_M=1;
                  imageflag=1;
                  /*printf("Mx=%d, My= %d\n", Mx,My);*/
                  value= ((float) x)/200;
                  /*printf("value= %f\n", value);*/
                  MX=x;
                  edit_flag=1;

                  glutPostRedisplay();
                  }
          if (inImageBoundaries(x, y))
                  {
                  printf("x=%d, y= %d\n", x,y);
                  if(imageflag==1)
                          {
                          inimagespace=1;
                          if(Center!=2)findCenter(x, y);
                          edit_flag=0;
                          }
                  if(Pick_triangle_flag==1&& triangle_flag==0)
                          {
                          Pick_triangle(array[0].x,array[0].y,array[1].x,array[1].y, x,  y, radd);
                          triangle_flag=1;
                          }
                  glutPostRedisplay();
                  }
          if (inPickTriangleButton( x, y))
                  {
                  if(Center==2)
                          {
                          printf("PICK ONE SIDE \n");
                          Pick_triangle_flag=1;
                          edit_flag=1;
                          }
                  }
          if (inProcessButton( x, y))
                  {
                  if(Pick_triangle_flag==1)
                          {
                          printf("...PROCESSING ..... \n");
                          Link_List();
                          find_max_min();
                          find_Conv_Div();
                          process_flag=1;
                          edit_flag=1;
                          glutPostRedisplay();
```

```
                        }
                }
        if (inFinishButton( x, y))
                {
                if(process_flag==1 &&finish_flag==0)
                        {
                        /*Find_lines();*/
                        finish_flag=1;
                        glutPostRedisplay();
                        process_flag=0;
                        Pick_triangle_flag=0;
                        }


                }
        if (inSaveButton( x, y))
                {
                write_ppm();
                printf("\nThe imege was saved to '%s'\n", ppmout);
                }
        }
if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        {
        if(inimagespace==1&&imageflag==1&& save_flag==1)
                        {
                        if (Center==1)
                                {
                                if ((fabs(array[0].x- array[1].x))<=8)array[1].x=array[0].x;
                                if ((fabs(array[0].y- array[1].y))<=8)array[1].y=array[0].y;
                                radd=findDis(array[0].x, array[0].y,array[1].x, array[1].y);
                                printf("array[0].x= %4.2f, array[0].y= %4.2f\n",array[0].x, array[0].y);
                                printf("array[1].x= %4.2f, array[1].y= %4.2f\n",array[1].x, array[1].y);
                                printf("radd= %4.2f\n",radd);
                                find_other_centers(array[0].x,array[0].y,radd);
                                Center=2;
                                }
                        else if (Center==0)
                                {
                                edit_flag=1;
                                Center=1;
                                }
                        }
        glutPostRedisplay();
        }

/**** if middle button then reset the selection and start again picking the points.**********/
if(button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
        {
        clear_all();
        glutPostRedisplay();
        }
}
```

```
/*=========================================================================*/
/*============================m_Motion=====================================*/
/*
    save the new x and y in the structure while the mouse is tracking
*/
void m_Motion(int Mx, int My)
{
My = (ppm.height+20) - My;       /* reverse y, so zero at bottom , and max at
top  */
Mx =  Mx-20;
/* if tracking Left button */
/* *************************/
          if(tracking_M)
                    {
                    if ( inValueMenu(Mx, My) && process_flag!=1 )
                              {
                              imageflag=1;
                              /*printf("Mx=%d, My= %d\n", Mx,My);*/
                              value= ((float) Mx)/200;
                              /*printf("value= %f\n", value);*/
                              MX=Mx;
                              edit_flag=1;
                              glutPostRedisplay();
                              }
                    }
}


/***********************************main, main, main, main*********************************/
/******************************************************************************************/
void General(){
glClear(GL_COLOR_BUFFER_BIT);
int i, j, c=0;
RUN *change;
float distance, angle;
          Draw_Menue();
          DrawPoint( MX, -29);
float XX, YY;
          /*distance= findDis( 20, 10, 10.3, 20.4);
          printf(" distance_1  =%f\n", distance );
          angle= degrees(0, 0, 10, 0);
          angle= degrees(0, 0, 0, 10);
          angle= degrees(0, 0, -10, 0);
          angle= degrees(0, 0, 0, -10);*/
          if (imageflag==0)
                    {
                    value=0 ;
                    writeValue();
                    display_Pixmap();
                    }
          else
                    {
                    writeValue();
```

```
            Bitmap(value);
            display_Bitmap();
            }
if (imageflag==1 && inimagespace==1)
                        {
                        if(Center!=2 && edit_flag!=1)
                                    {
                                    check_if_symetrical(int (Cxj),int (Cyi));
                                    }
                        if(Center==2)
                                    {
                                    Circlef_rgb( array[0].x  ,array[0].y  ,radd, 1,0,0);
                                    Circlef_rgb( array[1].x  ,array[1].y  ,radd, 0,1,0);
                                    /*printf(" indx_y  =%d\n", indx_y  );
                                    printf(" indx_x  =%d\n", indx_x  );*/
                                    for (i=0; i<indx_y; i++)
                                                for (j=0; j<indx_x; j++)
                                                            {
                                                /*printf(" centr[%d ][%d ].x =%d\n",i,j,centr[i][j].x );*/
                                                            F_Circlef_rgb( centr[i][j].x ,centr[i][j].y , 4, 1,0,0);
                                                            Circlef_rgb( centr[i][j].x ,centr[i][j].y , radd, 1,1,0);
                                                            star_area(centr[i][j].x, centr[i][j].y, radd, 1,0,1);
                                                            polygon(centr[i][j].x, centr[i][j].y , radd );
                                                            /*define the repeated unit coorgingly*/
                                                            draw_two_triangles(array[0].x,array[0].y,
                                                                                    array[1].x,array[1].y,
                                                                                    radd);
                                                            }

                                    }

                        if (Pick_triangle_flag==1&& edit_flag !=1)
                                    {
                                    /*F_Circlef_rgb(  array[2].x , array[2].y ,radd/25, 0,1,0);*/
                                    highlight_rectangle(array[0].x , array[0].y,
                                                            array[1].x , array[1].y,
                                                            array[2].x , array[2].y);
                                    /*F_Circlef_rgb(  array[3].x , array[3].y ,radd/25, 1,1,0);*/
                                    /*for( int pt=0; pt< inter_count; pt++)
                                                {
                                                F_Circlef_rgb(inter[pt].x, inter[pt].y ,2, 1,1,1);
                                                printf(" inter[%d].x =%f\n",pt, inter[pt].x);
                                                printf(" inter[%d].y =%f\n",pt, inter[pt].y);
                                                }*/

                                    Draw_line(array[0].x ,array[0].y, array[4].x , array[4].y );
                                    }
                        if (process_flag==1)
                                    {
                                    /*this should be moved back to clicks after finishing*/
                                    Draw_line(array[0].x ,array[0].y, array[4].x , array[4].y );
                                    for( int pt=0; pt< inter_count; pt++)
                                                {
```

```
                                        F_Circlef_rgb(inter[pt].x, inter[pt].y ,2, 0,0,1);
                                        printf(" inter[%d].x =%f\n",pt, inter[pt].x);
                                        printf(" inter[%d].y =%f\n",pt, inter[pt].y);
                                        }
                                }
                        if(finish_flag==1)
                                {
                                Find_lines();
                                for( int pt=0; pt< inter_count; pt++)
                                        {
                                        F_Circlef_rgb(inter[pt].x, inter[pt].y ,2, 1,1,1);
                                        }
                                }
                        Circlef_rgb( array[0].x  ,array[0].y  ,array[0].rad,
                                array[0].color[0],
                                array[0].color[1],
                                array[0].color[2]);
                        F_Circlef_rgb( array[0].x , array[0].y ,array[0].rad/8,
                                array[0].color[0],
                                array[0].color[1],
                                array[0].color[2]);
                        Circlef_rgb( array[1].x  ,array[1].y  ,array[1].rad,
                                array[1].color[0],
                                array[1].color[1],
                                array[1].color[2]);
                        F_Circlef_rgb( array[1].x , array[1].y ,array[1].rad/8,
                                array[1].color[0],
                                array[1].color[1],
                                array[1].color[2]);

                }

glutSwapBuffers();
}

/*****************************************************************************************/
/*
  Main program
*/

int main(int argc, char* argv[]){
int Window_width, x_end;
        /*check if the command line has the file manes to read and write*/
        if( argc==1)
                {
                printf("\n ERROR:No file(s) were found to read or write;\n command format must
                be in the form of 'warp in.ppm out.ppm'\n");
                return 0;
                }
        if( argc ==2)
                {
                read_ppm(argv[1]);
                }
```

```
        if( argc >=3)
                {
                read_ppm(argv[1]);
                ppmout=argv[2];
                }

        glutInit(&argc, argv);

  /* open window and establish coordinate system on it */
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
        if (ppm.width>=400)
                {
                Window_width= ppm.width+40;
                x_end=ppm.width+20;
                }
        else
                {
                Window_width= 400+40;
                x_end=400+20;
                }

        glutInitWindowSize( Window_width, ppm.height+140);
        glutCreateWindow("Rima _ppm image");
        gluOrtho2D(-20,  x_end , -120, ppm.height+20 );
        glutDisplayFunc(General);
        glutMouseFunc(clicks);
        glutMotionFunc(m_Motion);
        glClearColor(RGBWHITE,1);
        glutMainLoop();
  return 0;
}
```

VITA

PERSONAL DATA

Name: Rima Ahmad Al Ajlouni
Address: P.O.Box 1450
Amman, 11941, Jordan
E-mail: rqajlouni@neo.tamu.edu

EDUCATION

TEXAS A&M UNIVERSITY. College Station, TX
Ph.D. in Architecture, (August 2005).

THE UNIVERSITY OF JORDAN. Amman, Jordan.
M. Sc in Architecture Engineering, Faculty of Engineering & Technology, (January 1999).

THE UNIVERSITY OF JORDAN. Amman, Jordan.
B. Sc in Architecture Engineering, Faculty of Engineering & Technology, (June 1994).

WORK EXPERIENCE

Employer: Al-Balqa Applied University in Jordan.
Position: Assistant Teacher. Grade A
Dates: September 2001 to September 2002
Supervisor: Amman College of Engineering/ Dept. of Civil Engineering

Employer: Al-Balqa Applied University in Jordan.
Position: Assistant Teacher. Grade A
Dates: September 1999 to September 2001
Supervisor: Institute of Traditional Islamic Arts.

Employer: Al Mihrab Architectural Co. Jordan.
Position: Head Architect.
Dates: February 1996 to December 1998

Employer: Specialist Engineering Office (S.E.O.) Jordan.
Position: Architect.
Dates: August 1994 to January 1996.