Image Quality Analysis Using GLCM

Dhanashree Gadkari
University of Central Florida

IMAGE QUALITY ANALYSIS USING GLCM

by

DHANASHREE GADKARI
B.S.E.E. University of Pune, 2000

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Modeling and Simulation
in the College of Arts and Sciences
at the University of Central Florida
Orlando, Florida

Fall Term
2004

ABSTRACT

Gray level co-occurrence matrix has proven to be a powerful basis for use in texture classification. Various textural parameters calculated from the gray level co-occurrence matrix help understand the details about the overall image content.

The aim of this research is to investigate the use of the gray level co-occurrence matrix technique as an absolute image quality metric. The underlying hypothesis is that image quality can be determined by a comparative process in which a sequence of images is compared to each other to determine the point of diminishing returns. An attempt is made to study whether the curve of image textural features versus image memory sizes can be used to decide the optimal image size. The approach used digitized images that were stored at several levels of compression. GLCM proves to be a good discriminator in studying different images however no such claim can be made for image quality. Hence the search for the best image quality metric continues.

To Vishal, Dr. Clarke, Brian Goldiez

ACKNOWLEDGMENTS

I would like to acknowledge and thank my advisor, Dr. Thomas L. Clarke, for his continuous guidance and patience throughout my thesis studies. His support and understanding is greatly appreciated. I would like to thank Brian Goldiez for his technical contributions to this work, and for the additional help that he willingly provided. I would also like to express my gratitude towards Dr. Kincaid who has been a great person to work with.

Finally, I would like to thank my husband for his tremendous love, support and encouragement while bringing this thesis to a final conclusion.

TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES vii

CHAPTER 1 - INTRODUCTION 1

1.1 Objective quality measurement 1
1.2 Subjective quality measurement 2
1.3 Perceptual quality measurement 3

CHAPTER 2 - GLCM BACKGROUND 8

2.1 Choice of radius δ 10
2.2 Choice of angle θ 10
2.3 Choice of quantized gray levels (G) 11

CHAPTER 3 - VARIOUS APPLICATIONS OF GLCM 17

3.1 Texture Analysis of SAR Sea Ice Imagery 17
3.2 Synthesis of Textures 18
3.3 Texture Defect Detection 19
3.4 Circular GLCM 20
3.5 Object Recognition and Matching 21
3.6 Image Segmentation and Edge Detection 22
3.7 Color texture classification by integrative co-occurrence matrices 23

CHAPTER 4 EXPERIMENTATION 24

CHAPTER 5 OPTIMIZATION TECHNIQUES 80

5.1 Gray level quantization 80
5.2 Windowing Technique 81
5.3 Gray level Co-occurrence link list (GLCLL) 81
5.4 Gray level co-occurrence hybrid structure (GLCHS) 82

CONCLUSION AND FUTURE WORK 84

APPENDIX: C/OPENGL CODE 85

REFERENCES 110

LIST OF FIGURES

Figure1a-b. Test image, general form of GLCM ... 9
Figure 2a-b. GLCM for δ=1 and θ=0°, 45°, 90°, 135° ... 9
Figure 3. Texture defect detection example ... 19
Figure 4. Circular GLCM example ... 20
Figure 5a-e. Carpet image Quality level 1-5 ... 26
Figure 6a-e. Plant image Quality level 1-5 ... 27
Figure 7a-e. Water image Quality level 1-5 ... 28
Figure 8a-e. Student Union image Quality level 1-5 ... 29
Figure 9a-e. Grass image Quality level 1-5 ... 30
Figure 10a-e. Flowers image Quality level 1-5 ... 31
Figure 11a-b. Dataset 1 Energy ... 32
Figure 12a-b. Datset 1 Contrast ... 33
Figure 13a-b. Datset 1 Entropy ... 34
Figure 14a-b. Dataset 1 Homogeneity ... 35
Figure 15a-d. Dataset 2 images ... 37
Figure 16. Dataset 2 Contrast ... 37
Figure 17. Dataset 2 Entropy ... 38
Figure 18. Dataset 2 Homogeneity ... 39
Figure 19. Dataset 2 Energy ... 39
Figure 20a-c. FACE image quality level 1, 3, 5 ... 40
Figure 21a-h. FACE image Energy ... 43
Figure 22a-h. FACE image Contrast ... 47
Figure 23a-h. FACE image Entropy ... 52
Figure 24a-h. FACE image Homogeneity ... 56
Figure 25a-e. CERTIFICATE image quality level 1-5 ... 60
Figure 26a-h. CERTIFICATE image Energy ... 63
Figure 27a-h. CERTIFICATE image Contrast ... 67
Figure 28a-h. CERTIFICATE image Entropy ... 71
Figure 29a-h. CERTIFICATE image Homogeneity ... 75

LIST OF TABLES

Table 1. Dataset 1 25

Table 2. Dataset 3 FACE image 42

Table 3. Dataset 3 CERTIFICATE image 63

CHAPTER 1 - INTRODUCTION

Images play a crucial role in today's age of succinct information. The field of image processing has exhibited enormous progress over past few decades. Generally, the images dealt in virtual environments or entertainment applications possess high fidelity resulting in large storage requirements. Images may undergo distortions during preliminary acquisition process, compression, restoration, communication or final display. Hence image quality measurement plays a significant role in several image-processing applications. Image quality, for scientific and medical purposes, can be defined in terms of how well desired information can be extracted from the image. An image is said to have acceptable quality if it shows satisfactory usefulness, which means discriminability of image content, and satisfactory naturalness, which means identifiability of image content. Digital storage of images has created an important place in imaging. Image quality metrics are important performance variables for digital imaging systems and are used to measure the visual quality of compressed images. The three major types of quality measurements are [Joon01]:

1.1 Objective quality measurement

The objective image quality measurement seeks to measure the quality of images algorithmically. A good objective measure reflects the distortion on image due to blurring, noise, compression and sensor inadequacy. Objective analysis involves use of image quality/distortion metrics to automatically perceive image quality; the most widely being used are "Peak Signal-to-noise Ratio PSNR" and "Mean Squared Error MSE".

These methods provide mathematical deviations between original and processed images. The analysis depends on the number of images used in the measurement and the nature or type of measurement using the pixel elements of digitized images. Metrics have been defined either in spatial or frequency domain. These measurement techniques are easy to calculate, however they do not consider human visual sensitivities. They do not adequately predict distortion visibility and visual quality for images with large luminance variations or with varying content. It is believed that a combination of numerical and graphical measures may prove useful in judging image quality.

1.2 Subjective quality measurement

For several years, the image quality assessment (QA) has been performed subjectively using human observers based on their satisfaction. It depends on the type, size, range of images, observer's background and motivation and experimental conditions like lighting, display quality etc. The human visual system (HVS) is enormously complex with optical, synaptic, photochemical and electrical phenomena. The International Telecommunication Union (ITU) has recommended a 5-point scale using the adjectives bad, poor, fair, good and excellent. A numerical category scaling also can be used as an alternative, which is linear and hence more convenient to use. Subjective quality measurement techniques provide numerical values that quantify viewer's satisfaction, however are time-consuming and observer responses may vary. They provide no constructive methods for performance improvement and are difficult to use as a part of design process.

1.3 Perceptual quality measurement

The perceptual quality measurement techniques are based on models of human visual perception like image discrimination models and task performance based models. Ideally, they should be able to characterize spatial variations in quality across an image.

Image quality measures (IQM) are figures of merit used for the evaluation of imaging systems or coding/processing techniques. Image quality measurement is still an unsolved problem today. There are at least two factors, which contribute to difficulty in finding a complete algorithm for image quality measurement. First factor being that there are many different kinds of noises and each can affect the quality of image differently. Secondly it is not simple to mathematically prove the quality of an image without human judgment. Image QA algorithms can as well be classified as “Full-reference” or bivariant, in which the algorithm has access to the perfect image, “No-reference” or univariant, in which the algorithm has access only to the distorted image and “Reduced-reference”, in which the algorithm has partial information regarding the perfect image. All algorithms try to map the reconstructed image to some quantity that is positive and zero only when original and modified images are identical and also increases monotonically as the modified image looks worse. It is very useful to be able to automatically assess the quality of images when the number of images to be evaluated is large. Daly’s visual difference predictor (VDP) is a popular bivariant tool to assess image quality [Daly93]. It computes a map of visible differences between a degraded image and the reference. The Karunasekera and Kingsbury (KK) model gives a quality mark to the degraded image compared to the reference for the assessment of visual quality of the JPEG compressed image [Karu96]. The KK model gives an image quality assessment that corresponds to the visibility of the blocking artifacts. The visibility is evaluated by the reaction time required by an observer to identify the degraded image. The results match those of the human visual system. Some other quality criteria simply compute a distance between the degraded image and the reference. When two images are compared in terms of quality, one desires a measure that parallels human visual system with the expectations that the differences in quality judged by human eye to be large are also mathematically large and if the differences are insignificant to human eye, the error size should be small. Dung and others [Dung98] proposed the Image Quality Measure Error (IQME), which supports measuring image quality locally as well as globally. It takes into consideration the effect of the change of each pixel value to the local area, which contains that pixel and the effect of that area to the whole image. The number of pixels changed is also taken into consideration since the overall quality of the image depends on it as well. IQMs based on pixel-difference, correlation, spectral, context and human visual system are studied comprehensively using analysis of variance.

Texture happens to be an important characteristic for the automated or semi automated interpretation of digital images. Texture analysis has history of more than three decades. During the 1970s and early 1980s, the algorithms have been mainly based on first and second order statistics of the image pixel gray level values as spatial domain gray level co-occurrence matrix (SDCM) and neighboring gray level dependence matrix (NGLDM). In the mid 1980s, model based methods such as Markov Random Fields (MRF) and simultaneous autoregressive (SAR) appeared. Wavelets gained importance in late 1980s. Texture analysis is a crucial problem since it conditions the quality of segmentation and interpretation in lots of applications such as in the textile industry or for satellite imaging. Texture analysis has applications in image segmentation and classification, biomedical image analysis, and automatic detection of surface defects. When describing the content of a region, textures give a better understanding about the region as compared to intensity descriptors such as average gray-level, minimum and maximum gray level. The three common ways of analyzing textures are statistical, structural and spectral approaches. Each of these approaches is explained below [Gonz02].

- Structural approach:

This approach defines a grammar for the way that the pattern of the texture produces structure. The texture is parsed to see if it matches the grammar. The parse tree for a pattern in a particular region is used as a descriptor.

- Spectral approach:

If textures are periodic patterns, another useful way to analyze them is the use of the frequency domain. The entire frequency domain has as much information as the image itself. The frequency domain contains information from all parts of the image and is useful for global texture analysis. Information can be condensed either by collapsing a particular frequency across all orientations or by collapsing all frequencies in a particular direction. This technique referred to as collapsed frequency domains gives two one-dimensional descriptors useful for discriminating textures. Local frequency content can be defined by using some form of co-joint spatial frequency representation. One of the ways is to examine the N x N neighborhood around a point and compute its Fourier transform. Moving from one textured region to other region changes the frequency content of the window. Differences in the frequency content of each window are used as a means of segmentation. They are normally used to discriminate between different regions or to classify them, to produce descriptions so that we can reproduce textures and to segment an image based on textures. It is difficult to know which attributes might be used for an application because the efficiency of each type of attribute is often badly known. A study [Rose01] showed that classical attributes are relatively complementary and provide good recognition rate if they are combined.

- Statistical approach:

Textures are generally random however possess consistent properties. Hence an obvious way to describe texture is their statistical properties. Various moments based on the gray level histogram computed from a digital image can be used to describe statistical properties such as mean, variance, skewness and flatness. The first four moments are easier to describe intuitively, beyond which the description becomes harder. The first moment is the mean intensity; the second central moment is the variance describing how similar the intensities are within the region. The third central moment, skew, describes how symmetric the intensity distribution is about the mean and the fourth central moment, kurtosis, describes how flat the distribution is. Given a sufficient number of moments, it is possible to reconstruct an image. The histogram-based measurements suffer from the limitation that they carry no information regarding the relative spatial position of pixels with one another. The spatial dependence relationship can be incorporated by considering the distribution of intensities as well as the position of pixels with equal or nearly equal intensity values. The technique involves statistically sampling the way certain gray levels occur in relation to other gray levels. This method obtains the gray level co-occurrence matrix (GLCM) of specified texture, which further gives various descriptors by measuring texture properties. Gray level co-occurrence matrices are widely used to discriminate texture images and are studied in greater depth in the next chapter.

The research is presented in four main chapters: Chapter 2 provides the fundamental background of GLCM technique and the traditional applications of GLCM have been described in chapter 3. The experimental methodology has been discussed in chapter 4 whereas chapter 5 briefs the optimization techniques that can be incorporated in GLCM processing.

CHAPTER 2 - GLCM BACKGROUND

Texture is one of the important characteristics used in identifying objects or regions of interest in an image. Texture contains important information about the structural arrangement of surfaces. The textural features based on gray-tone spatial dependencies have a general applicability in image classification. The three fundamental pattern elements used in human interpretation of images are spectral, textural and contextual features. Spectral features describe the average tonal variations in various bands of the visible and/or infrared portion of an electromagnetic spectrum. Textural features contain information about the spatial distribution of tonal variations within a band. The fourteen textural features proposed by Haralick et all [Hara73] contain information about image texture characteristics such as homogeneity, gray-tone linear dependencies, contrast, number and nature of boundaries present and the complexity of the image. Contextual features contain information derived from blocks of pictorial data surrounding the area being analyzed.

Haralick et all first introduced the use of co-occurrence probabilities using GLCM for extracting various texture features. GLCM is also called as Gray level Dependency Matrix. It is defined as "A two dimensional histogram of gray levels for a pair of pixels, which are separated by a fixed spatial relationship." GLCM of an image is computed using a displacement vector *d,* defined by its radius δ and orientation θ. Consider a 4x4 image represented by figure 1a with four gray-tone values 0 through 3. A generalized GLCM for that image is shown in figure 1b where #(i,j) stands for number of times gray tones *i* and *j* have been neighbors satisfying the condition stated by displacement vector d.

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

Figure1a. Test image

Gray tone	0	1	2	3
0	#(0,0)	#(0,1)	#(0,2)	#(0,3)
1	#(1,0)	#(1,1)	#(1,2)	#(1,3)
2	#(2,0)	#(2,1)	#(2,2)	#(2,3)
3	#(3,0)	#(3,1)	#(3,2)	#(3,3)

Figure 1b. General form of GLCM

The four GLCM for angles equal to 0°, 45°, 90° and 135° and radius equal to 1 are shown in figure 2 a-d.

4	2	1	0
2	4	0	0
1	0	6	1
0	0	1	2

Figure 2a. GLCM for δ=1 and θ=0°

6	0	2	0
0	4	2	0
2	2	2	2
0	0	2	0

Figure 2c. GLCM for δ=1 and θ=90°

4	1	0	0
1	2	2	0
0	2	4	1
0	0	1	0

Figure 2b. GLCM for δ=1 and θ=45°

2	1	3	0
1	2	1	0
3	1	0	2
0	0	2	0

Figure 2d. GLCM for δ=1 and θ=135°

These are symmetric matrices hence evaluation of either upper or lower triangle serves the purpose. Frequency normalization can be employed by dividing value in each cell by the total number of pixel pairs possible. Hence the normalization factor for 0° would be $(N_x\text{-}1) \times N_y$ where N_x represents the width and N_y represents the height of the image. The quantization level is an equally important consideration for determining the co-occurrence texture features. Also, neighboring co-occurrence matrix elements are highly correlated as they are measures of similar image qualities. Each of these factors is discussed ahead in detail.

2.1 Choice of radius δ

Various research studies show δ values ranging from 1, 2 to 10. Applying large displacement value to a fine texture would yield a GLCM that does not capture detailed textural information. From the previous studies, it has been concluded that overall classification accuracies with $\delta = 1, 2, 4, 8$ are acceptable with the best results for $\delta = 1$ and 2. This conclusion is justified, as a pixel is more likely to be correlated to other closely located pixel than the one located far away. Also, displacement value equal to the size of the texture element improves classification.

2.2 Choice of angle θ

Every pixel has eight neighboring pixels allowing eight choices for θ, which are 0°, 45°, 90°, 135°, 180°, 225°, 270° or 315°. However, taking into consideration the definition of GLCM, the co-occurring pairs obtained by choosing θ equal to 0° would be similar to those obtained by choosing θ equal to 180°. This concept extends to 45°, 90°

and 135° as well. Hence, one has four choices to select the value of θ. Sometimes, when the image is isotropic, or directional information is not required, one can obtain isotropic GLCM by integration over all angles.

2.3 Choice of quantized gray levels (G)

The dimension of a GLCM is determined by the maximum gray value of the pixel. Number of gray levels is an important factor in GLCM computation. More levels would mean more accurate extracted textural information, with increased computational costs. The computational complexity of GLCM method is highly sensitive to the number of gray levels and is proportional to $O(G^2)$ [Clau02].

Thus for a predetermined value of G, a GLCM is required for each unique pair of δ and θ. GLCM is a second-order texture measure. The GLCM's lower left triangular matrix is always a reflection of the upper right triangular matrix and the diagonal always contains even numbers. Various GLCM parameters are related to specific first-order statistical concepts. For instance, contrast would mean pixel pair repetition rate, variance would mean spatial frequency detection etc. Association of a textural meaning to each of these parameters is very critical. Traditionally, GLCM is dimensioned to the number of gray levels G and stores the co-occurrence probabilities g_{ij}. To determine the texture features, selected statistics are applied to each GLCM by iterating through the entire matrix. The textural features are based on statistics which summarize the relative frequency distribution which describes how often one gray tone will appear in a specified spatial relationship to another gray tone on the image.

Following notations are used to explain the various textural features:

g_{ij} = $(i, j)^{th}$ entry in GLCM

$g_x(i)$ = i^{th} entry in marginal probability matrix obtained by summing rows of

$$g_{ij} = \sum_{j=1}^{N_g} g(i, j)$$

N_g = Number of distinct gray levels in the image

$$\sum_i = \sum_{i=1}^{N_g}$$

$$\sum_j = \sum_{j=1}^{N_g}$$

$$g_y(i) = \sum_{i=1}^{N_g} g(i, j)$$

$$g_{x+y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} g(i, j) \text{ where } i+j = k = 2, 3, \ldots, 2\,N_g$$

$$g_{x-y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} g(i, j) \text{ where } |i-j| = k = 0, 1, \ldots, N_g-1$$

Few of the common statistics applied to co-occurrence probabilities are discussed ahead.

1) Energy:

$$\text{Energy (ene)} = \sum_i \sum_j g_{ij}^{\ 2}$$

This statistic is also called Uniformity or Angular second moment. It measures the textural uniformity that is pixel pair repetitions. It detects disorders in textures. Energy reaches a maximum value equal to one. High energy values occur when the gray level distribution has a constant or periodic form. Energy has a normalized range. The GLCM of less homogeneous image will have large number of small entries.

2) Entropy:

$$\text{Entropy (ent)} = -\sum_i \sum_j g_{ij} \log_2 g_{ij}$$

This statistic measures the disorder or complexity of an image. The entropy is large when the image is not texturally uniform and many GLCM elements have very small values. Complex textures tend to have high entropy. Entropy is strongly, but inversely correlated to energy.

3) Contrast:

$$\text{Contrast (con)} = \sum_i \sum_j (i-j)^2 g_{ij}$$

This statistic measures the spatial frequency of an image and is difference moment of GLCM. It is the difference between the highest and the lowest values of a contiguous set of pixels. It measures the amount of local variations present in the image. A low contrast image presents GLCM concentration term around the principal diagonal and features low spatial frequencies.

4) Variance:

$$\text{Variance (var)} = \sum_i \sum_j (i-\mu)^2 g_{ij} \text{ where } \mu \text{ is the mean of } g_{ij}$$

This statistic is a measure of heterogeneity and is strongly correlated to first order statistical variable such as standard deviation. Variance increases when the gray level values differ from their mean.

5) Homogeneity:

$$\text{Homogeneity (hom)} = \sum_i \sum_j \frac{1}{1+(i-j)^2} g_{ij}$$

This statistic is also called as Inverse Difference Moment. It measures image homogeneity as it assumes larger values for smaller gray tone differences in pair elements. It is more sensitive to the presence of near diagonal elements in the GLCM. It has maximum value when all elements in the image are same. GLCM contrast and homogeneity are strongly, but inversely, correlated in terms of equivalent distribution in the pixel pairs population. It means homogeneity decreases if contrast increases while energy is kept constant.

6) Correlation:

$$\text{Correlation (cor)} = \frac{\sum_i \sum_j (ij) g_{ij} - \mu_x \mu_y}{\sigma_x \sigma_y}$$ where μ_x, μ_y, σ_x and σ_y are the means and standard deviations of g_x and g_y

The correlation feature is a measure of gray tone linear dependencies in the image.

The rest of the textural features are secondary and derived from those listed above.

7) Sum Average:

$$\text{Sum Average (sa)} = \sum_{i=2}^{2N_g} i g_{x+y}(i)$$

8) Sum Entropy:

$$\text{Sum Entropy (se)} = - \sum_{i=2}^{2N_g} g_{x+y}(i) \log\{g_{x+y}(i)\}$$

9) Sum Variance:

$$\text{Sum Variance (sv)} = \sum_{i=2}^{2N_g} (i - sa)^2 \, g_{x+y}(i)$$

10) Difference Variance:

Difference Variance = variance of g_{x-y}

11) Difference Entropy:

$$\text{Difference Entropy} = - \sum_{i=0}^{N_g-1} g_{x-y}(i) \log\{g_{x-y}(i)\}$$

12) Maximum Correlation Coefficient:

Maximum Correlation Coefficient (MCC) = (second largest eigen value of Q $)^{0.5}$

$$\text{Where Q(I,j)} = \sum_k \frac{g(i,k)g(j,k)}{g_x(i)g_y(k)}$$

13) , 14) Information Measures of Correlation:

$$\text{Information measure of correlation 1 (IMC1)} = \frac{HXY - HXY1}{\max\{HX, HY\}}$$

$$\text{Information measure of correlation 2 (IMC2)} = \sqrt{(1 - \exp[-2.0(HXY2 - HXY)]}$$

$HXY = -\sum_i \sum_j g_{ij} \log_2 g_{ij}$ where HX and HY are entropies of g_x and g_y

$$HXY1 = -\sum_i \sum_j g_{ij} \log_2\{g_x(i)g_y(j)\}$$

$$HXY2 = -\sum_i \sum_j g_x(i)g_y(j) \log_2\{g_x(i)g_y(j)\}$$

The question what exactly the textural features represent from a human perception point of view can be a subject for a thorough experimentation. Of the textural features described above, the angular second moment, the entropy, the sum entropy, the difference entropy, the information measure of correlation and the maximal correlation features have the invariance property. Earlier studies [Wang02] cite “Energy” and “Contrast” to be the most efficient parameters for discriminating different textural patterns. The general thumb rules used in the selection of the textural features can be stated as follows:

- Energy is preferred to entropy as its values belong to normalized range.
- Contrast is associated with the average gray level difference between neighbor pixels. It is similar to variance however preferred due to reduced computational load and its effectiveness as a spatial frequency measure.
- Energy and contrast are the most significant parameters in terms of visual assessment and computational load to discriminate between different textural patterns.

CHAPTER 3 - VARIOUS APPLICATIONS OF GLCM

GLCM has been used extensively in the field of image processing. It has been applied from a range of applications like texture analysis to synthesis including gray scale as well as color texture recognition. A few of its popular applications are discussed ahead.

3.1 Texture Analysis of SAR Sea Ice Imagery

This application uses GLCM to quantitatively evaluate textural parameters and determine which parameter values are best for mapping sea ice textures. The importance of gray-level quantization, displacement and orientation factors for representing sea ice in synthetic aperture radar (SAR) imagery is studied [Soh99]. The theory is based on a computationally efficient expression $\chi^2 = N \{\sum_i \sum_j (g_{ij}^{\;2} / r_i c_j) - 1\}$ where $r_i = \sum_{j=1}^{N} g(i,j)$ and $c_j = \sum_{i=1}^{N} g(i,j)$ and $N = \sum_{i=1}^{Ng} \sum_{j=1}^{Ng} g(i,j)$. Here *g(i, j)* represents the GLCM and N_g represents the total number of gray values. Finally the matrix yielding highest value of χ^2 is supposed to be optimal. Three types of co-occurrence matrices were studied as follows:

- Mean Displacement Mean Orientation (MDMO): It assumes that every matrix of specific displacement and orientation is partially and cumulatively representative for the sample. Feature measures of matrices of the four orientations of 0°, 45°, 90° and 135° are averaged and then further averaged over displacement range.

- Optimal Displacement Mean Orientation (ODMO): It assumes that only the matrix whose χ^2 value is the highest with a specific displacement value is truly and sufficiently representative for the sample.
- Optimal Displacement Optimal Orientation (ODOO): It assumes that only the matrix whose χ^2 value is the highest with a specific displacement and orientation is truly and sufficiently representative for the sample.

Few of the important conclusions drawn from the experiments were that MDMO implementation is better, ODMO and ODOO have almost same performances indicating insignificance of orientation, range of displacement values is more representative than single value, and 64 gray level representation is efficient and sufficient for analysis of SAR images.

3.2 Synthesis of Textures

An algorithm for generating synthetic textures based on GLCM is presented, [Lohm95] which is used to imitate real textures taken from satellite images. A histogram is computed from the desired GLCM. Then an initial image that has the desired histogram is randomly generated. Further, a chain of images is iteratively produced such that the new image is improvement over the initial in terms of error of distance of current co-occurrences from desired co-occurrences. The iteration stops when the error goes below a pre-specified threshold value. The algorithm converges only if a solution exists. The difference between the real and synthetic textures is indistinguishable by the human eye, which implies that co-occurrence features are well suited for characterizing these types of images.

3.3 Texture Defect Detection

A combination of wavelet theory and co-occurrence matrices [Lati00] is used to detect defects in textile images. Texture defect detection can be defined as the process of determining the location and/or extent of collection of pixels in a textured image with remarkable deviation in their intensity values or spatial arrangement with respect to the background texture. The algorithm comprises of four main steps, which are decomposition of the gray level image into sub-bands, partitioning the textured image into non-overlapping sub-windows, extracting co-occurrence features and finally classifying each sub-window as defective or non-defective. Wavelet filter coefficients are used to obtain images I_{LL}, I_{LH}, I_{HL} and I_{HH} where L and H represent low-pass and high-pass bands respectively.

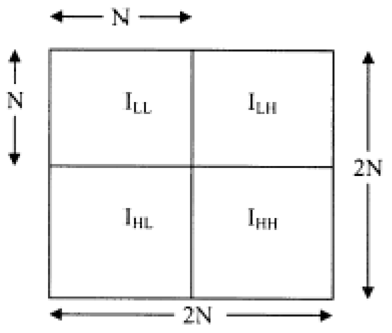


Figure 3. Decomposition of image I of size 2N x 2N

The study concludes that focusing on a particular band with high discriminatory power improves the detection performance and increases computational efficiency as well.

3.4 Circular GLCM

GLCM can be used to study the short wavelength anomalies in the Earth's gravitational field [Coop04]. In this case, the GLCM textural measures use a vector that connects pairs of pixels within a kernel that is moved over the image. This is a unique method as it deals with circular features to enhance the elusive details. The vectors connect points that lie on the perimeter of circles of different radii. A mid-point algorithm is used to select the points that lay on each circle. The circle's radius within the kernel ranges from one to a user-specified maximum size. A rose diagram is used to show the directions of the vectors in the kernel. This unique kernel is useful for the analysis of anisotropic textures. Inverse difference moment has been specifically used which yields a strong response at the central locations of the features of interest.

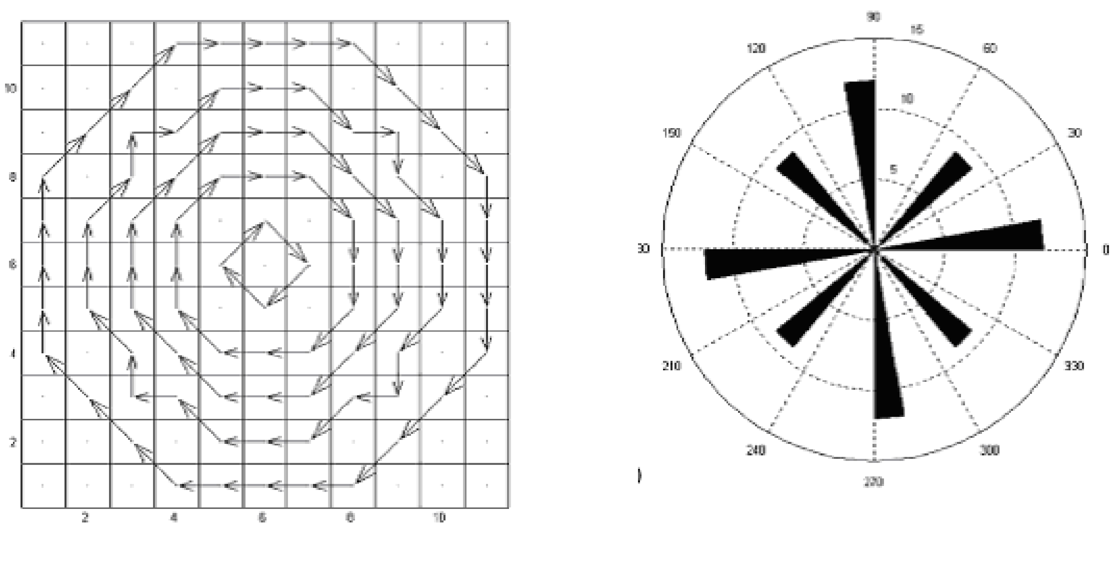


Figure 4. Circular GLCM vectors and corresponding Rose diagram

Thus the use of GLCM vectors that follow circular contours that occur in gravity data due to its inherently monopolar nature helps detecting circular features as well as enhance linear features that lie at any orientation. Moreover this method is inexpensive as compared to other existing geophysical methods.

3.5 Object Recognition and Matching

This application discusses a novel method based on quantitative estimation of relations between some elementary image structures, which are represented by elements of special multidimensional co-occurrence matrices (MDCM) [Kova96]. An image of any object can be considered as a composition of elementary structures, the elements of which carry some attributes (e.g. gray level value, gradient magnitude, orientation) and have some relations (e.g. gray level difference, relative gradient orientation). An M-dimensional co-occurrence matrix is used where each of the attributes and relations correspond to different axis of the matrix. Object is made recognizable due to the balanced presence of some specific elementary structures in it. A MDCM is an M-dimensional array, the elements of which have the general form of (a_1, a_2, …a_{M1}; b_1, b_2, … b_{M2}) where M1 + M2 = M and a_i takes all possible values a certain attribute could take for an elementary structure, while b_j takes all possible values a certain relation could take. GLCM is used as a powerful way of representing the properties of the elementary geometric structures. Such features, when identified, lead to linearly separable classes and then a simple classifier identifies a certain object. This approach, as compared to clustering method, uses ratio of matrix/histogram elements, eliminating the need to specify a metric and the measuring units. The success of the method depends on the correct choice of the attributes and relations and the availability of sufficient number of examples.

3.6 Image Segmentation and Edge Detection

The field of image analysis has been researched for several years. It involves the process of extracting information from an image and analyzing it to achieve a specific goal. The two main steps involved are:

- Image segmentation: segmentation of an image into homogeneous regions with respect to certain image characteristic, example – regions of uniform grey level.
- Edge detection: the extraction of the locations in the image having changes in intensity.

The differences between the key image features are emphasized using adaptive transforms [Hadd93]. The Sobel edge operator E is a square matrix with 3x3 dimensions and elements [1, 0, -1; 2, 0, -2; 1, 0, -1]. Convolving the Sobel edge operator E with the image in the neighborhood of a pixel yields good results as vertical edge strength image. It is referred to as edge co-occurrence matrix. The parameters of the peak along the leading diagonal are determined using correlation techniques. The matrix is further labeled in a way to reflect the content of the image. Labeling of the matrix requires detailed knowledge about the distributions in the matrix. The labeled matrix is used as a look-up table for simultaneously segmenting the regions of an image and for detecting the prominent edges for a particular edge operator. Using results of several transforms, it is possible to detect edges of all orientations.

3.7 Color texture classification by integrative co-occurrence matrices

Color is an important issue in digital image processing. It is a vectorial feature assigned to each pixel. Color information improves the results of gray scale texture features. Two categories of co-occurrence matrices (CMs) are proposed for color texture classification [Palm03]:

- Single channel co-occurrence matrices (SCMs): They consist of gray scale CMs successively applied to separated color channels.
- Multi channel co-occurrence matrices (MCMs): These capture correlation between textures of different color channels. They provide the opportunity to study the effect of texture and pure color analysis in one unified framework. No spatial adjacency but channel adjacency is regarded.

The studies show that several experiments were conducted on various specific databases. This is a novel approach in the field of color texture recognition.

CHAPTER 4 EXPERIMENTATION

Earlier literature shows that GLCM textural features are used for category-identification of images representing different content [Hara73]. For instance the energy parameter can be used as a measure of homogeneity for comparison of two images, one representing grasslands and the other is representing water body. Since the water body image will have fewer gray tone transitions, its energy value is expected to be lower than that of the grasslands image. Similarly, the contrast feature, which represents local variation present in an image, would be higher for grasslands image as compared to the other image. The various kinds of datasets relevant for analysis may include photomicrographs, aerial photographs of natural or man-made scenes, high altitude satellite pictures.

The primary objectives of the study were as follows:

- Objective 1: To study if the textural features followed some specific trend as the quality of images increased.
- Objective 2: To study if the orientation of the overall image content can be used to speculate the most appropriate choice of GLCM angle

This study focuses on digital images representing content ranging from simple text, periodic patterns, natural scenes, plants to human faces. Each dataset was formed by storing an image at five quality levels using jpeg compression technique and maintaining constant pixel resolution. This study is in a way unique from the earlier GLCM research works because it analyses different compressed versions of the same image. Quality level

1 signifies the poorest quality while quality level 5 is the best. The Irfanview application was used for this purpose, which is freely downloadable [Irfa04].

The first dataset comprised of six different images stored at five compression levels. The following table gives the memory sizes of each image:

Image	Quality level 1	Quality level 2	Quality level 3	Quality level 4	Quality level 5
Carpet (Figure 5a-5e)	6KB	7KB	8KB	9KB	13KB
Plant (Figure 6a-6e)	5KB	6KB	7KB	8KB	11KB
Water (Figure 7a-7e)	5KB	7KB	7KB	8KB	12KB
Student Union (Figure 8a-8e)	6KB	7KB	8KB	9KB	12KB
Grass (Figure 9a-9e)	12KB	12KB	13KB	15KB	20KB
Flowers (Figure 10a-10e)	10KB	12KB	14KB	16KB	21KB

Table 1. Dataset 1

A Sony digital camera DSC-F717, set at pixel resolution of 640x480, was used to take the pictures. Using the Irfanview application, all pictures were resampled at 160x120 sizes for faster computation. The lanczos filter was used for resampling which offers better quality at the cost of higher processing time.

A) CARPET Image:

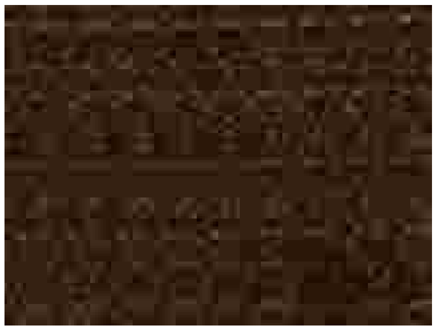


Figure 5a. Quality level 1

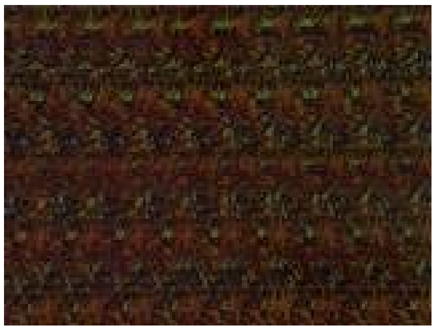


Figure 5d. Quality level 4

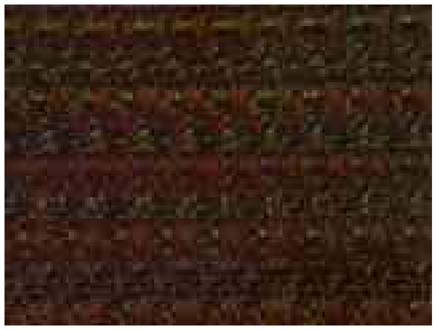


Figure 5b. Quality level 2

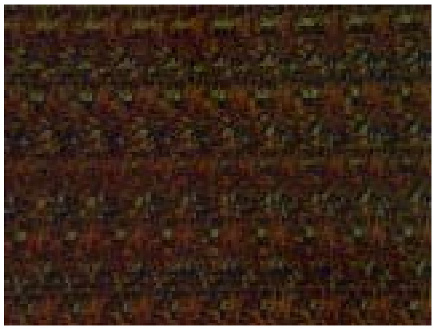


Figure 5e. Quality level 5

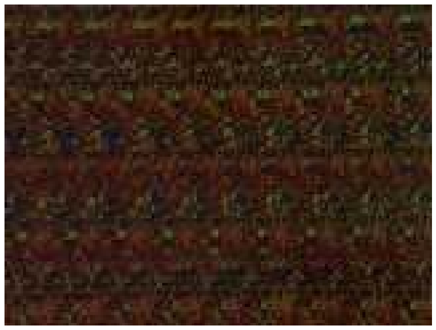


Figure 5c. Quality level 3

B) PLANT Image:

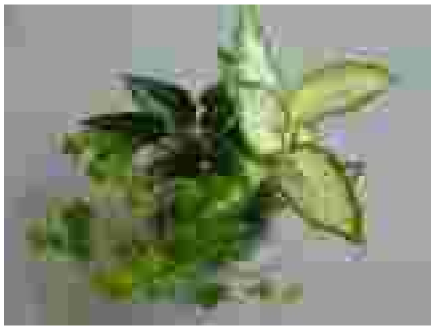


Figure 6a. Quality level 1

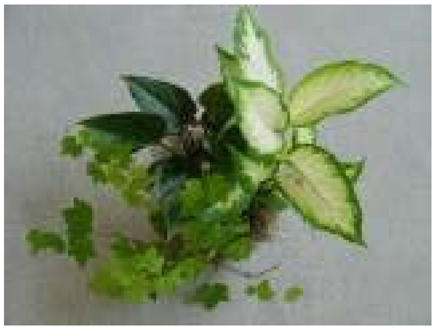


Figure 6d. Quality level 4

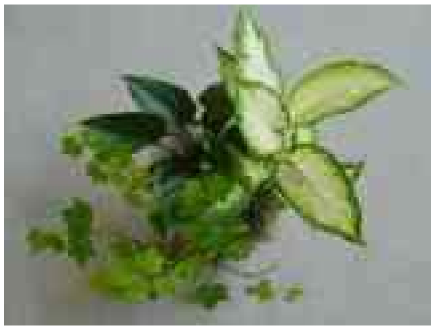


Figure 6b. Quality level 2

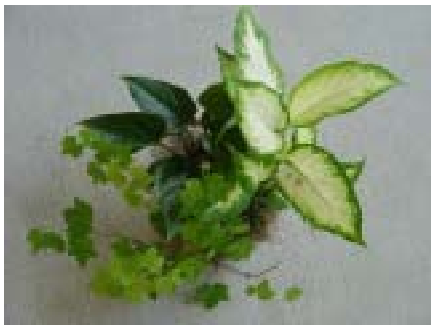


Figure 6e. Quality level 5

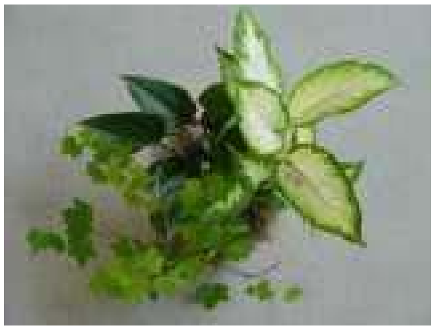


Figure 6c. Quality level 3

C) WATER Image:

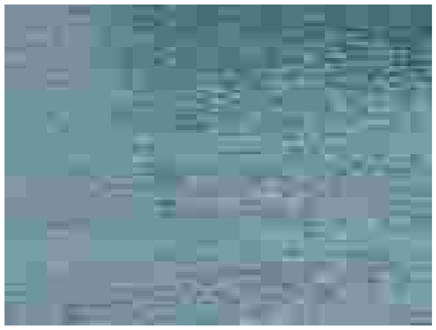


Figure 7a. Quality level 1

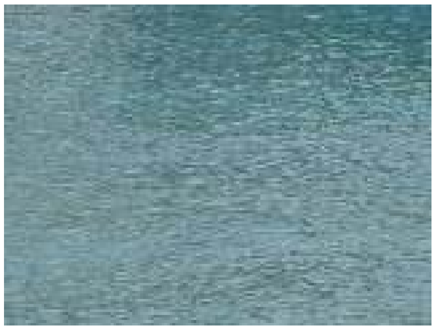


Figure 7d. Quality level 4

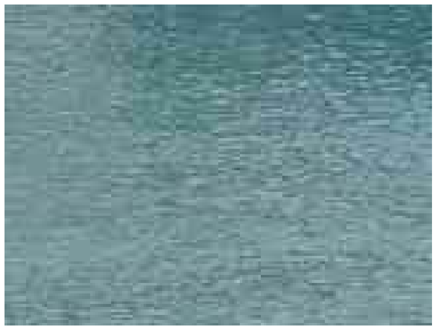


Figure 7b. Quality level 2

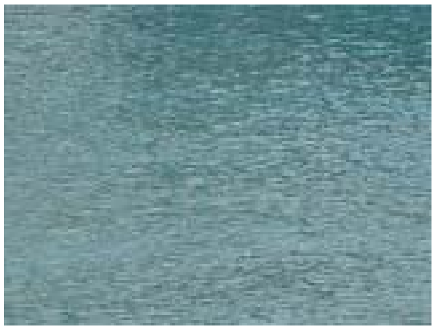


Figure 7e. Quality level 5

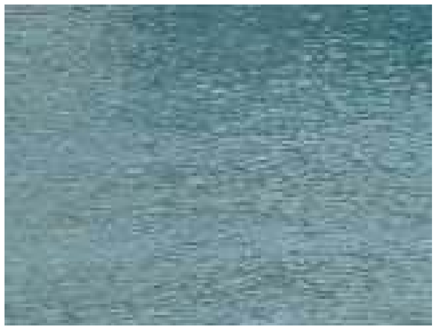


Figure 7c. Quality level 3

D) STUDENT UNION Image:

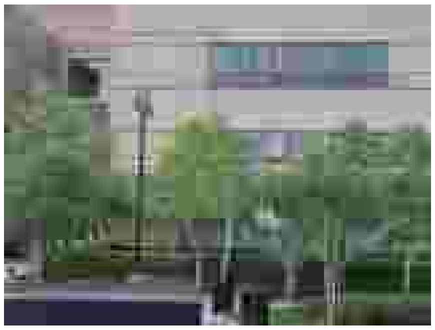


Figure 8a. Quality level 1

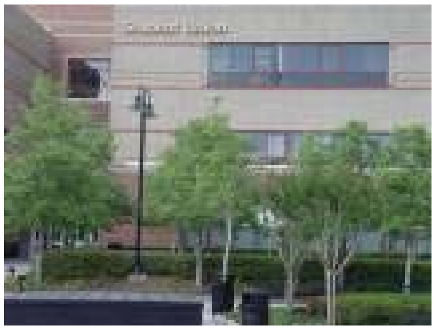


Figure 8d. Quality level 4

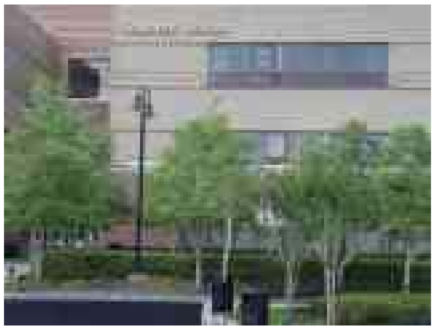


Figure 8b. Quality level 2

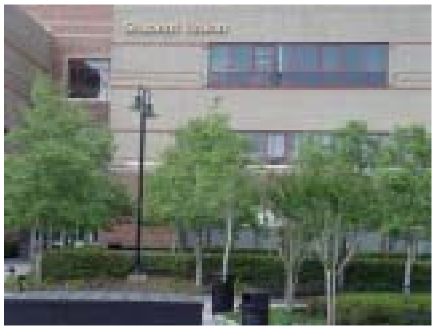


Figure 8e. Quality level 5

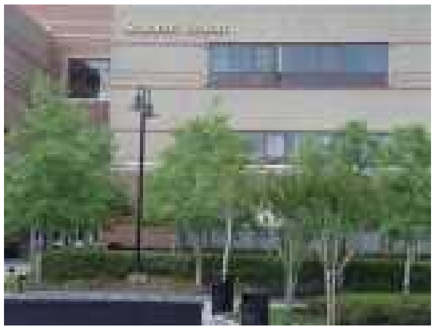


Figure 8c. Quality level 3

E) GRASS Image:

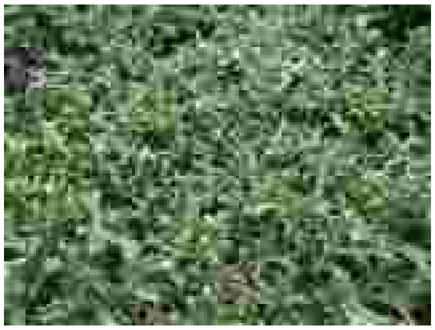


Figure 9a. Quality level 1

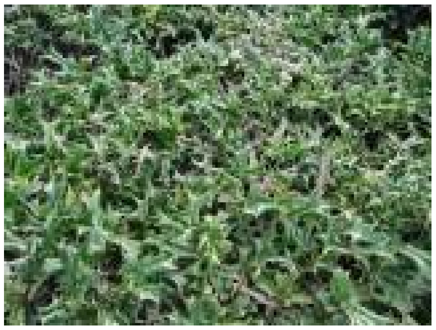


Figure 9d. Quality level 4

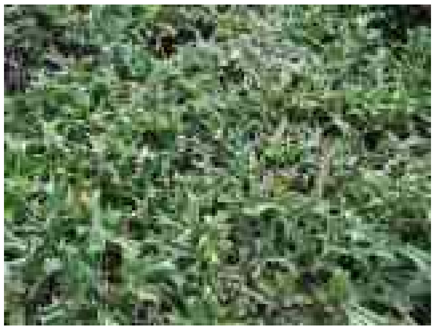


Figure 9b. Quality level 2

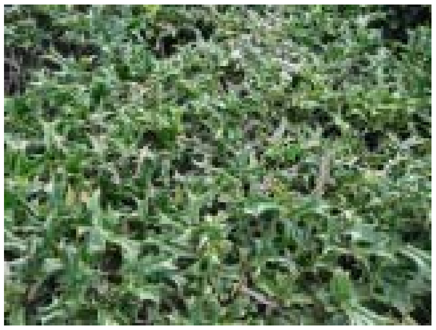


Figure 9e. Quality level 5

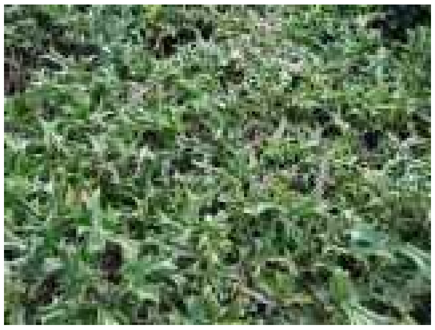


Figure 9c. Quality level 3

F) FLOWERS Image:

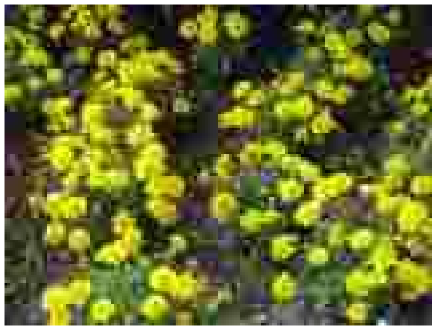


Figure 10a. Quality level 1

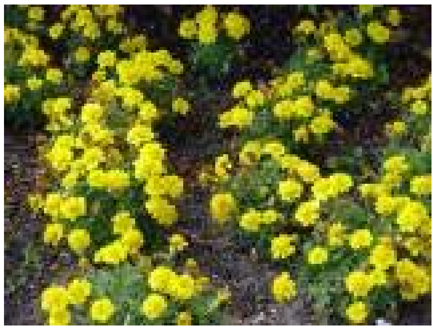


Figure 10d. Quality level 4

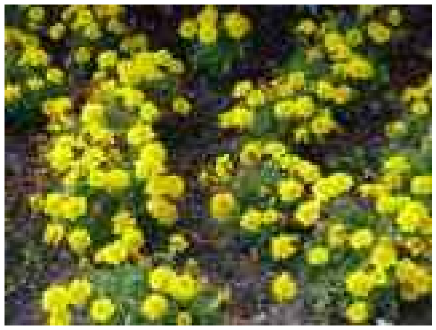


Figure 10b. Quality level 2

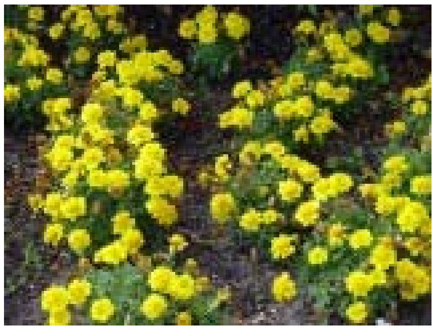


Figure 10e. Quality level 5

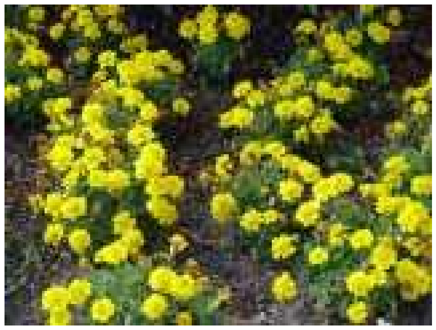
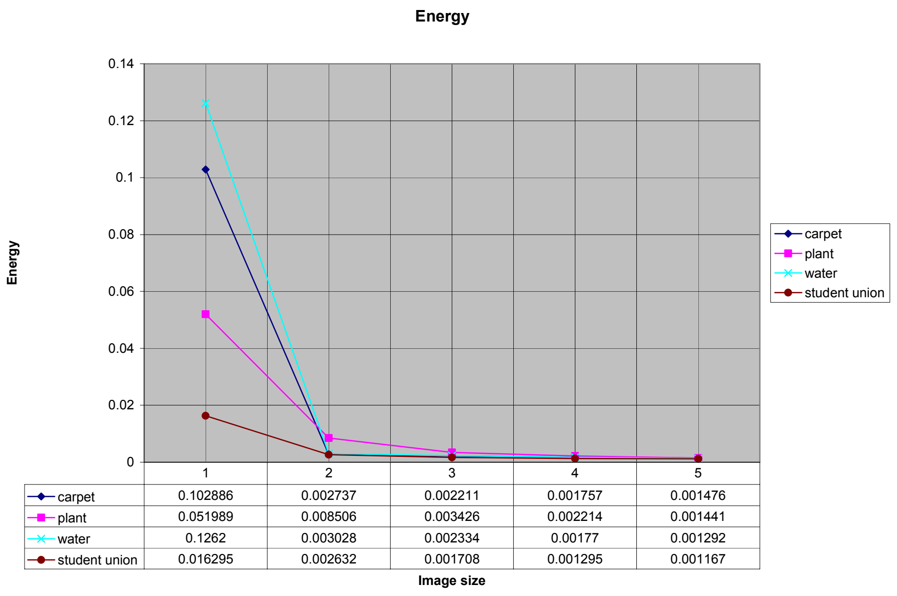


Figure 10c. Quality level 3

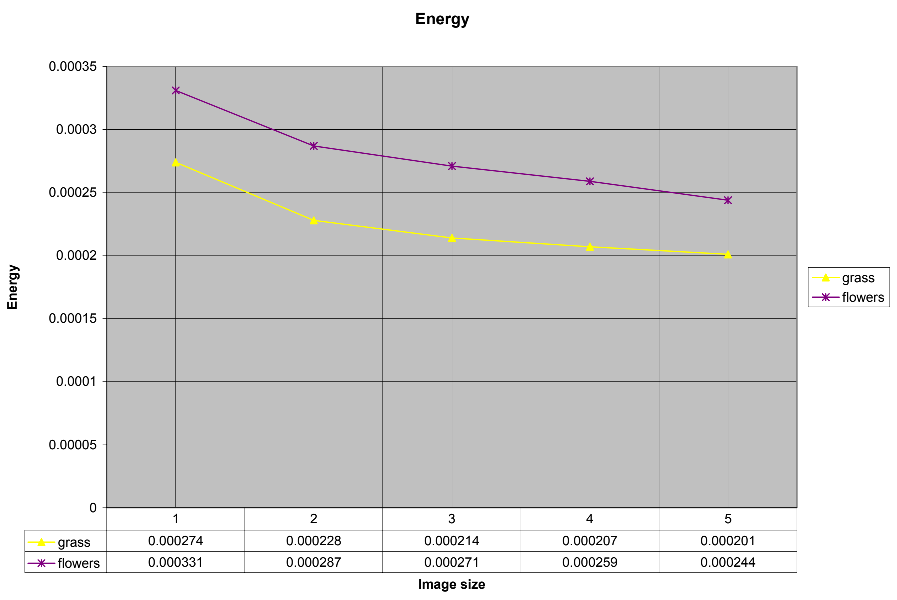
As discussed in the previous chapter, radius and angle happen to be the crucial parameters for GLCM processing. In this experimentation, the radius was set to 1 and angle was set to 0°, and textural parameters for all the thirty images were calculated. Each run took approximately 2 minutes of processing time.

Following plots were obtained for the six images. It is observed that the values of the textural parameters for the grass and flowers images lie in a different range as compared to the rest of the images. Hence each of the textural plots has been grouped into two for better readability.



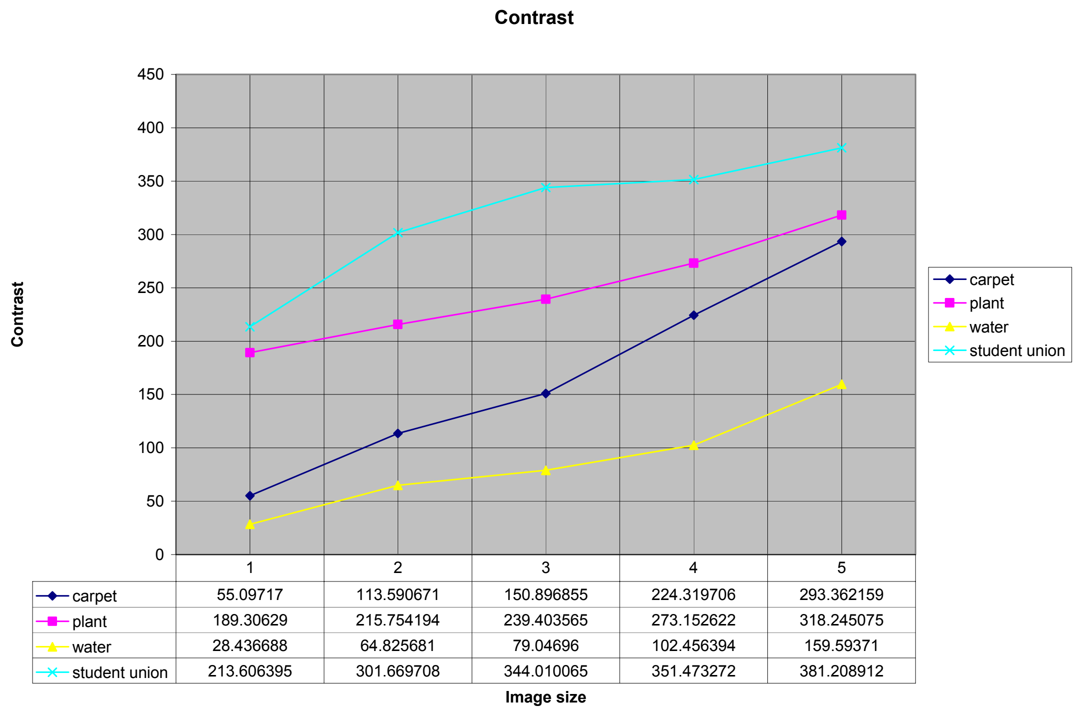
	1	2	3	4	5
carpet	0.102886	0.002737	0.002211	0.001757	0.001476
plant	0.051989	0.008506	0.003426	0.002214	0.001441
water	0.1262	0.003028	0.002334	0.00177	0.001292
student union	0.016295	0.002632	0.001708	0.001295	0.001167

Figure 11a. Energy (carpet, plant, water, student union images)



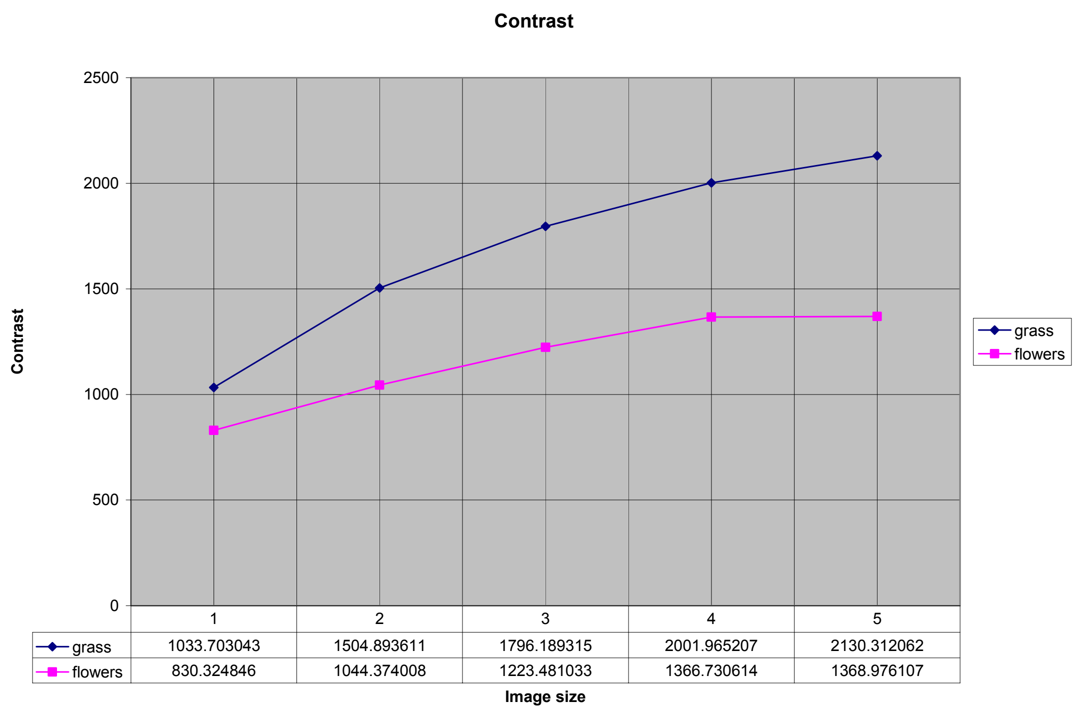
	1	2	3	4	5
▲ grass	0.000274	0.000228	0.000214	0.000207	0.000201
✳ flowers	0.000331	0.000287	0.000271	0.000259	0.000244

Figure 11b. Energy (grass, flowers images)



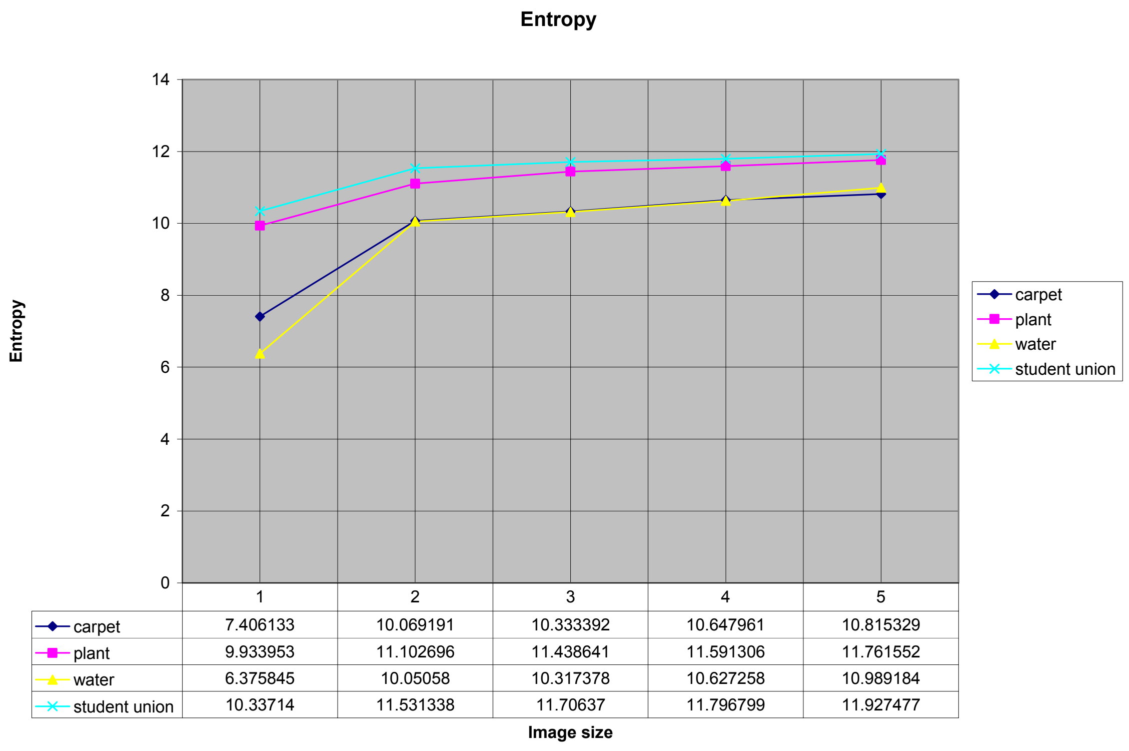
	1	2	3	4	5
◆ carpet	55.09717	113.590671	150.896855	224.319706	293.362159
■ plant	189.30629	215.754194	239.403565	273.152622	318.245075
▲ water	28.436688	64.825681	79.04696	102.456394	159.59371
✕ student union	213.606395	301.669708	344.010065	351.473272	381.208912

Figure 12a. Contrast (carpet, plant, water, student union images)



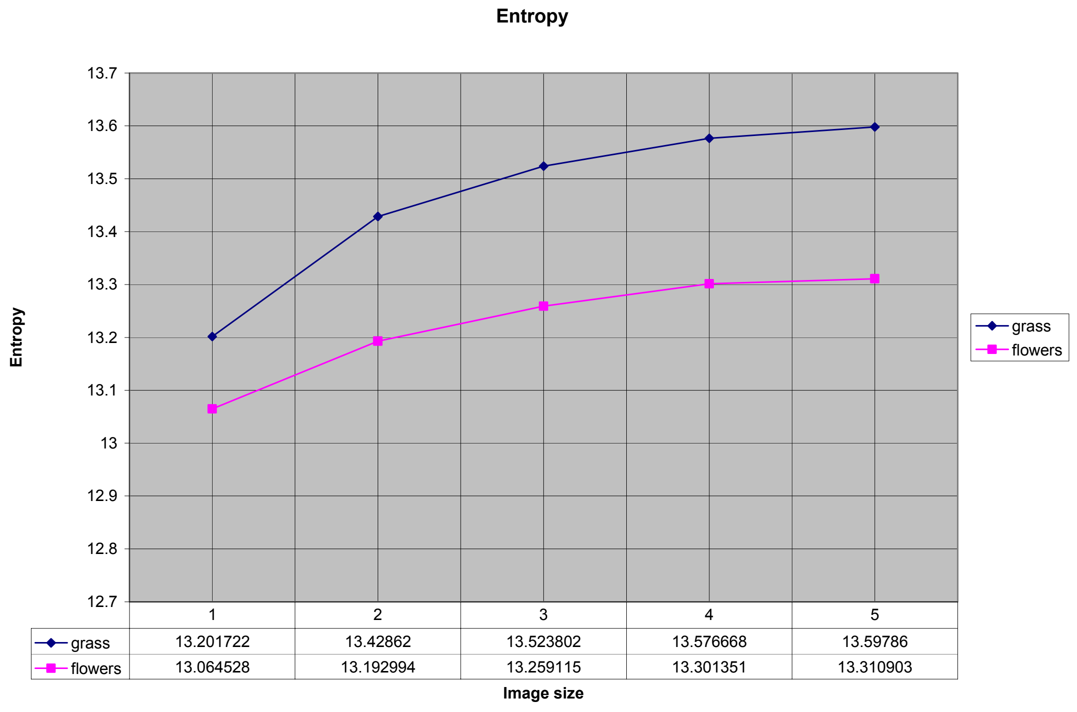
	1	2	3	4	5
grass	1033.703043	1504.893611	1796.189315	2001.965207	2130.312062
flowers	830.324846	1044.374008	1223.481033	1366.730614	1368.976107

Figure 12b. Contrast (grass, flowers images)



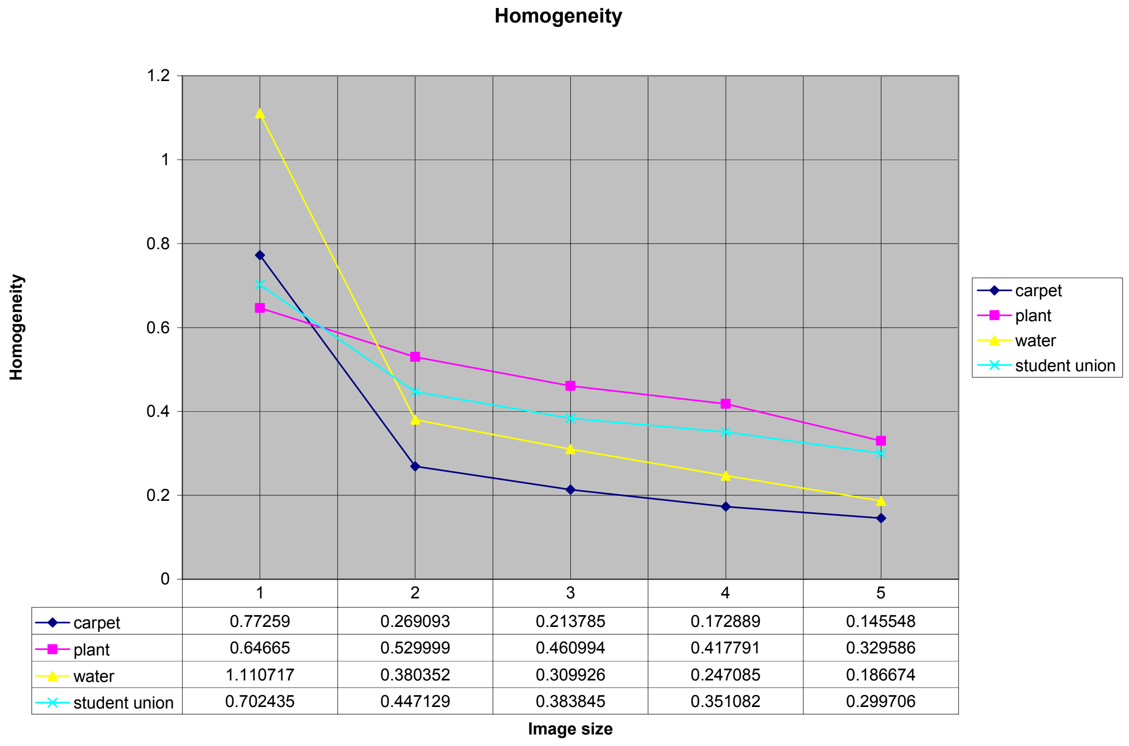
	1	2	3	4	5
carpet	7.406133	10.069191	10.333392	10.647961	10.815329
plant	9.933953	11.102696	11.438641	11.591306	11.761552
water	6.375845	10.05058	10.317378	10.627258	10.989184
student union	10.33714	11.531338	11.70637	11.796799	11.927477

Figure 13a. Entropy (carpet, plant, water, student union images)



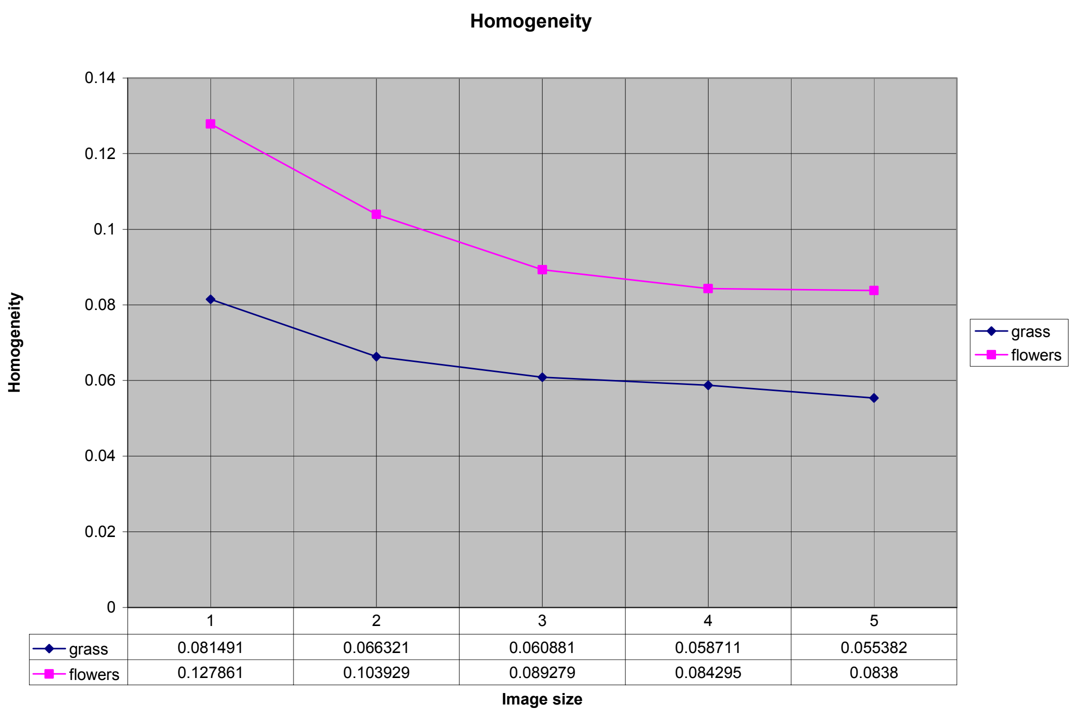
	1	2	3	4	5
grass	13.201722	13.42862	13.523802	13.576668	13.59786
flowers	13.064528	13.192994	13.259115	13.301351	13.310903

Figure 13b. Entropy (grass, flowers images)



	1	2	3	4	5
carpet	0.77259	0.269093	0.213785	0.172889	0.145548
plant	0.64665	0.529999	0.460994	0.417791	0.329586
water	1.110717	0.380352	0.309926	0.247085	0.186674
student union	0.702435	0.447129	0.383845	0.351082	0.299706

Figure 14a. Homogeneity (carpet, plant, water, student union images)



	1	2	3	4	5
grass	0.081491	0.066321	0.060881	0.058711	0.055382
flowers	0.127861	0.103929	0.089279	0.084295	0.0838

Figure 14b. Homogeneity (grass, flowers images)

The analysis showed that energy and homogeneity decrease with increasing image quality, whereas contrast and entropy showed consistent increase with increasing image quality for all the images. There is no change in the sign of first derivative.

The next task was to study the effect of the values of angle on the textural parameters. Hence, four images were generated having orientations in the vertical, horizontal, front diagonal and back diagonal directions. Each image was processed for four combinations of radius and angle that is 1 and 0°, 1 and 45°, 1 and 90° and finally 1 and 135°.

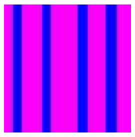


Figure 15a. Dataset 2 Vertical strips

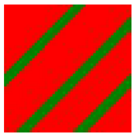


Figure 15c. Dataset 2 Front diagonal strips

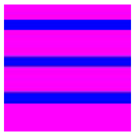


Figure 15b. Dataset 2 Horizontal strips

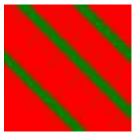
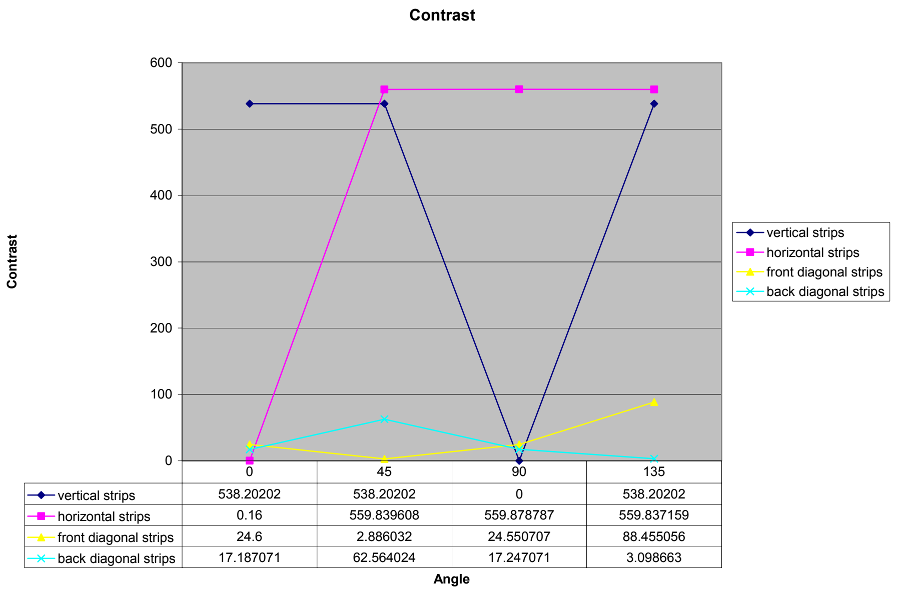


Figure 15d. Dataset 2 Back diagonal strips

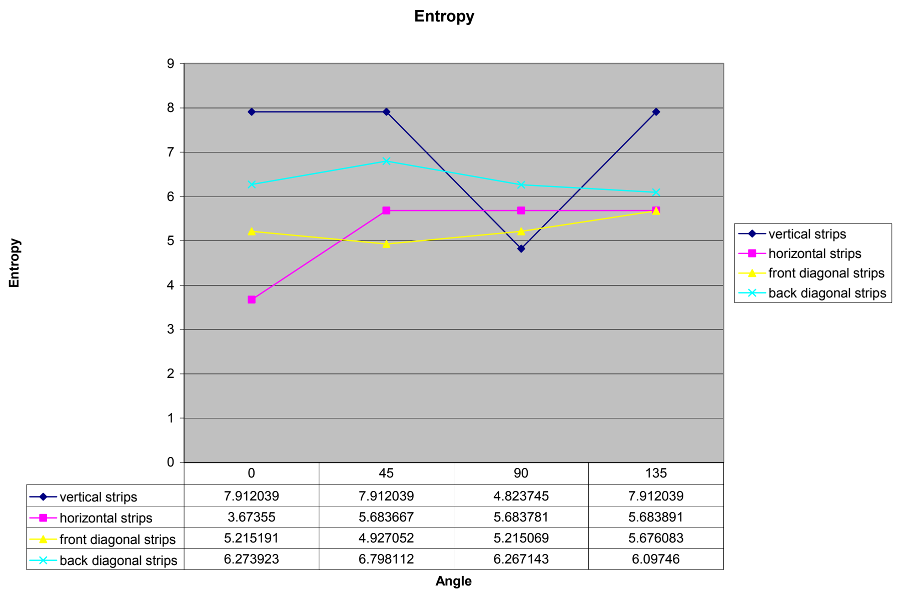
The objective was to observe whether the selected value of angle of GLCM had any specific relationship with the orientation of image content, for instance, to observe whether textural parameter of vertical strip image showed maximum/minimum value for 90° as compared to 0°, 45° and 135°.



	0	45	90	135
vertical strips	538.20202	538.20202	0	538.20202
horizontal strips	0.16	559.839608	559.878787	559.837159
front diagonal strips	24.6	2.886032	24.550707	88.455056
back diagonal strips	17.187071	62.564024	17.247071	3.098663

Figure 16. Dataset 2 Contrast

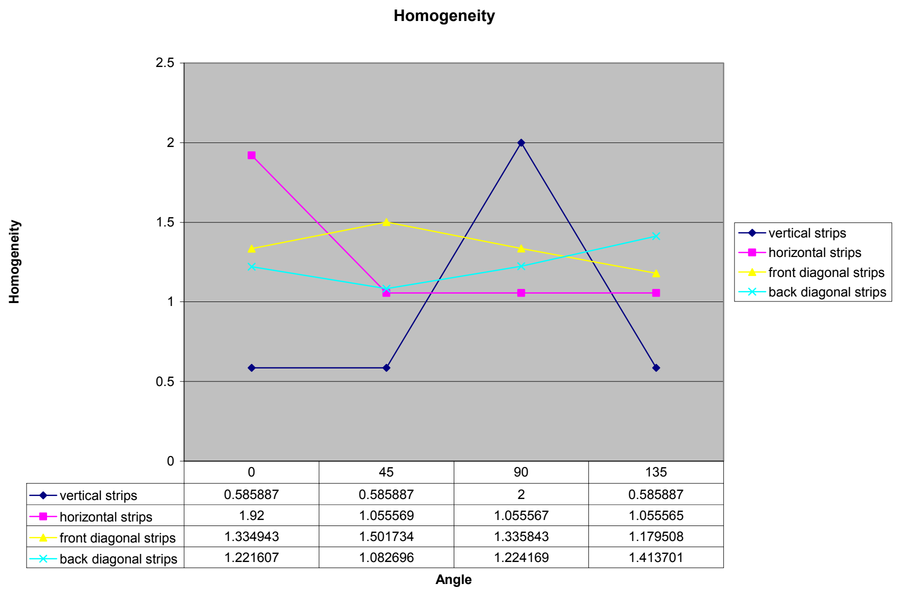
It can be seen that contrast value is least at 90° for vertical strips image, at 0° for horizontal image, at 45° for front diagonal image and at 135° for back diagonal image.



	0	45	90	135
vertical strips	7.912039	7.912039	4.823745	7.912039
horizontal strips	3.67355	5.683667	5.683781	5.683891
front diagonal strips	5.215191	4.927052	5.215069	5.676083
back diagonal strips	6.273923	6.798112	6.267143	6.09746

Figure 17. Dataset 2 Entropy

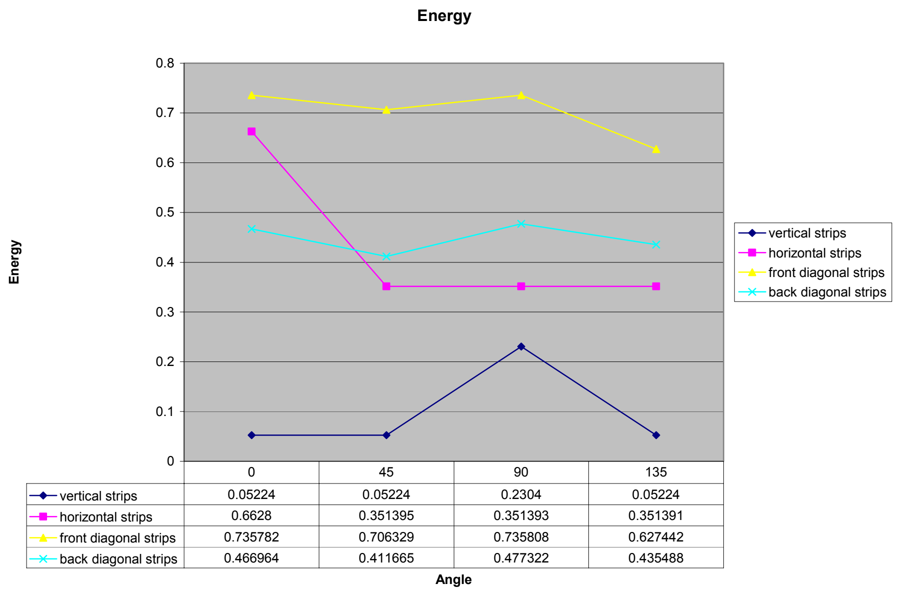
It can be seen that entropy value is least at 90° for vertical strips image, at 0° for horizontal image, at 45° for front diagonal image and at 135° for back diagonal image.



	0	45	90	135
vertical strips	0.585887	0.585887	2	0.585887
horizontal strips	1.92	1.055569	1.055567	1.055565
front diagonal strips	1.334943	1.501734	1.335843	1.179508
back diagonal strips	1.221607	1.082696	1.224169	1.413701

Figure 18. Dataset 2 Homogeneity

It can be seen that homogeneity value is maximum at 90° for vertical strips image, at 0° for horizontal image, at 45° for front diagonal image and at 135° for back diagonal image.



	0	45	90	135
vertical strips	0.05224	0.05224	0.2304	0.05224
horizontal strips	0.6628	0.351395	0.351393	0.351391
front diagonal strips	0.735782	0.706329	0.735808	0.627442
back diagonal strips	0.466964	0.411665	0.477322	0.435488

Figure 19. Dataset 2 Energy

It can be seen that energy value is maximum at 90° for vertical strips image and at 0° for horizontal image. However these trends are not followed for front diagonal image and back diagonal image! As it can be observed that the energy value is not maximum for the front diagonal image for 45° as well as for the back diagonal image for 135°. This helps in concluding that it would be difficult to use the orientation of the overall image content in deciding the most appropriate value of angle for GLCM processing.

Hence in the final part of experimentation it was decided to do exhaustive runs for all combinations of radius and angle. A dataset called FACE was used which comprises of images of human faces. To start of with, this image was saved at quality levels 1, 3 and 5 using standard jpeg compression. The FACE dataset consisted a total of 48 runs, each requiring approximately processing time of 2 minutes.

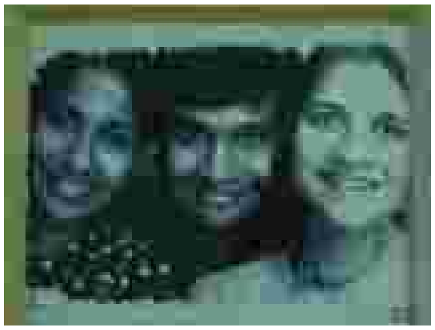


Figure 20a. FACE quality level 1 (6KB)

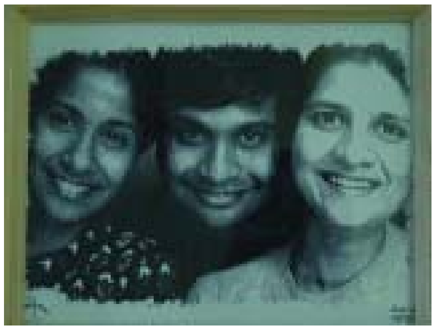


Figure 20c. FACE quality level 5 (6KB)

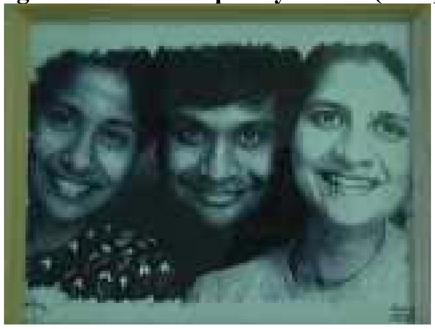


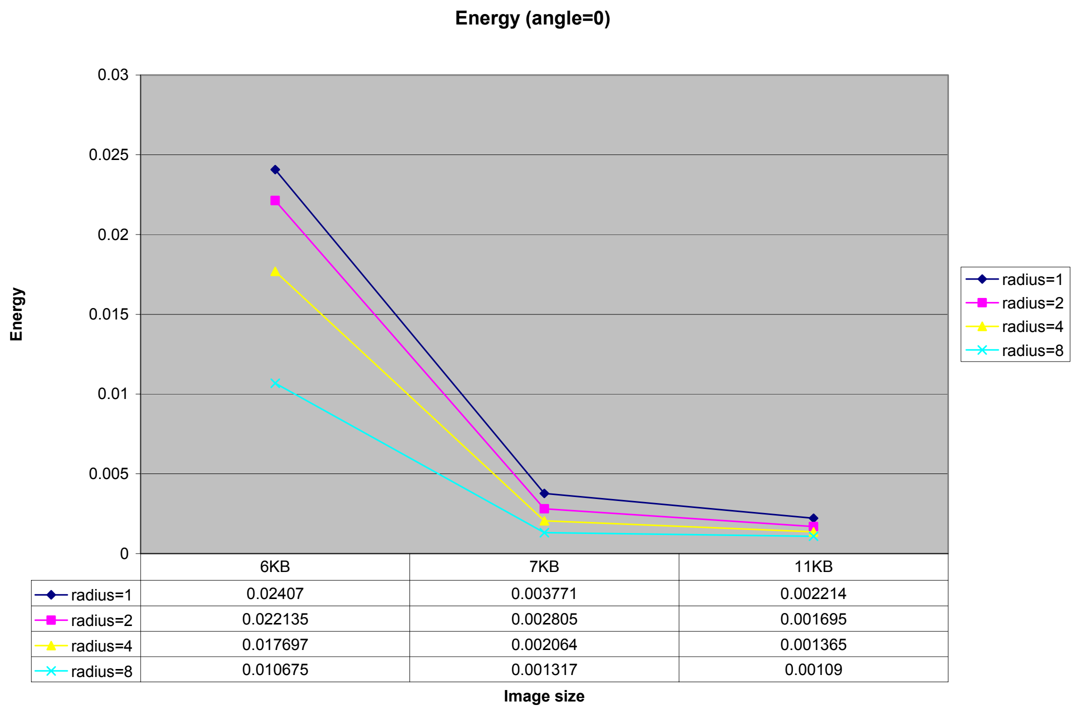
Figure 20b. FACE quality level 3 (7KB)

Image size	Radius	Angle	Energy	Contrast	Entropy	Homogeneity
6KB	1	0	0.02407	171.761	10.26177	0.617437
7KB	1	0	0.003771	202.8114	11.07969	0.456161
11KB	1	0	0.002214	237.0179	11.19007	0.393059
6KB	1	45	0.019735	377.2859	10.88752	0.414932
7KB	1	45	0.002648	395.5287	11.6069	0.282185
11KB	1	45	0.001759	422.8177	11.57984	0.267074
6KB	1	90	0.02235	347.0806	10.5414	0.552665
7KB	1	90	0.003593	361.9449	11.34828	0.400357
11KB	1	90	0.002261	397.3172	11.36973	0.377045
6KB	1	135	0.019751	387.3405	11.00565	0.411819
7KB	1	135	0.002609	411.6924	11.72667	0.274801
11KB	1	135	0.001726	442.775	11.69309	0.269048
6KB	2	0	0.022135	469.401	10.69552	0.526195
7KB	2	0	0.002805	502.9805	11.56605	0.358082
11KB	2	0	0.001695	529.2688	11.59099	0.31164
6KB	2	45	0.015361	969.8808	11.37108	0.312272
7KB	2	45	0.001908	972.4885	12.01651	0.203788
11KB	2	45	0.001374	986.6043	11.98389	0.209876
6KB	2	90	0.018862	896.1711	11.15472	0.452017
7KB	2	90	0.002565	946.5845	11.96345	0.297973
11KB	2	90	0.00167	977.4247	11.96004	0.281211
6KB	2	135	0.015742	1021.588	11.59683	0.30791
7KB	2	135	0.001943	1027.106	12.24931	0.200889
11KB	2	135	0.001325	1040.207	12.21975	0.212727
6KB	4	0	0.017697	1039.107	11.09772	0.428483
7KB	4	0	0.002064	1008.678	11.88342	0.278374
11KB	4	0	0.001365	1025.913	11.86913	0.248683
6KB	4	45	0.008784	2067.087	11.79518	0.209122
7KB	4	45	0.001269	2042.199	12.33174	0.144673
11KB	4	45	0.001014	2068.228	12.2902	0.146804
6KB	4	90	0.012794	2063.425	11.81806	0.335178
7KB	4	90	0.001826	2082.977	12.55724	0.213546

11KB	4	90	0.00131	2108.965	12.55691	0.208584
6KB	4	135	0.009972	2310.118	12.27307	0.212055
7KB	4	135	0.001317	2266.555	12.82502	0.143375
11KB	4	135	0.000971	2287.544	12.79987	0.143962
6KB	8	0	0.010675	1780.929	11.46861	0.28607
7KB	8	0	0.001317	1730.349	12.09104	0.194486
11KB	8	0	0.00109	1758.263	12.04626	0.188907
6KB	8	45	0.002938	3954.321	12.01462	0.108293
7KB	8	45	0.000767	3948.928	12.48789	0.090927
11KB	8	45	0.000737	3983.652	12.41357	0.090491
6KB	8	90	0.005912	4229.268	12.62757	0.164786
7KB	8	90	0.001249	4187.663	13.26445	0.123869
11KB	8	90	0.001173	4213.264	13.17731	0.14189
6KB	8	135	0.006251	4696.982	12.98565	0.124847
7KB	8	135	0.000964	4594.344	13.45011	0.090663
11KB	8	135	0.000713	4641.184	13.38461	0.084786

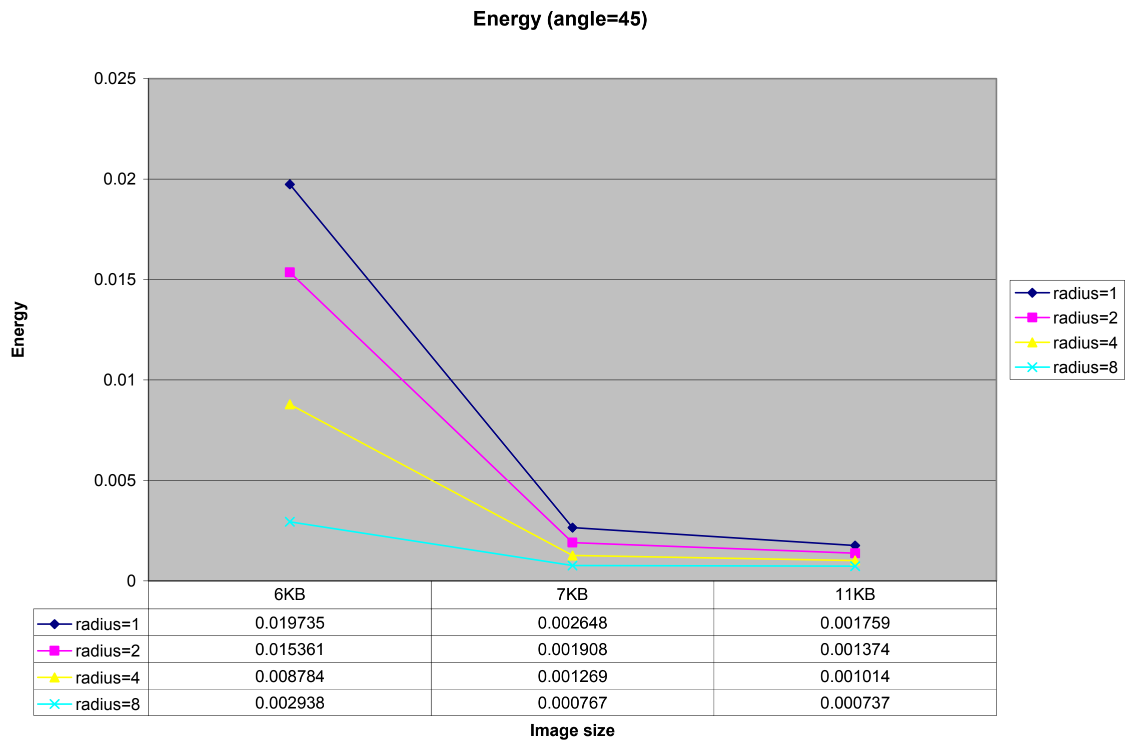
Table 2. Dataset 3 FACE image

All the four parameters, energy, contrast, entropy and homogeneity were studied. The study starts with the energy feature for which eight graphs were plotted. The set of first four graphs depicts variation in radius for specific angles:



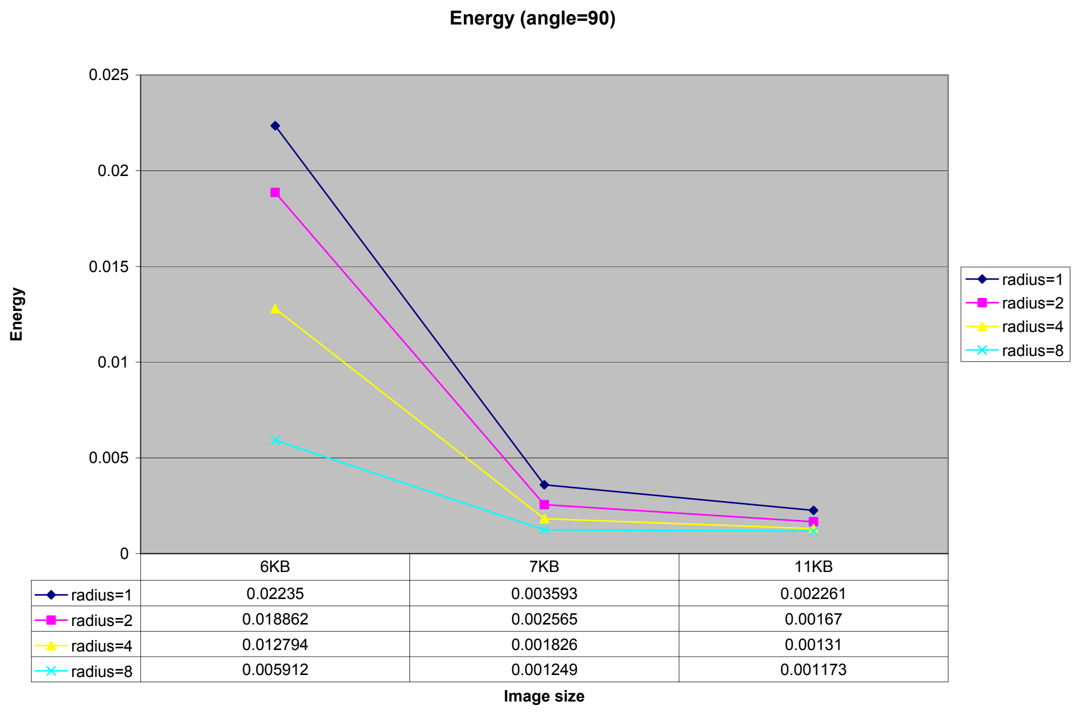
	6KB	7KB	11KB
radius=1	0.02407	0.003771	0.002214
radius=2	0.022135	0.002805	0.001695
radius=4	0.017697	0.002064	0.001365
radius=8	0.010675	0.001317	0.00109

Figure 21a. Energy angle=0



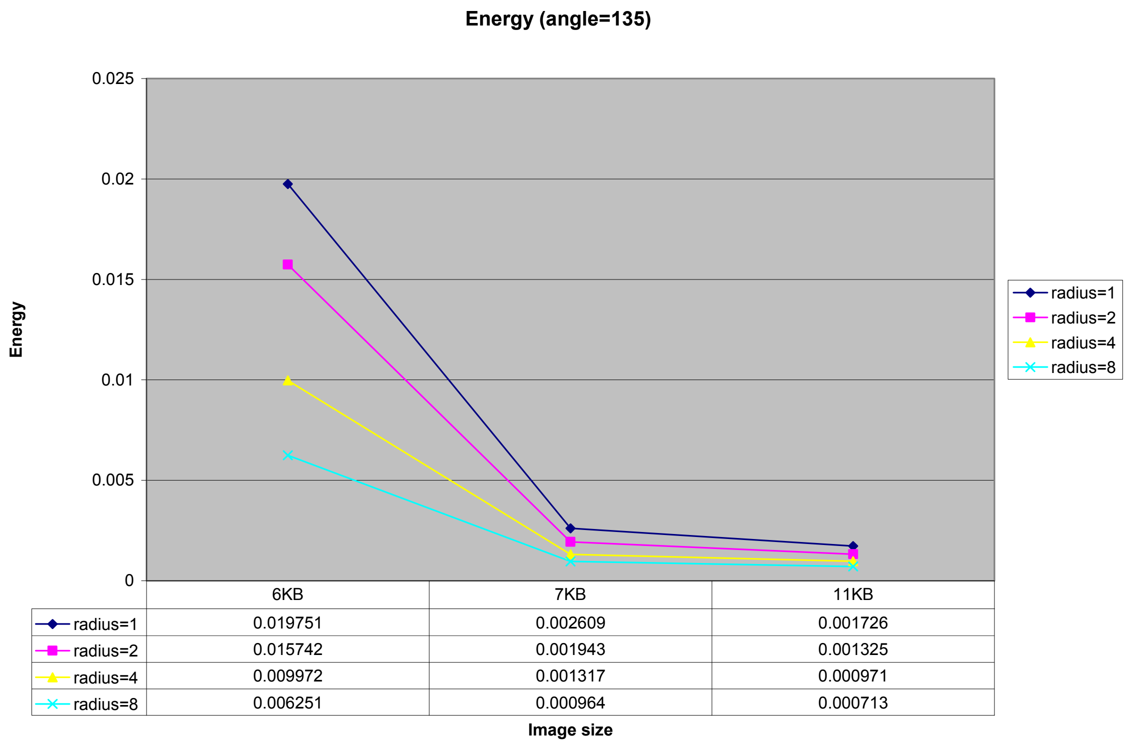
	6KB	7KB	11KB
radius=1	0.019735	0.002648	0.001759
radius=2	0.015361	0.001908	0.001374
radius=4	0.008784	0.001269	0.001014
radius=8	0.002938	0.000767	0.000737

Figure 21b. Energy angle=45



	6KB	7KB	11KB
◆ radius=1	0.02235	0.003593	0.002261
■ radius=2	0.018862	0.002565	0.00167
▲ radius=4	0.012794	0.001826	0.00131
× radius=8	0.005912	0.001249	0.001173

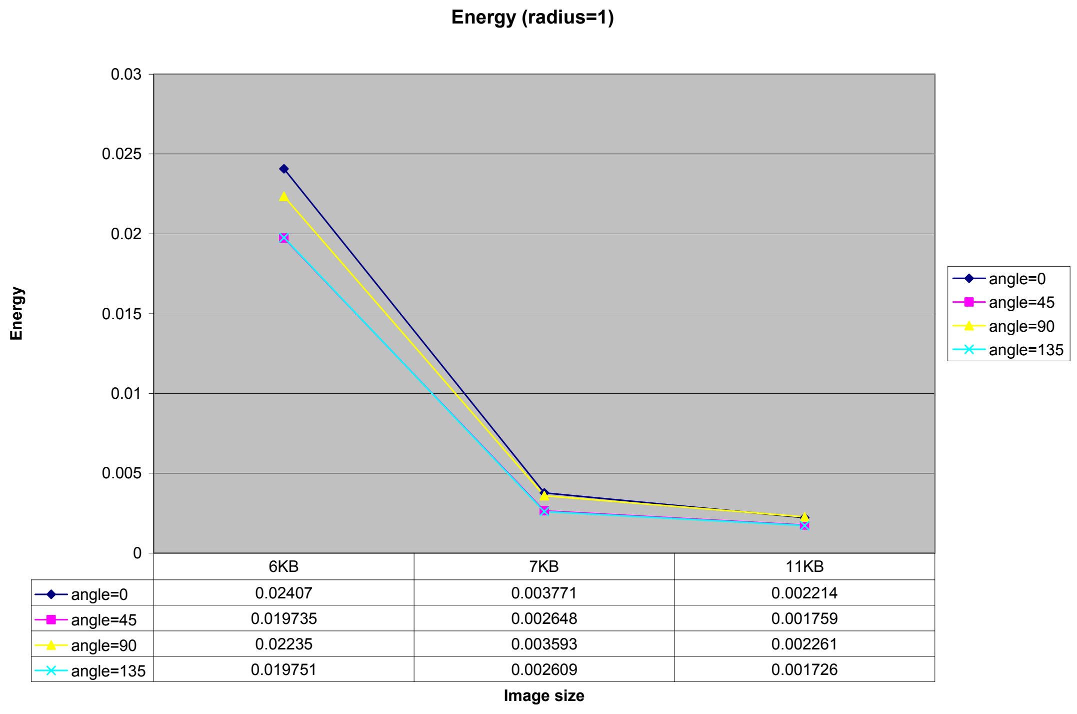
Figure 21c. Energy angle=90



	6KB	7KB	11KB
◆ radius=1	0.019751	0.002609	0.001726
■ radius=2	0.015742	0.001943	0.001325
▲ radius=4	0.009972	0.001317	0.000971
× radius=8	0.006251	0.000964	0.000713

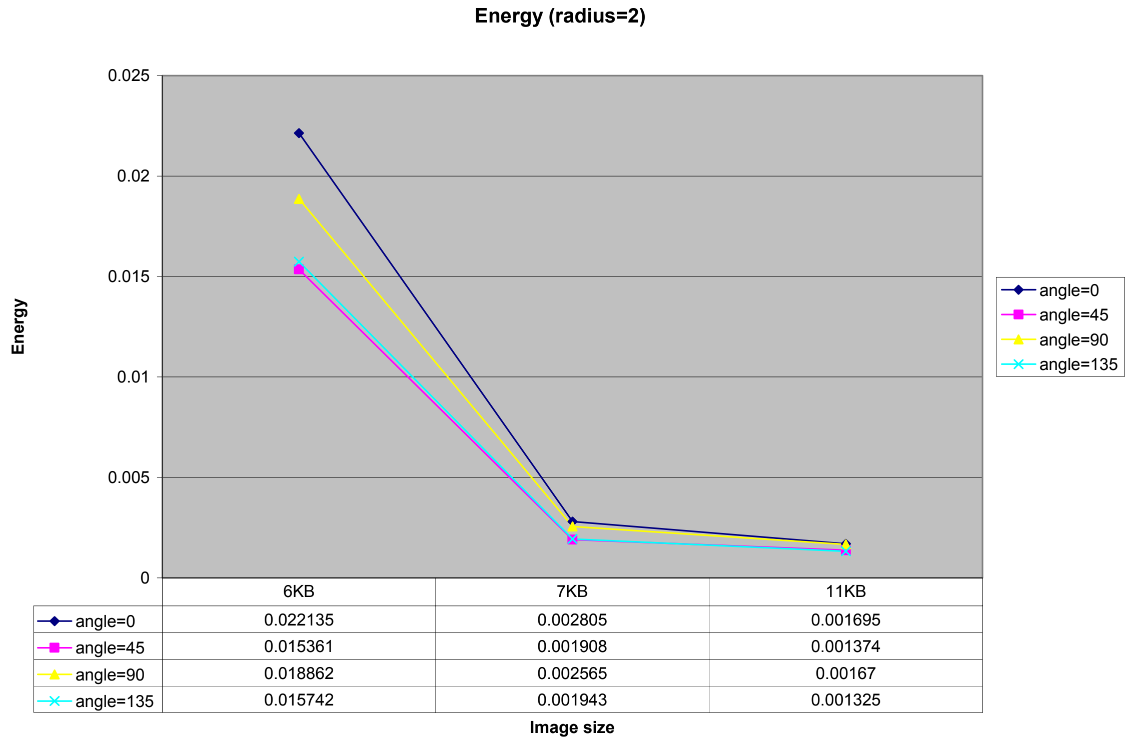
Figure 21d. Energy angle=135

The set of next four graphs depicts variation in angle for specific values of radius.



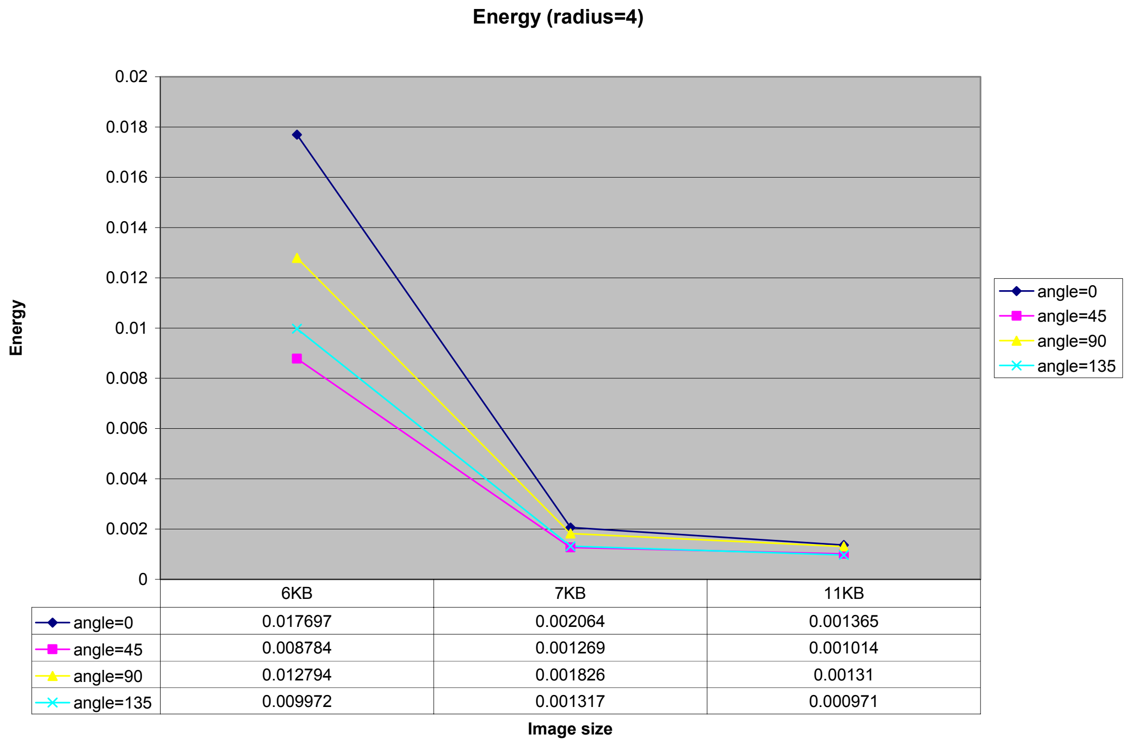
	6KB	7KB	11KB
angle=0	0.02407	0.003771	0.002214
angle=45	0.019735	0.002648	0.001759
angle=90	0.02235	0.003593	0.002261
angle=135	0.019751	0.002609	0.001726

Figure 21e. Energy radius=1



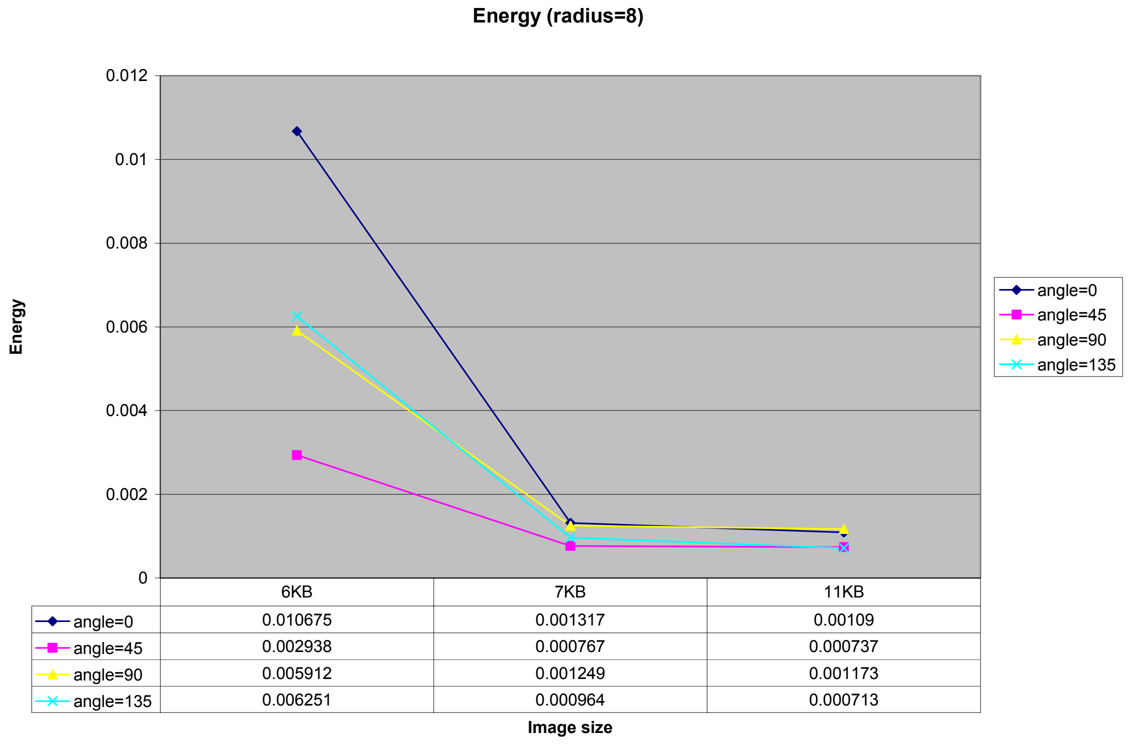
	6KB	7KB	11KB
angle=0	0.022135	0.002805	0.001695
angle=45	0.015361	0.001908	0.001374
angle=90	0.018862	0.002565	0.00167
angle=135	0.015742	0.001943	0.001325

Figure 21f. Energy radius=2



	6KB	7KB	11KB
angle=0	0.017697	0.002064	0.001365
angle=45	0.008784	0.001269	0.001014
angle=90	0.012794	0.001826	0.00131
angle=135	0.009972	0.001317	0.000971

Figure 21g. Energy radius=4

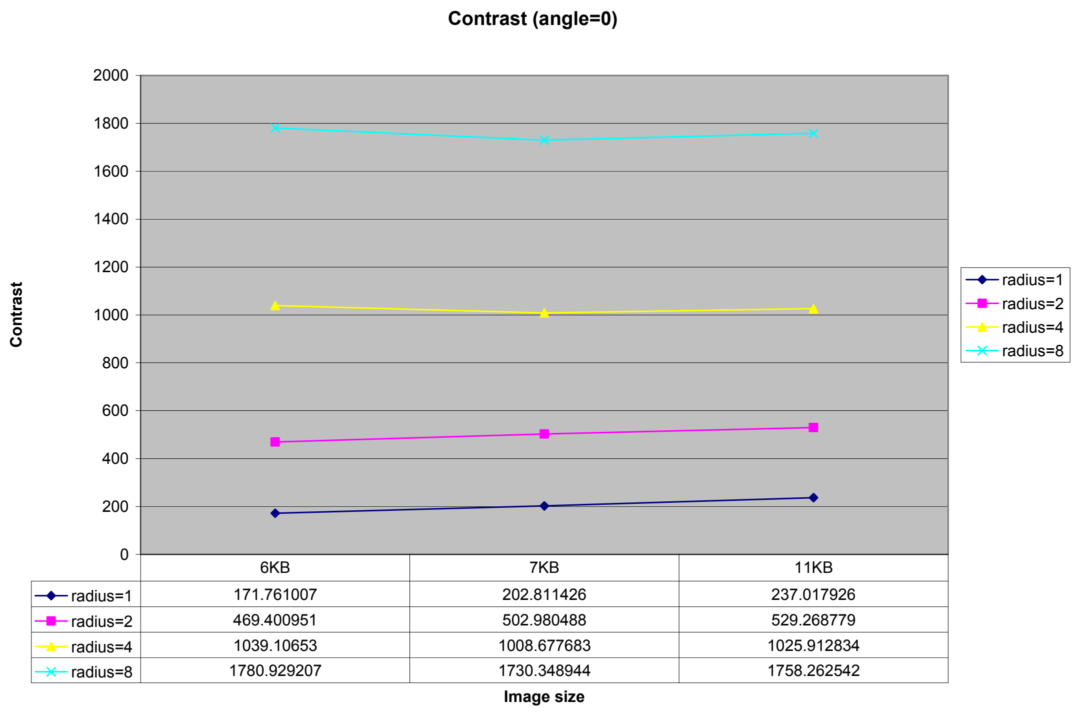


	6KB	7KB	11KB
angle=0	0.010675	0.001317	0.00109
angle=45	0.002938	0.000767	0.000737
angle=90	0.005912	0.001249	0.001173
angle=135	0.006251	0.000964	0.000713

Figure 21h. Energy radius=8

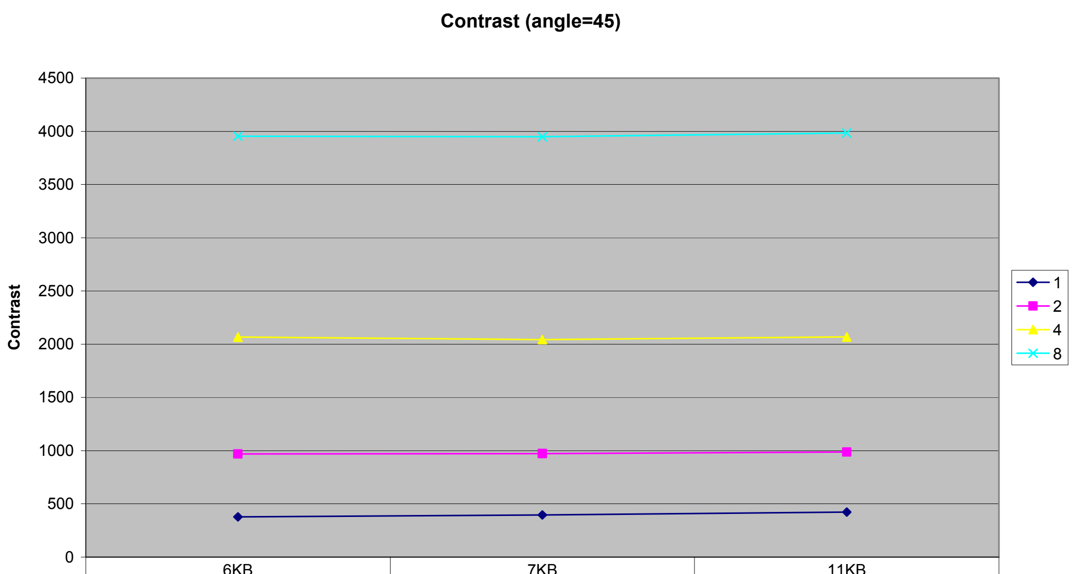
The study of the graphs shows that for all the combinations of radius (1, 2, 4, 8) and angle (0°, 45°, 90°, 135°), energy always decreases with increase in image quality.

On similar basis, the contrast feature was studied:



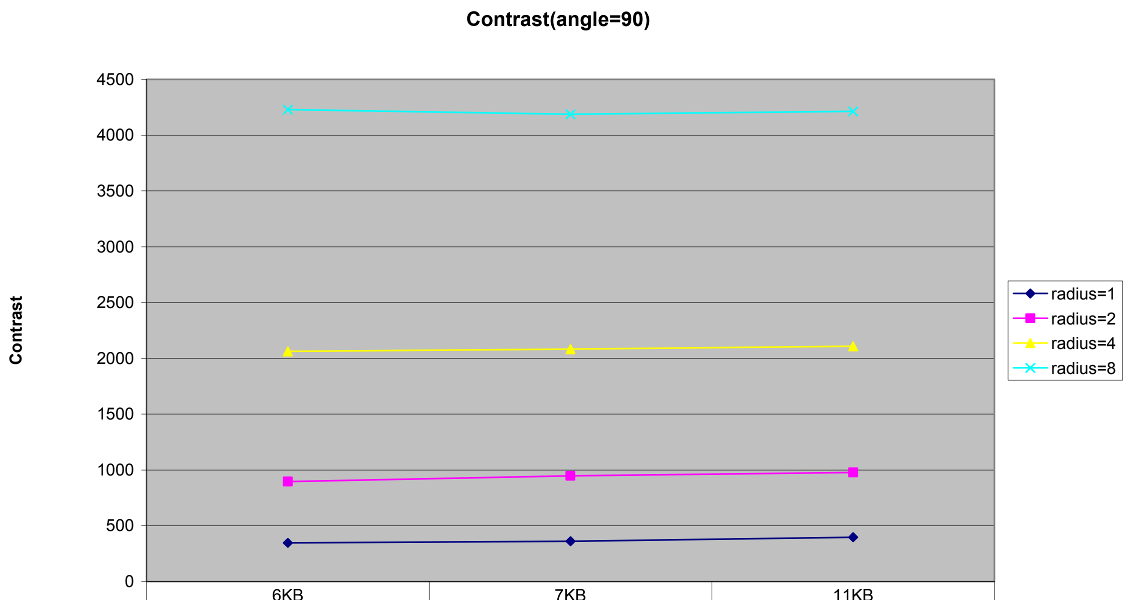
	6KB	7KB	11KB
radius=1	171.761007	202.811426	237.017926
radius=2	469.400951	502.980488	529.268779
radius=4	1039.10653	1008.677683	1025.912834
radius=8	1780.929207	1730.348944	1758.262542

Figure 22a. Contrast angle=0



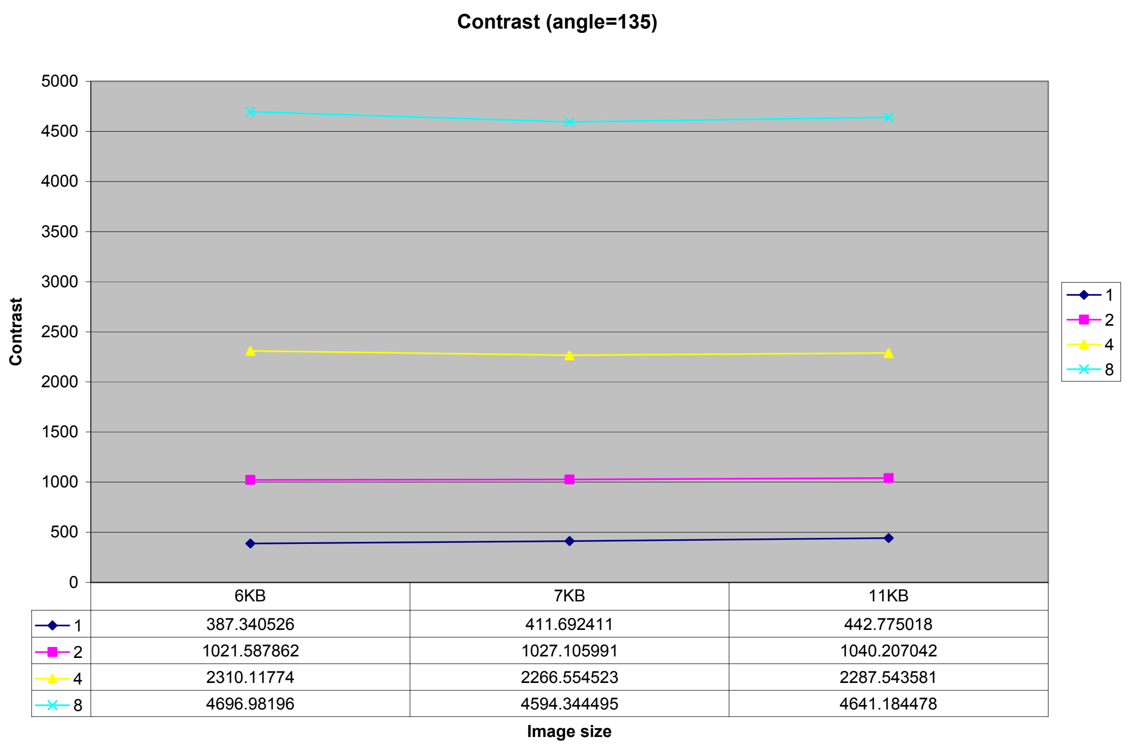
	6KB	7KB	11KB
1	377.285878	395.528677	422.817722
2	969.880824	972.488526	986.604274
4	2067.087458	2042.198973	2068.228261
8	3954.321468	3948.928257	3983.652054

Figure 22b. Contrast angle=45



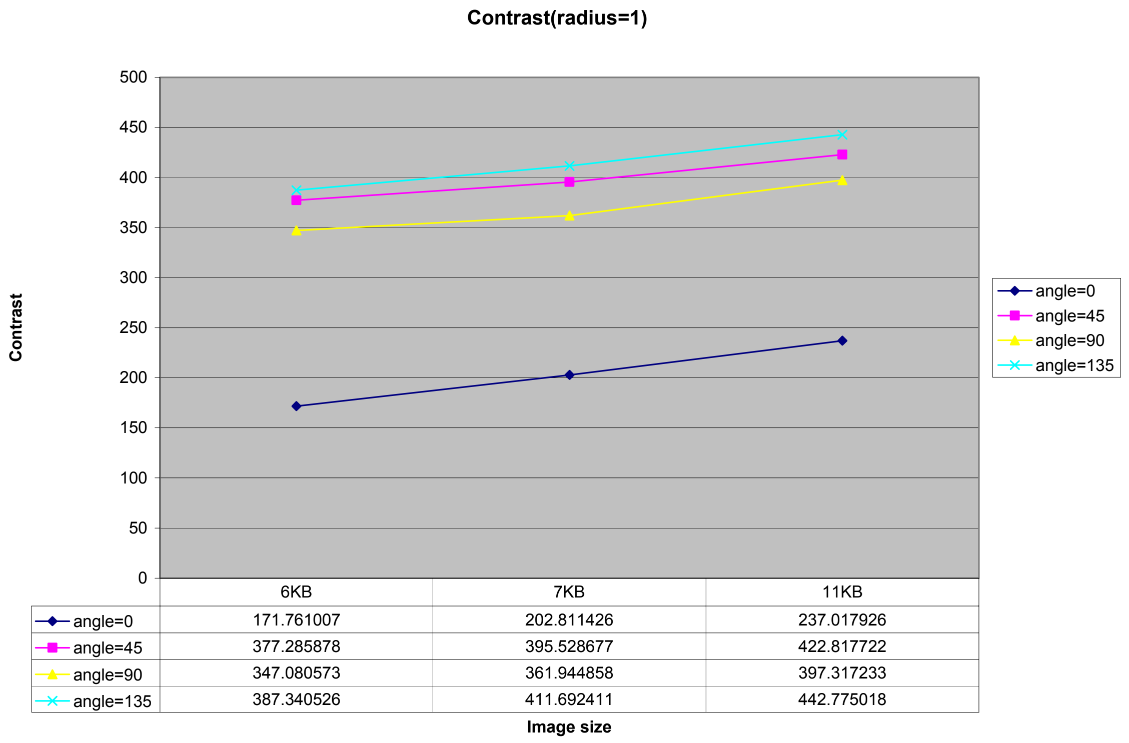
	6KB	7KB	11KB
radius=1	347.080573	361.944858	397.317233
radius=2	896.171074	946.584524	977.424672
radius=4	2063.424566	2082.976612	2108.964541
radius=8	4229.268472	4187.663221	4213.264236

Figure 22c. Contrast angle=90



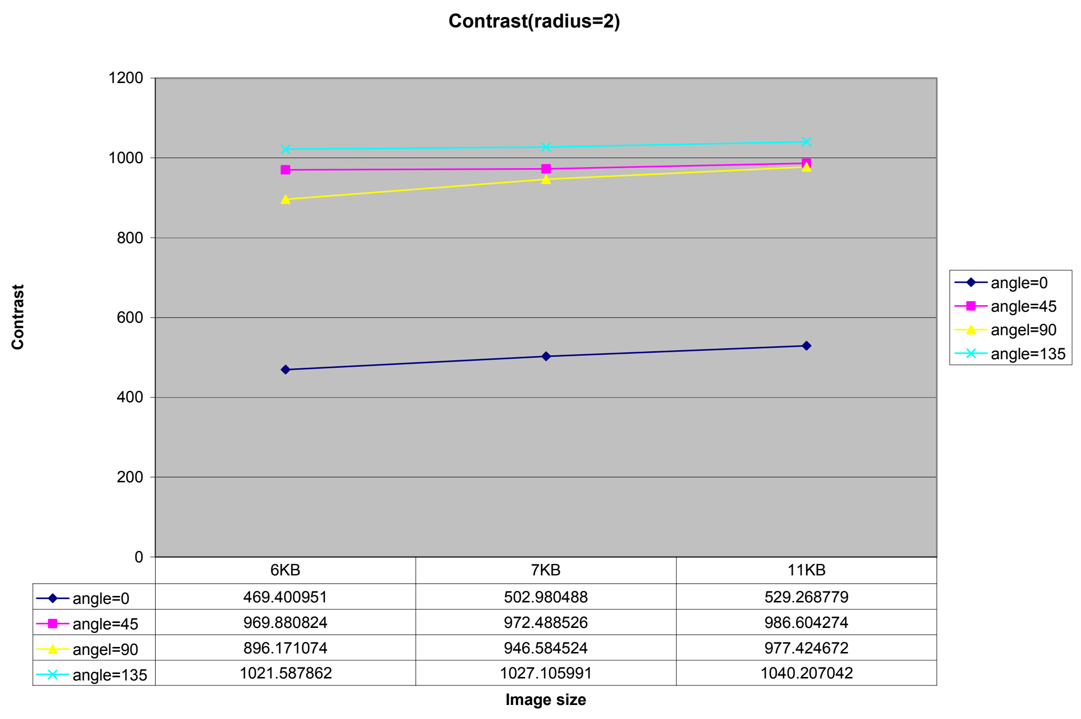
	6KB	7KB	11KB
1	387.340526	411.692411	442.775018
2	1021.587862	1027.105991	1040.207042
4	2310.11774	2266.554523	2287.543581
8	4696.98196	4594.344495	4641.184478

Figure 22d. Contrast angle=135



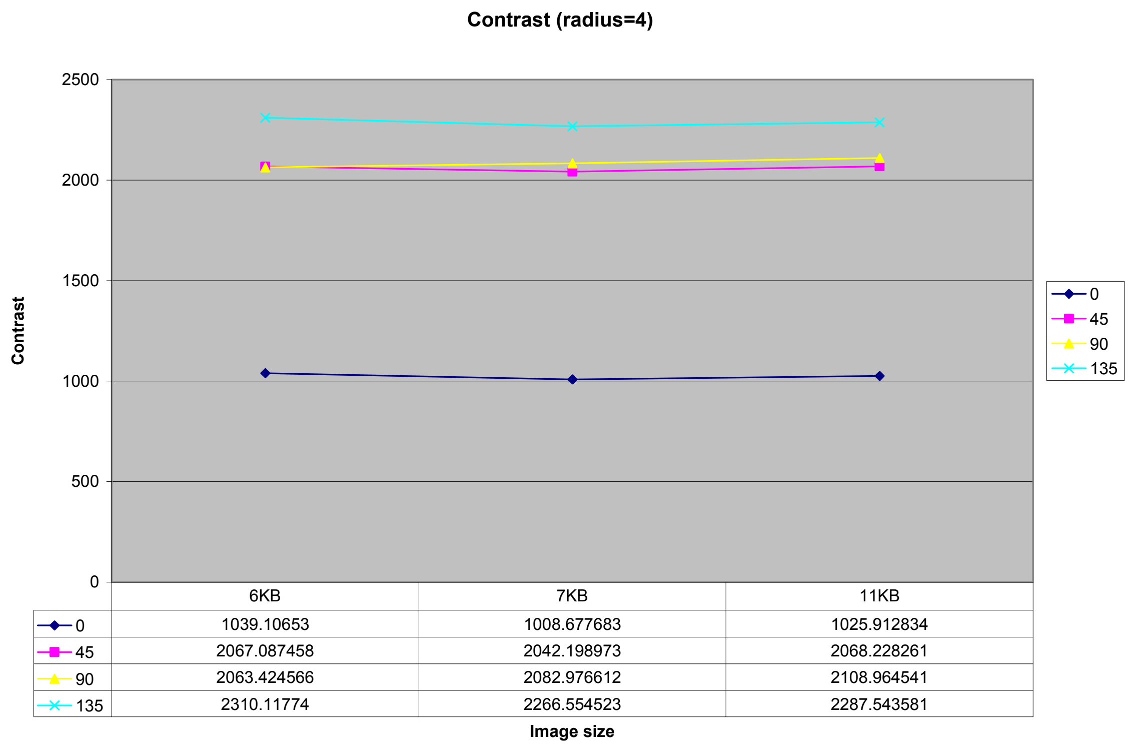
	6KB	7KB	11KB
angle=0	171.761007	202.811426	237.017926
angle=45	377.285878	395.528677	422.817722
angle=90	347.080573	361.944858	397.317233
angle=135	387.340526	411.692411	442.775018

Figure 22e. Contrast radius=1



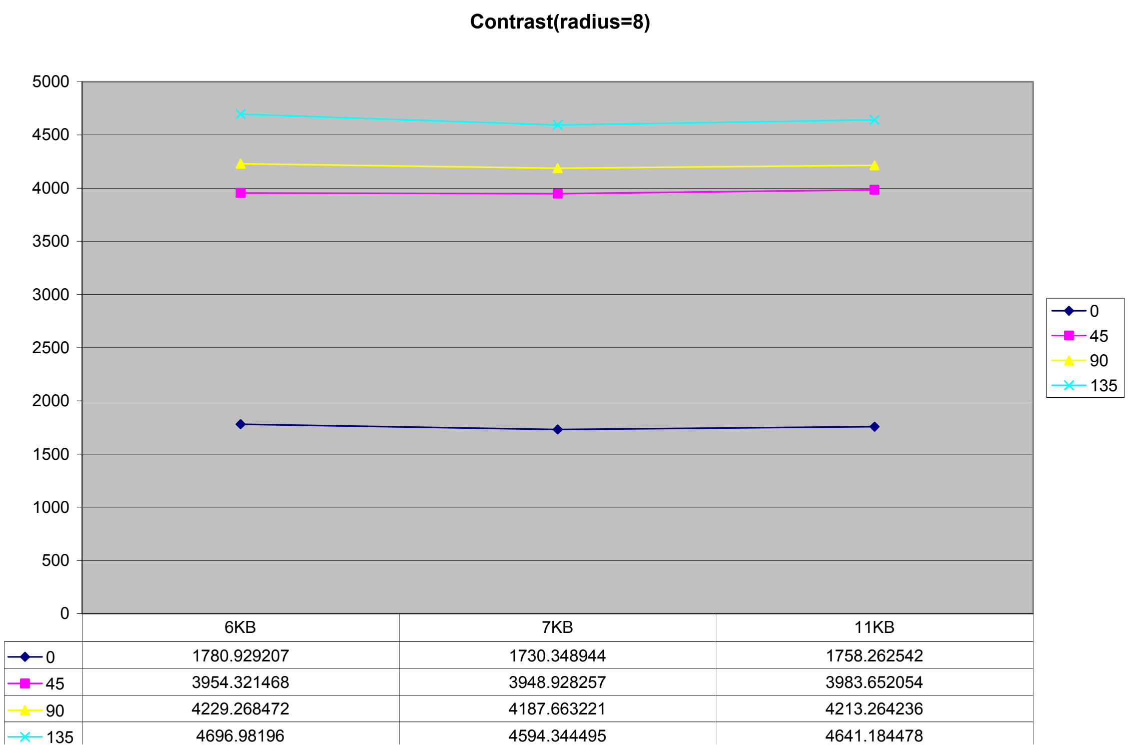
	6KB	7KB	11KB
angle=0	469.400951	502.980488	529.268779
angle=45	969.880824	972.488526	986.604274
angel=90	896.171074	946.584524	977.424672
angle=135	1021.587862	1027.105991	1040.207042

Figure 22f. Contrast radius=2



	6KB	7KB	11KB
0	1039.10653	1008.677683	1025.912834
45	2067.087458	2042.198973	2068.228261
90	2063.424566	2082.976612	2108.964541
135	2310.11774	2266.554523	2287.543581

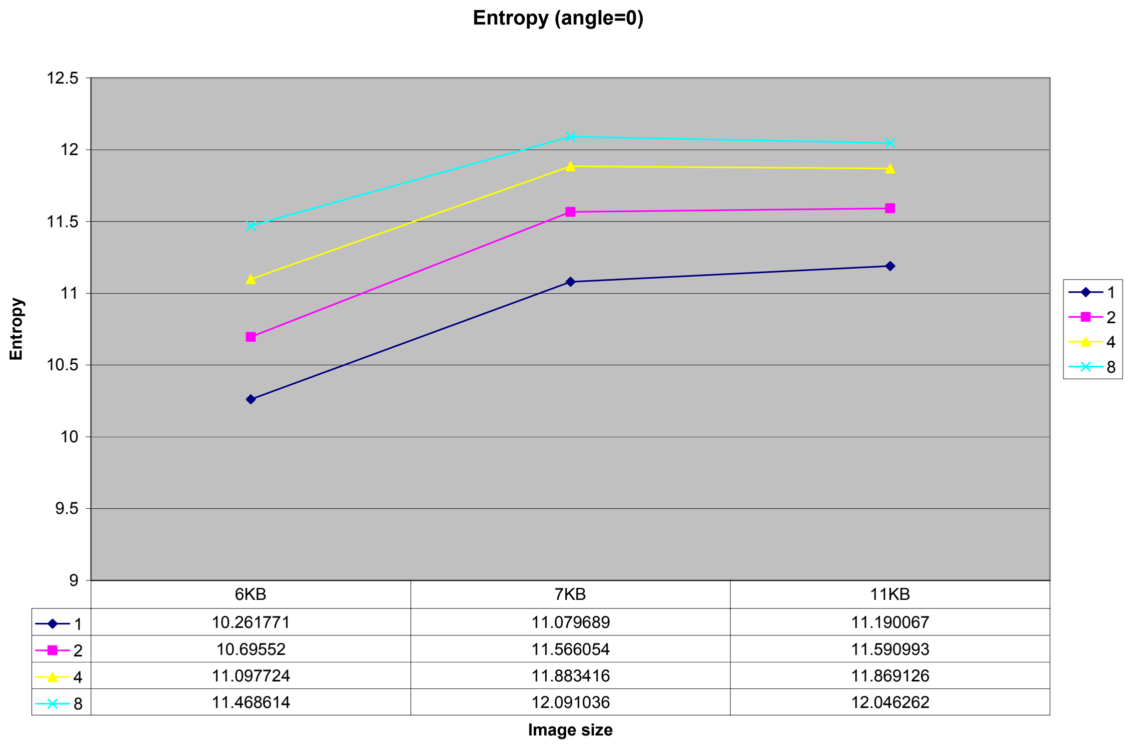
Figure 22g. Contrast radius=4



	6KB	7KB	11KB
0	1780.929207	1730.348944	1758.262542
45	3954.321468	3948.928257	3983.652054
90	4229.268472	4187.663221	4213.264236
135	4696.98196	4594.344495	4641.184478

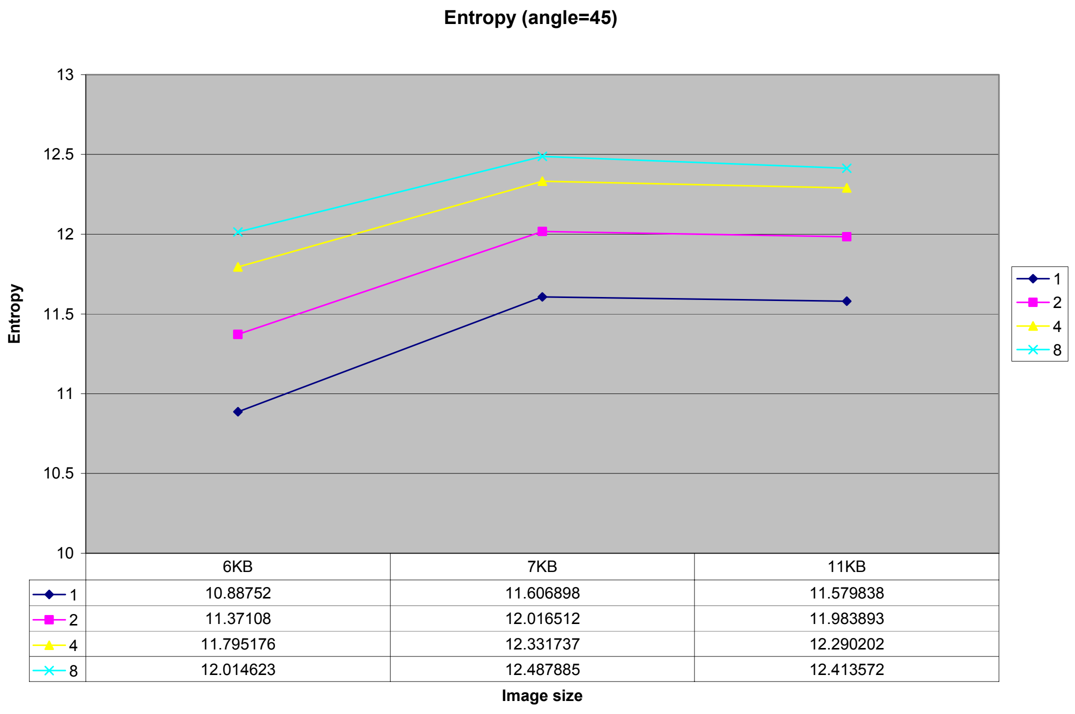
Figure 22h. Contrast radius=8

These are the plots for Entropy feature:



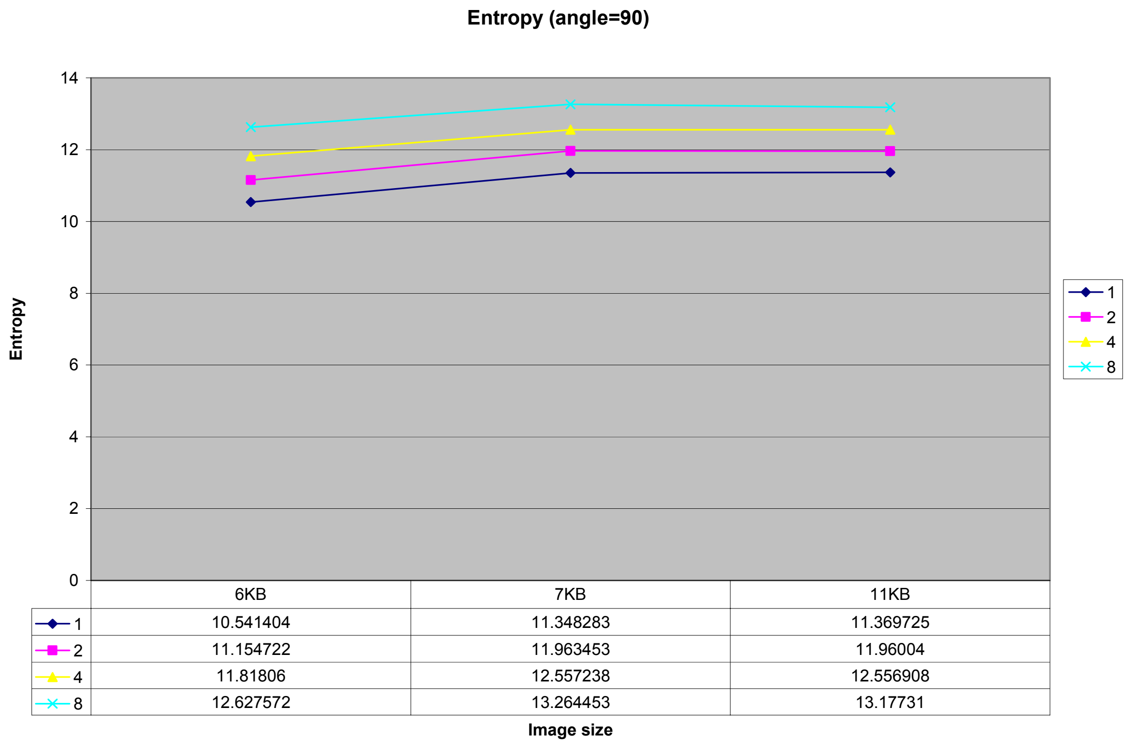
	6KB	7KB	11KB
1	10.261771	11.079689	11.190067
2	10.69552	11.566054	11.590993
4	11.097724	11.883416	11.869126
8	11.468614	12.091036	12.046262

Figure 23a. Entropy angle=0



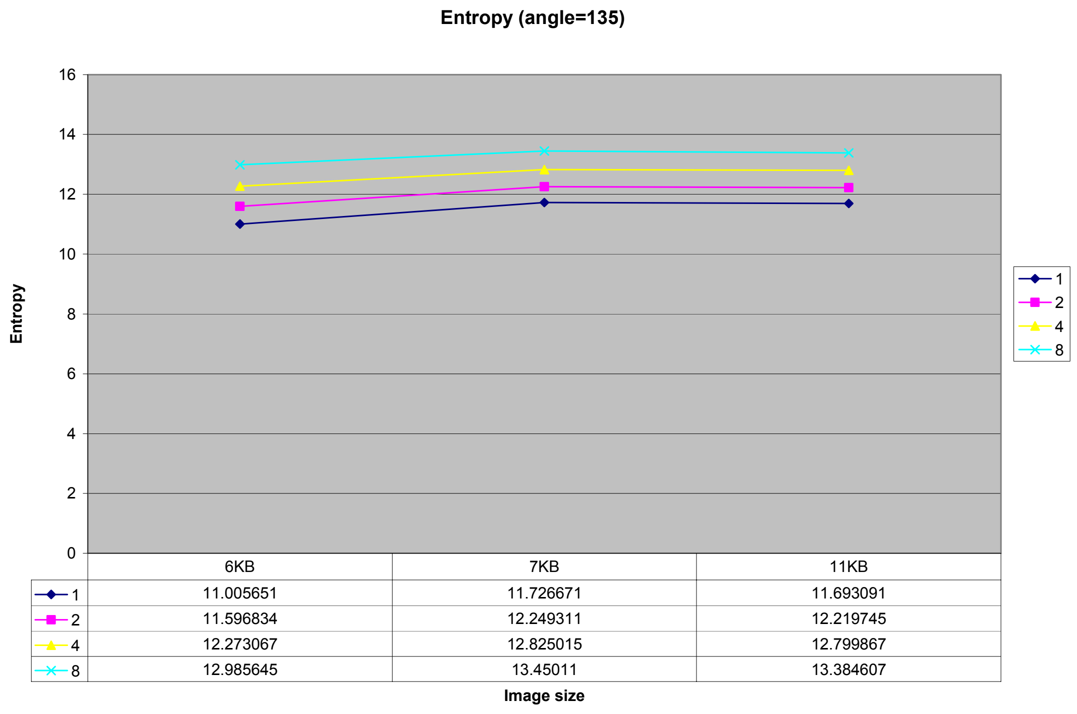
	6KB	7KB	11KB
1	10.88752	11.606898	11.579838
2	11.37108	12.016512	11.983893
4	11.795176	12.331737	12.290202
8	12.014623	12.487885	12.413572

Figure 23b. Entropy angle=45



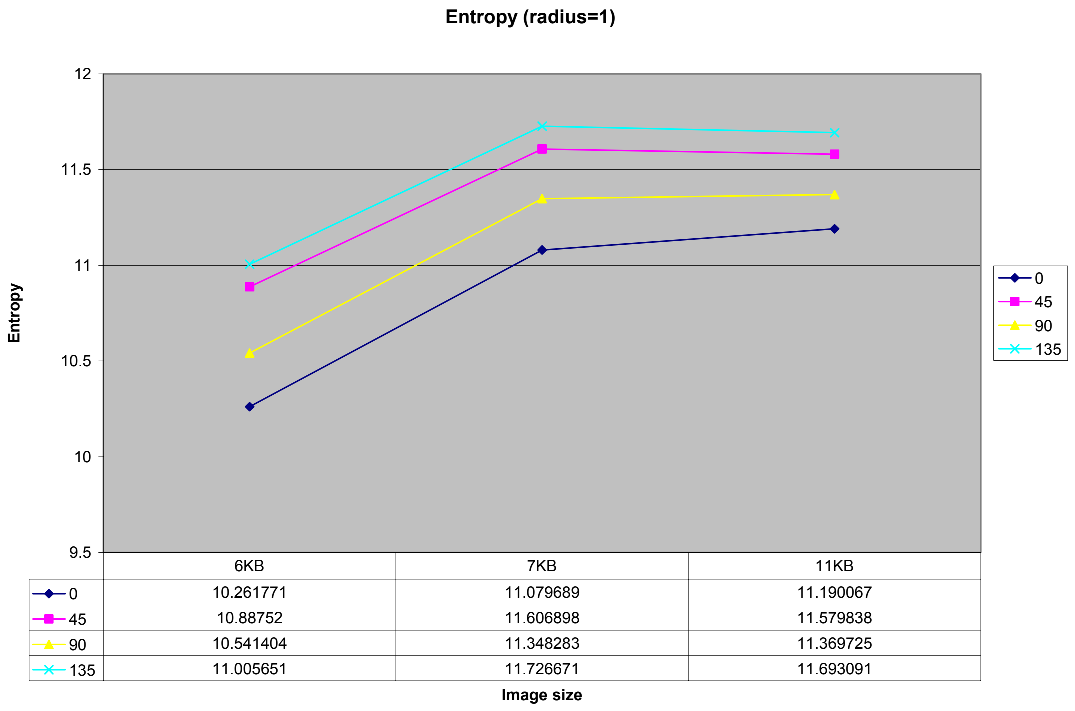
	6KB	7KB	11KB
◆ 1	10.541404	11.348283	11.369725
■ 2	11.154722	11.963453	11.96004
▲ 4	11.81806	12.557238	12.556908
× 8	12.627572	13.264453	13.17731

Figure 23c. Entropy angle=90



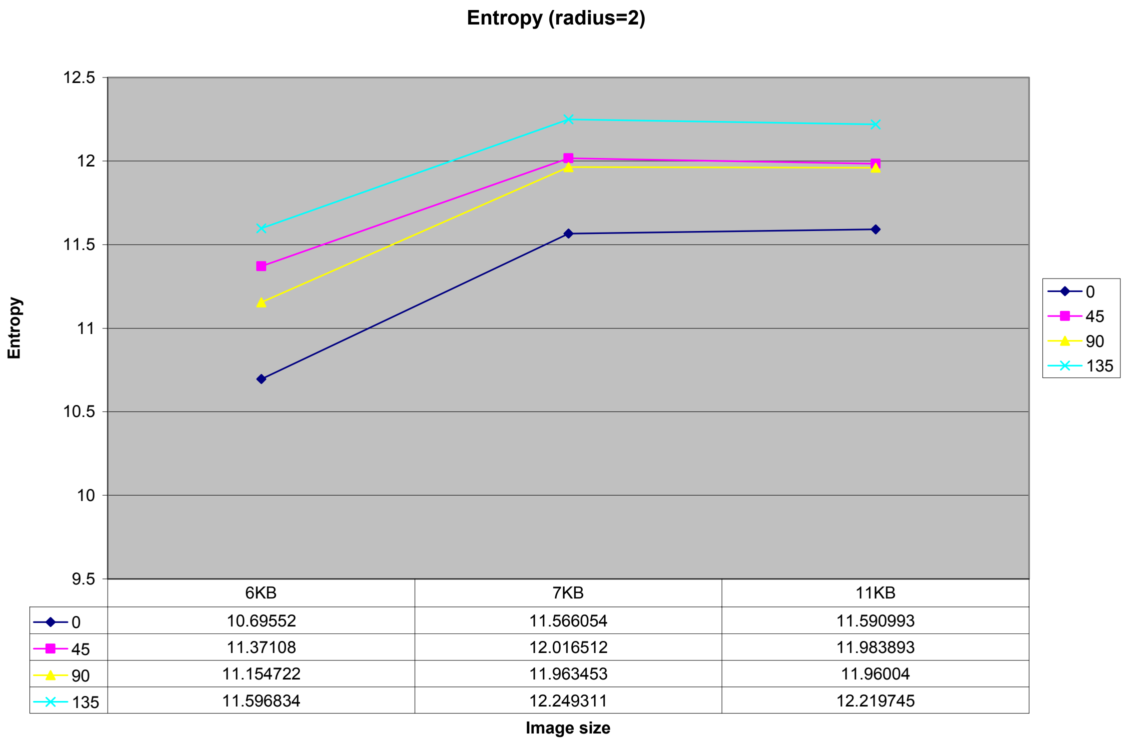
	6KB	7KB	11KB
◆ 1	11.005651	11.726671	11.693091
■ 2	11.596834	12.249311	12.219745
▲ 4	12.273067	12.825015	12.799867
× 8	12.985645	13.45011	13.384607

Figure 23d. Entropy angle=135



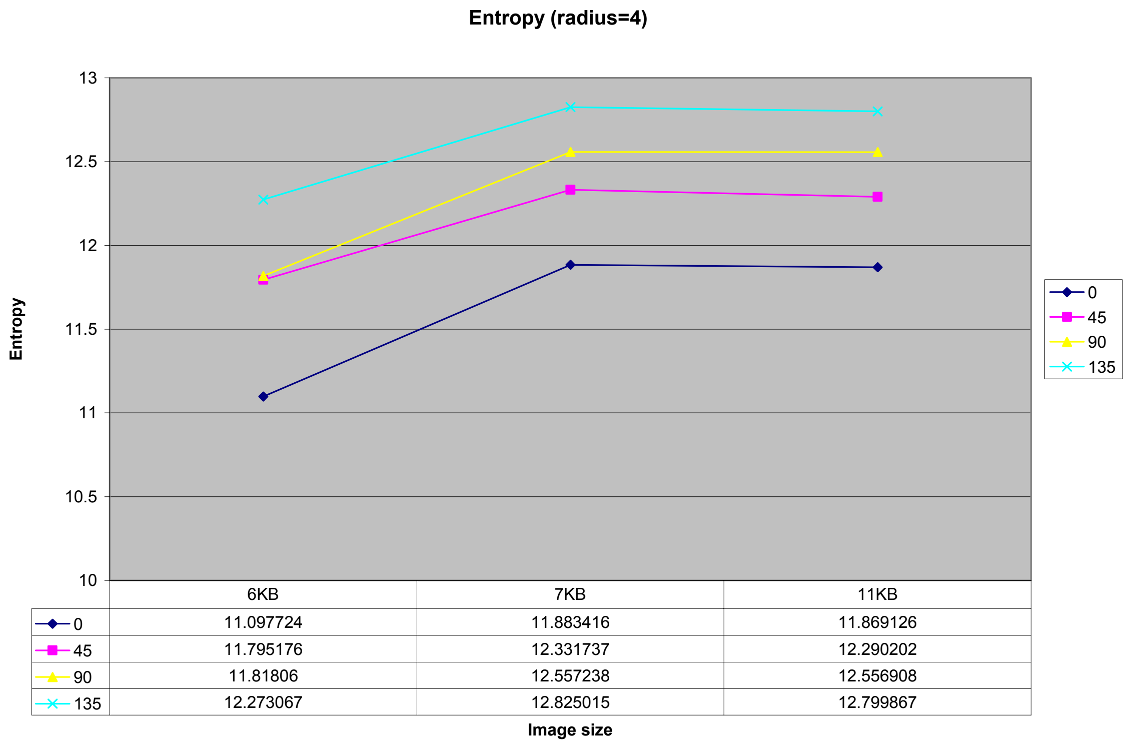
	6KB	7KB	11KB
0	10.261771	11.079689	11.190067
45	10.88752	11.606898	11.579838
90	10.541404	11.348283	11.369725
135	11.005651	11.726671	11.693091

Figure 23e. Entropy radius=1



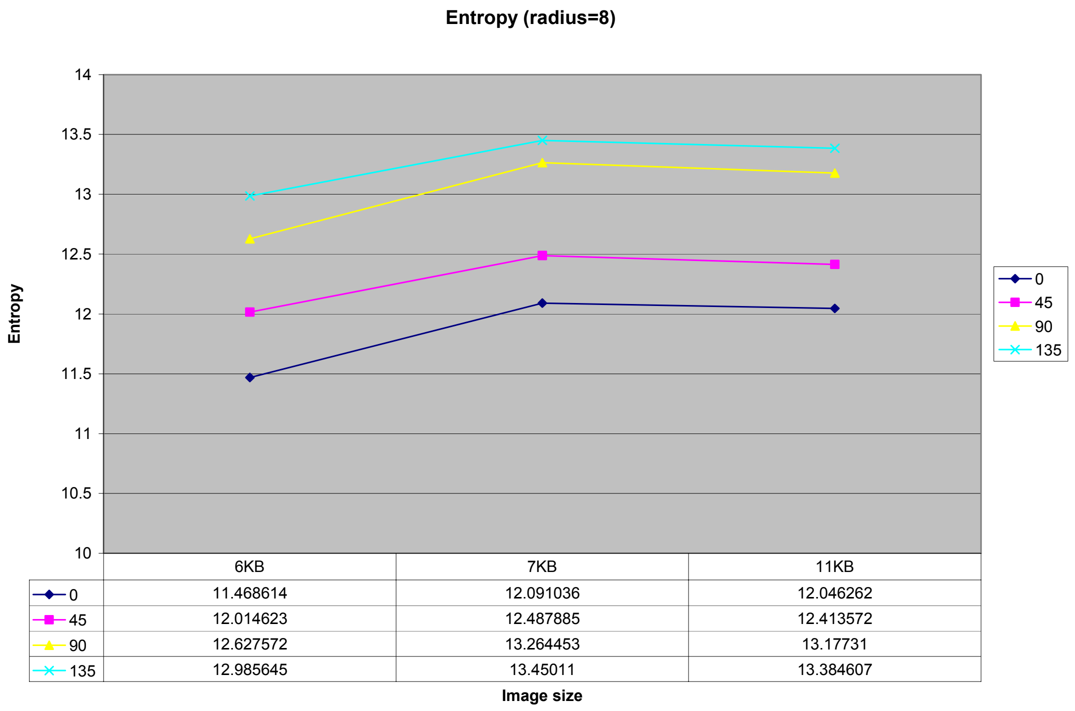
	6KB	7KB	11KB
0	10.69552	11.566054	11.590993
45	11.37108	12.016512	11.983893
90	11.154722	11.963453	11.96004
135	11.596834	12.249311	12.219745

Figure 23f. Entropy radius=2



	6KB	7KB	11KB
0	11.097724	11.883416	11.869126
45	11.795176	12.331737	12.290202
90	11.81806	12.557238	12.556908
135	12.273067	12.825015	12.799867

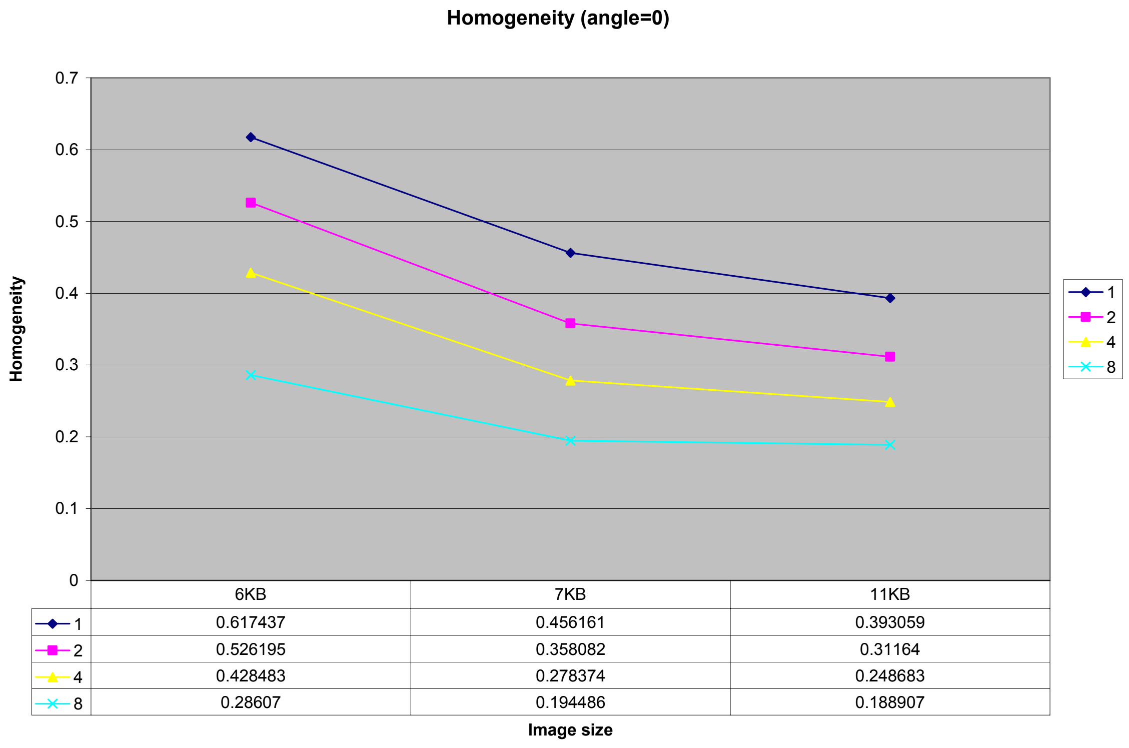
Figure 23g. Entropy radius=4



	6KB	7KB	11KB
0	11.468614	12.091036	12.046262
45	12.014623	12.487885	12.413572
90	12.627572	13.264453	13.17731
135	12.985645	13.45011	13.384607

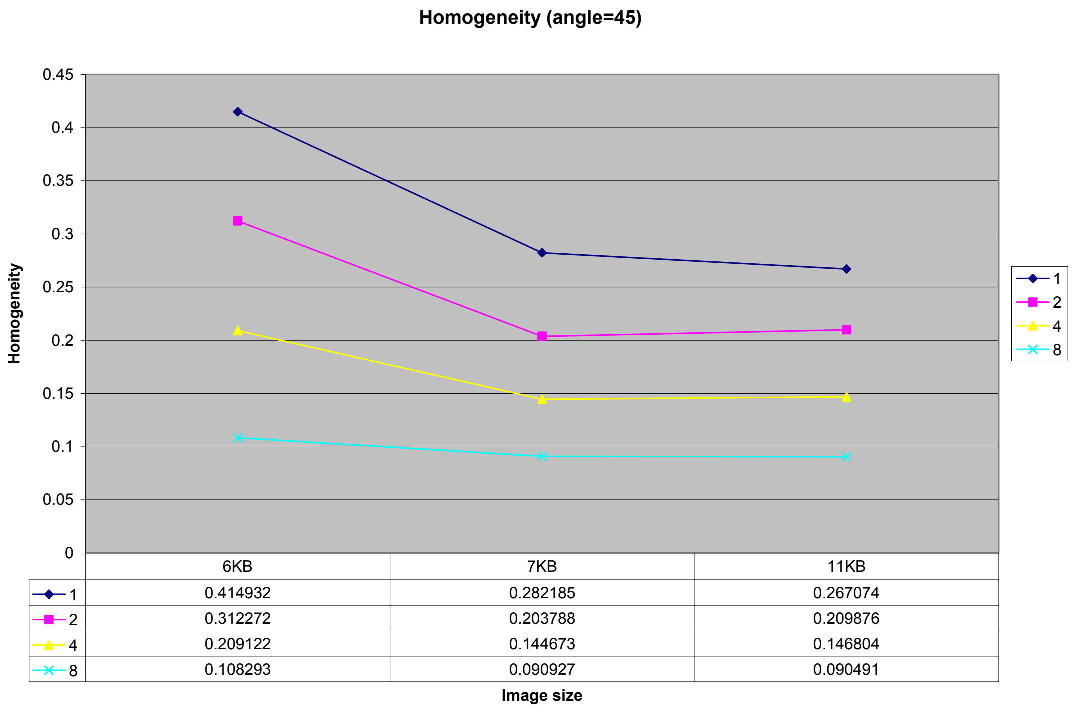
Figure 23h. Entropy radius=8

The plots for homogeneity feature:



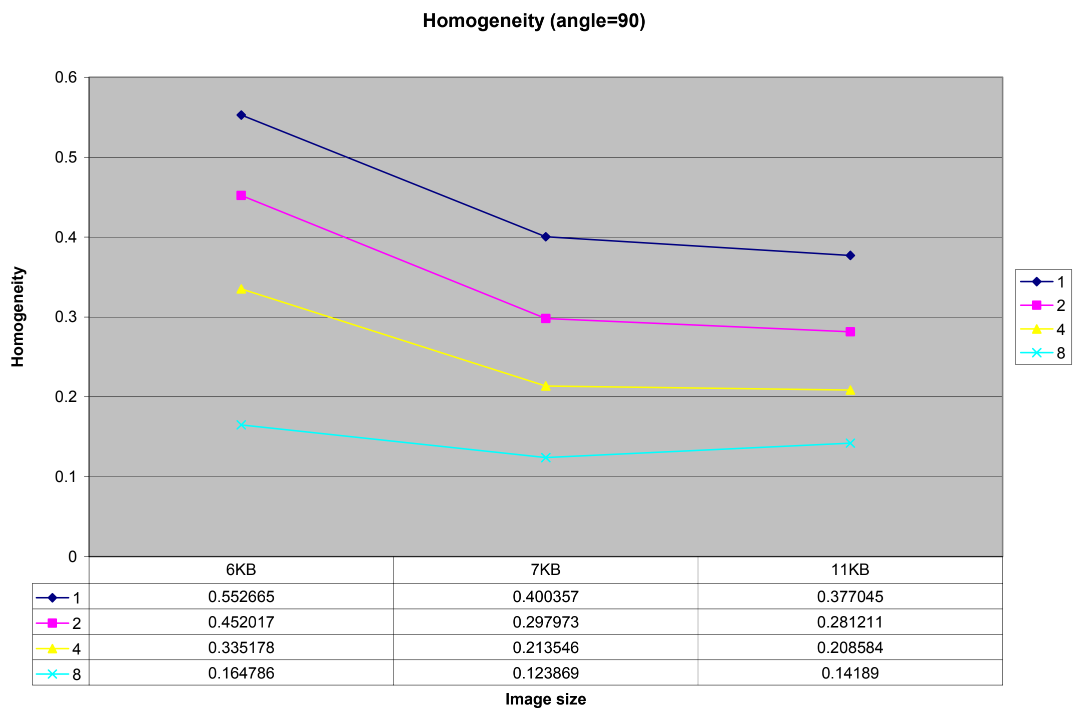
	6KB	7KB	11KB
1	0.617437	0.456161	0.393059
2	0.526195	0.358082	0.31164
4	0.428483	0.278374	0.248683
8	0.28607	0.194486	0.188907

Figure 24a. Homogeneity angle=0



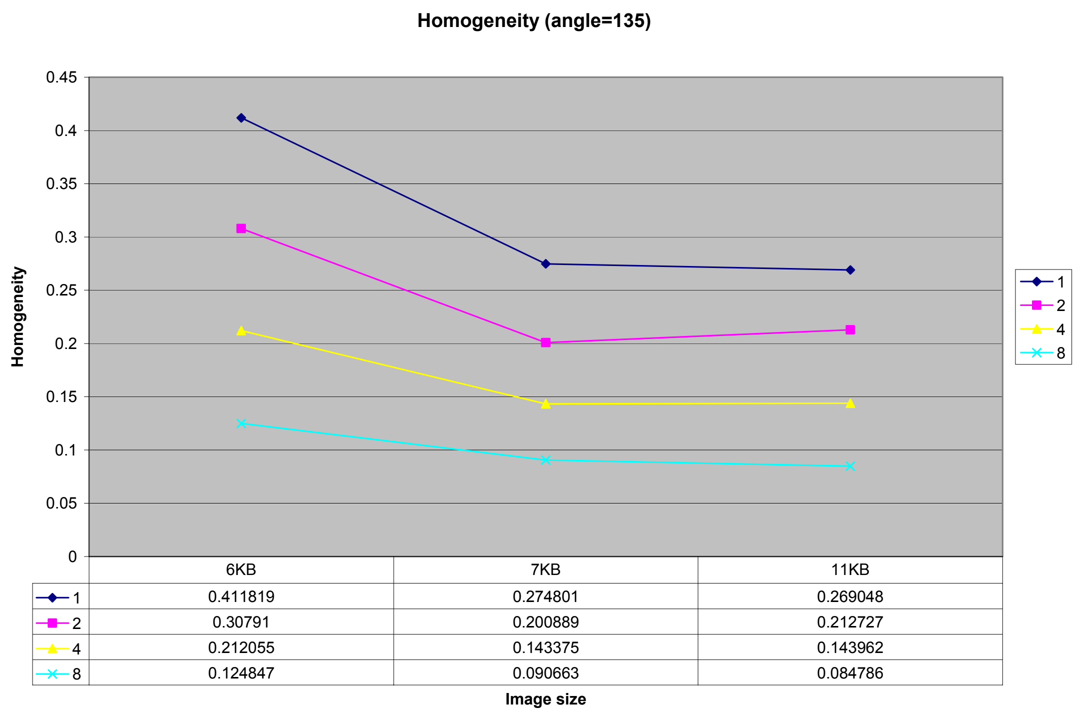
	6KB	7KB	11KB
1	0.414932	0.282185	0.267074
2	0.312272	0.203788	0.209876
4	0.209122	0.144673	0.146804
8	0.108293	0.090927	0.090491

Figure 24b. Homogeneity angle=45



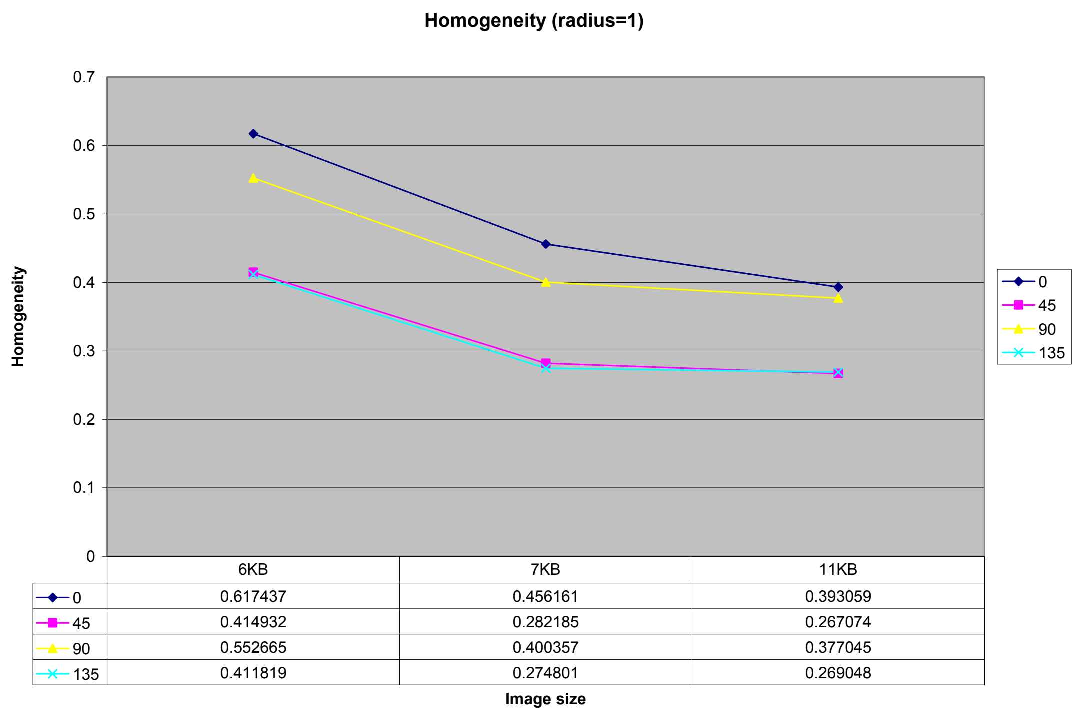
	6KB	7KB	11KB
◆ 1	0.552665	0.400357	0.377045
■ 2	0.452017	0.297973	0.281211
▲ 4	0.335178	0.213546	0.208584
✕ 8	0.164786	0.123869	0.14189

Figure 24c. Homogeneity angle=90



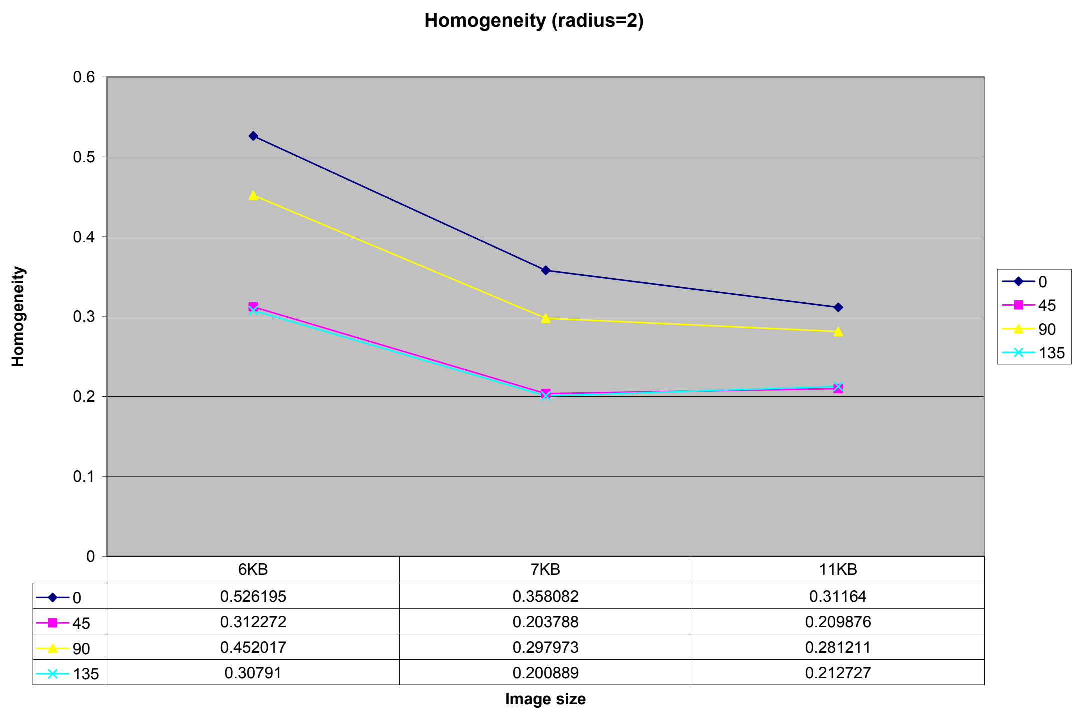
	6KB	7KB	11KB
◆ 1	0.411819	0.274801	0.269048
■ 2	0.30791	0.200889	0.212727
▲ 4	0.212055	0.143375	0.143962
✕ 8	0.124847	0.090663	0.084786

Figure 24d. Homogeneity angle=135



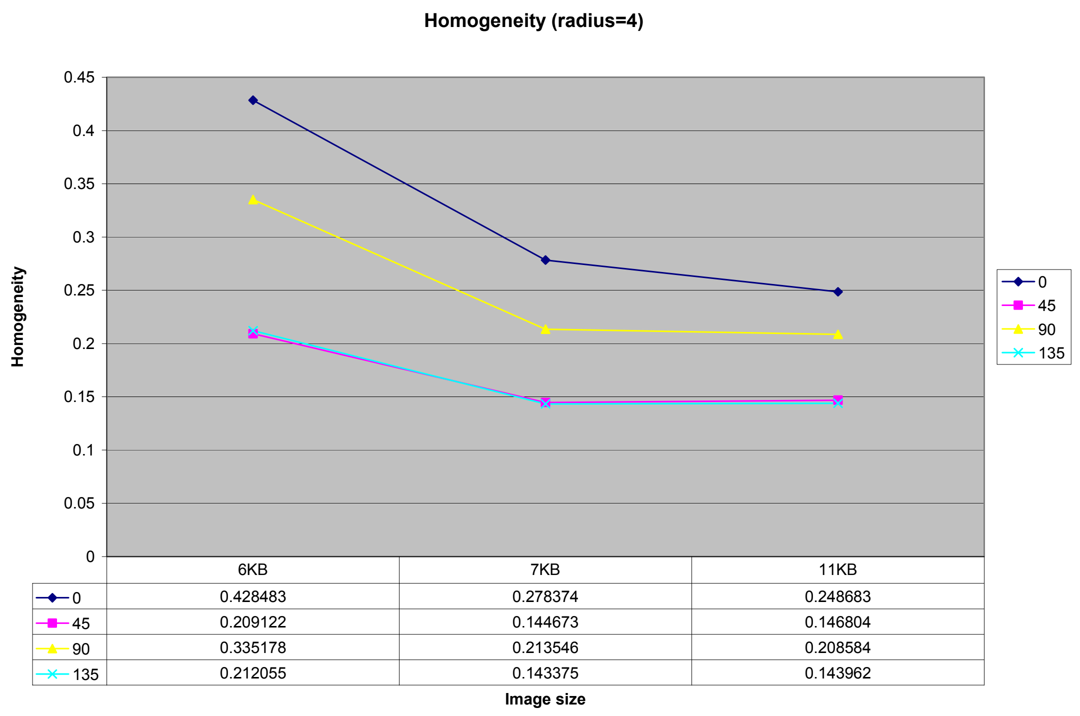
	6KB	7KB	11KB
◆ 0	0.617437	0.456161	0.393059
■ 45	0.414932	0.282185	0.267074
▲ 90	0.552665	0.400357	0.377045
✕ 135	0.411819	0.274801	0.269048

Figure 24e. Homogeneity radius=1



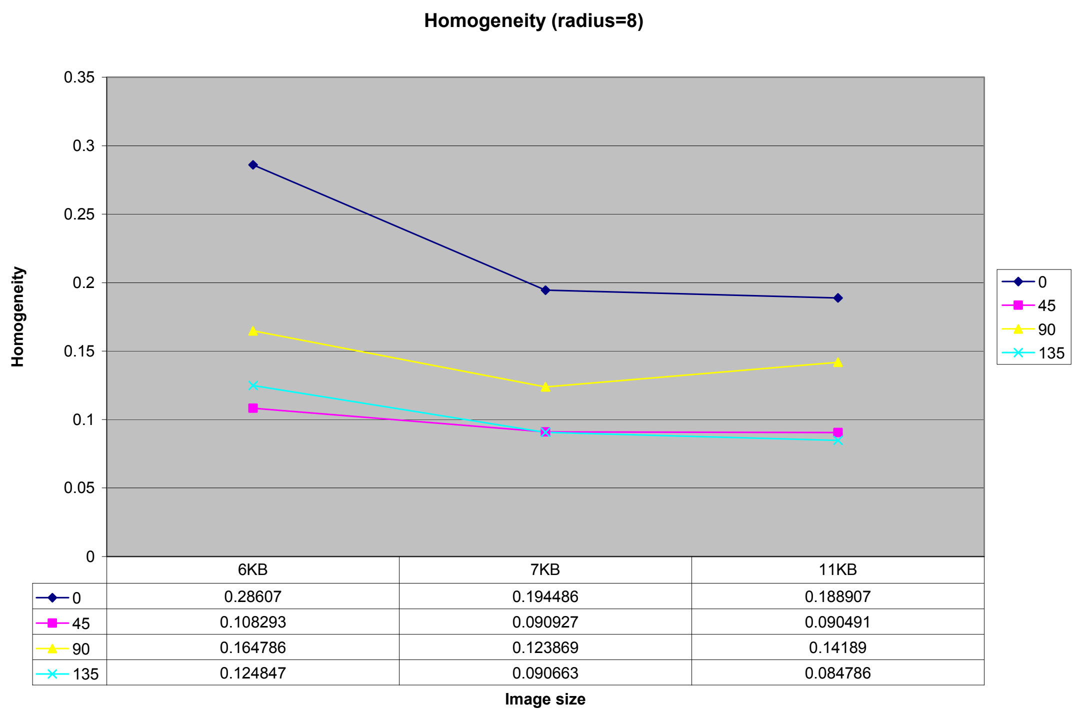
	6KB	7KB	11KB
◆ 0	0.526195	0.358082	0.31164
■ 45	0.312272	0.203788	0.209876
▲ 90	0.452017	0.297973	0.281211
✕ 135	0.30791	0.200889	0.212727

Figure 24f. Homogeneity radius=2



	6KB	7KB	11KB
0	0.428483	0.278374	0.248683
45	0.209122	0.144673	0.146804
90	0.335178	0.213546	0.208584
135	0.212055	0.143375	0.143962

Figure 24g. Homogeneity radius=4



	6KB	7KB	11KB
0	0.28607	0.194486	0.188907
45	0.108293	0.090927	0.090491
90	0.164786	0.123869	0.14189
135	0.124847	0.090663	0.084786

Figure 24h. Homogeneity radius=8

The analysis of the results showed that the curves of the textural features did not follow a specific trend for all the combinations of radius and angle. For instance, the contrast and entropy curves change the sign of first derivative for radius equal to 8 and angle equal to 0°.

Similar exhaustive analysis was performed on the second dataset, CERTIFICATE, representing simple text. Quality levels 1 through 5 were used. A total of 80 runs were performed to analyze the features.

Dataset 3 CERTIFICATE images:

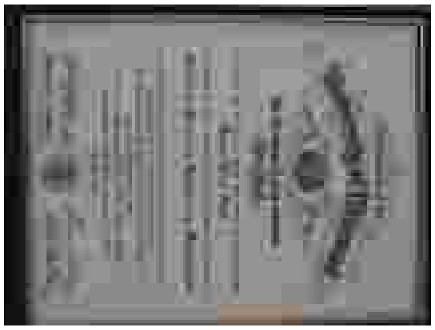


Figure 25a. CERTIFICATE quality level 1

(6KB)

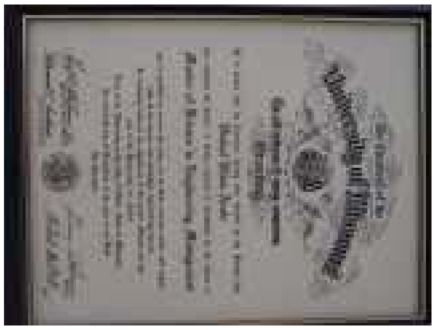


Figure 25c. CERTIFICATE quality level 3

(8KB)

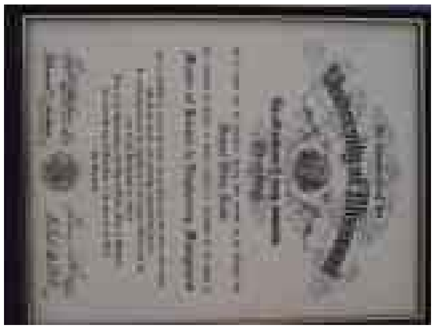


Figure 25b. CERTIFICATE quality level 2

(7KB)

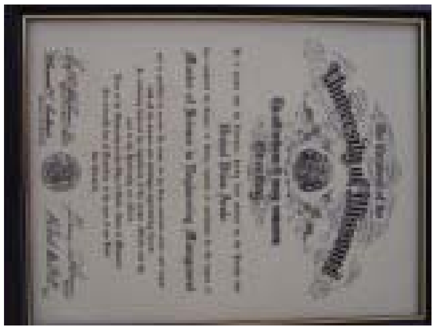


Figure 25d. CERTIFICATE quality level 5

(11KB)

Figure 25e. CERTIFICATE quality level 4 (9KB)

These were the results obtained:

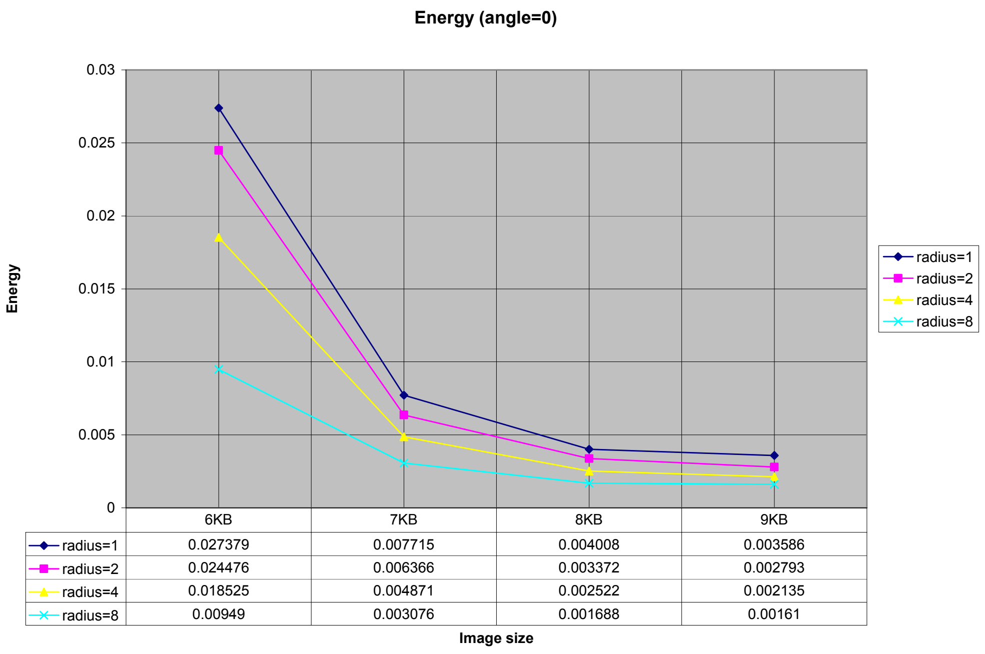
Image size	Radius	Angle	Energy	Contrast	Entropy	Homogeneity
6KB	1	0	0.027379	287.2065	9.783876	0.510128
7KB	1	0	0.007715	335.0582	10.78946	0.464195
8KB	1	0	0.004008	421.4891	11.07274	0.420977
9KB	1	0	0.003586	430.4905	11.05674	0.410885
6KB	1	45	0.021091	463.181	10.28181	0.2892
7KB	1	45	0.005217	637.521	11.21314	0.244366
8KB	1	45	0.002328	746.0709	11.45729	0.234125
9KB	1	45	0.002308	754.4606	11.41144	0.241451
6KB	1	90	0.032086	443.7578	9.230447	0.790542
7KB	1	90	0.007033	596.3367	10.62541	0.451566
8KB	1	90	0.003731	645.5999	10.92591	0.415467
9KB	1	90	0.00331	665.5436	10.98611	0.394092
6KB	1	135	0.021021	475.4983	10.39187	0.285888
7KB	1	135	0.005151	642.326	11.30022	0.247777
8KB	1	135	0.002385	758.9157	11.55252	0.235528
9KB	1	135	0.002297	762.8987	11.51412	0.239714

6KB	2	0	0.024476	694.4235	10.06599	0.441217
7KB	2	0	0.006366	764.2224	11.12403	0.387998
8KB	2	0	0.003372	828.6582	11.33644	0.354863
9KB	2	0	0.002793	828.5843	11.33926	0.335974
6KB	2	45	0.016733	1052.688	10.62079	0.221683
7KB	2	45	0.004043	1150.561	11.45675	0.189758
8KB	2	45	0.001778	1193.276	11.6344	0.180653
9KB	2	45	0.001777	1190.999	11.5753	0.198746
6KB	2	90	0.027324	921.1234	9.801897	0.678564
7KB	2	90	0.00567	1027.203	11.14337	0.347361
8KB	2	90	0.002697	1064.571	11.41439	0.321483
9KB	2	90	0.002427	1094.404	11.42713	0.305335
6KB	2	135	0.016377	1082.483	10.83886	0.21717
7KB	2	135	0.003914	1181.906	11.64734	0.18922
8KB	2	135	0.001773	1230.544	11.83341	0.183591
9KB	2	135	0.001846	1233.883	11.77956	0.19612
6KB	4	0	0.018525	1017.375	10.34188	0.381935
7KB	4	0	0.004871	1049.382	11.29839	0.326618
8KB	4	0	0.002522	1118.893	11.4781	0.292997
9KB	4	0	0.002135	1117.56	11.48222	0.268608
6KB	4	45	0.010575	1526.242	10.89026	0.178823
7KB	4	45	0.002908	1617.764	11.56614	0.166107
8KB	4	45	0.001354	1671.662	11.70695	0.144582
9KB	4	45	0.00143	1678.077	11.6497	0.157295
6KB	4	90	0.019759	1531.76	10.47997	0.551872
7KB	4	90	0.004195	1639.605	11.56554	0.29527
8KB	4	90	0.00205	1653.875	11.78932	0.258164
9KB	4	90	0.002004	1691.017	11.7834	0.234682
6KB	4	135	0.009319	1603.564	11.31929	0.172967
7KB	4	135	0.002503	1695.924	11.96369	0.155629
8KB	4	135	0.001362	1754.898	12.12374	0.143635
9KB	4	135	0.001479	1773.781	12.05537	0.154013
6KB	8	0	0.00949	1561.06	10.69729	0.283164
7KB	8	0	0.003076	1549.745	11.38525	0.226056
8KB	8	0	0.001688	1627.402	11.51916	0.195653
9KB	8	0	0.00161	1635.994	11.47102	0.205251

6KB	8	45	0.005472	2504.095	10.96345	0.135835
7KB	8	45	0.001925	2553.321	11.53527	0.131247
8KB	8	45	0.001093	2640.996	11.65373	0.106359

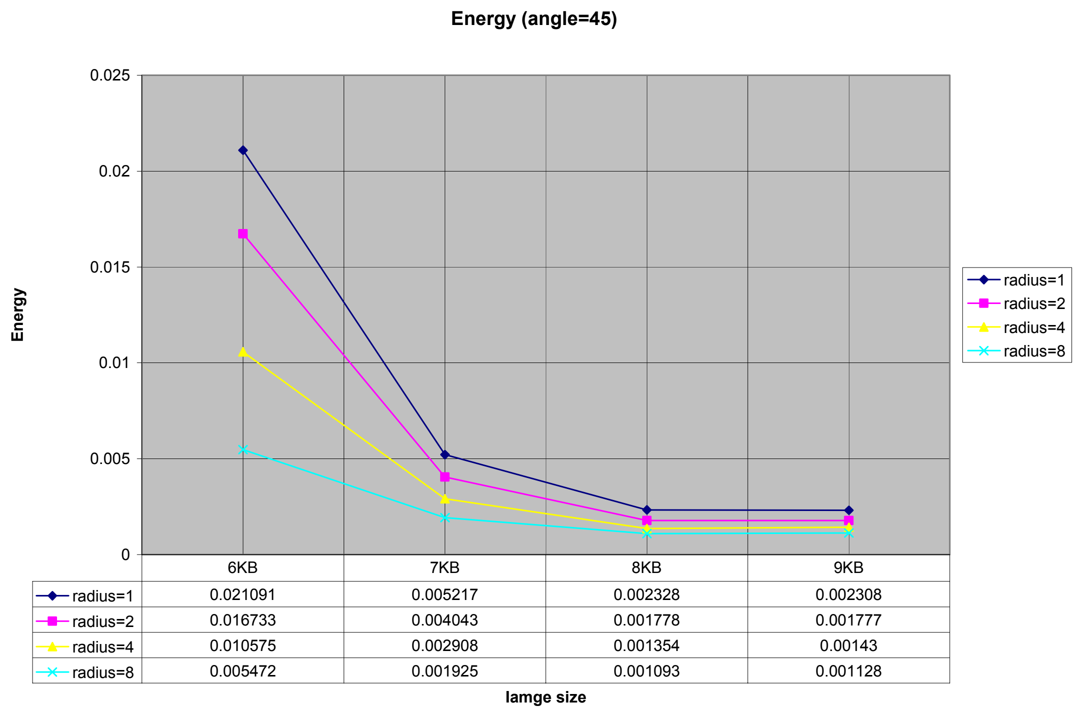
Table 3. Dataset 3 CERTIFICATE image

The corresponding plots are listed below. The sequence follows eight graphs for energy, contrast, entropy and homogeneity respectively, which is similar to that of FACE image:



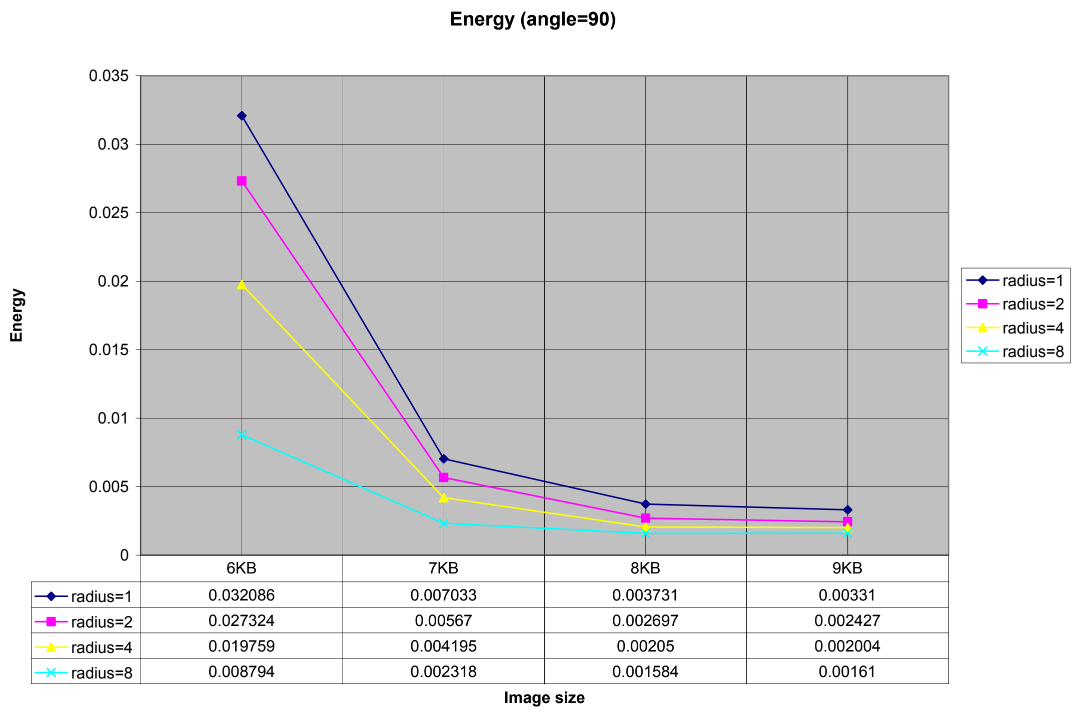
	6KB	7KB	8KB	9KB
radius=1	0.027379	0.007715	0.004008	0.003586
radius=2	0.024476	0.006366	0.003372	0.002793
radius=4	0.018525	0.004871	0.002522	0.002135
radius=8	0.00949	0.003076	0.001688	0.00161

Figure 26a. Energy angle=0



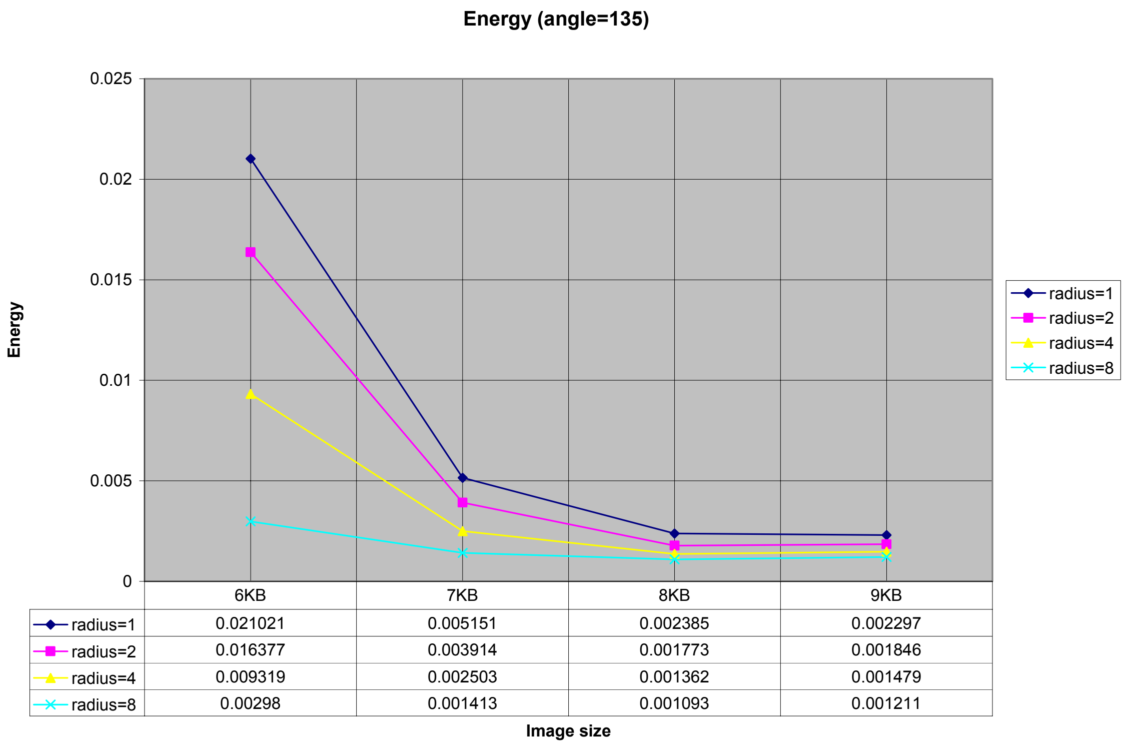
	6KB	7KB	8KB	9KB
◆ radius=1	0.021091	0.005217	0.002328	0.002308
■ radius=2	0.016733	0.004043	0.001778	0.001777
▲ radius=4	0.010575	0.002908	0.001354	0.00143
✕ radius=8	0.005472	0.001925	0.001093	0.001128

Figure 26b. Energy angle=45



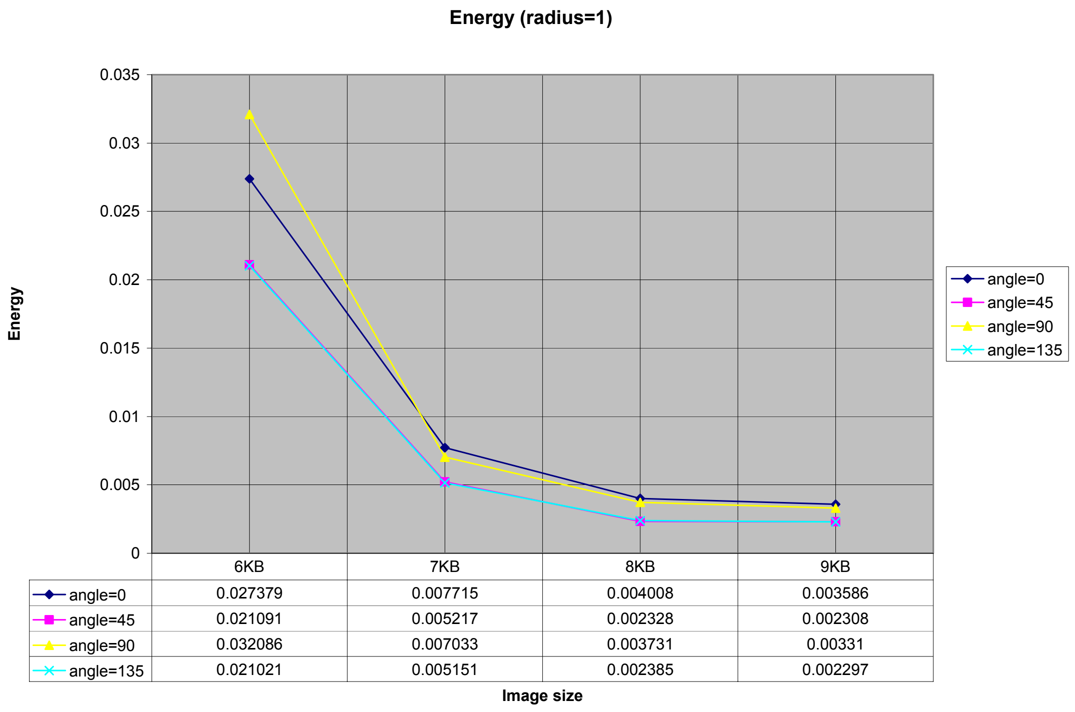
	6KB	7KB	8KB	9KB
◆ radius=1	0.032086	0.007033	0.003731	0.00331
■ radius=2	0.027324	0.00567	0.002697	0.002427
▲ radius=4	0.019759	0.004195	0.00205	0.002004
✕ radius=8	0.008794	0.002318	0.001584	0.00161

Figure 26c. Energy angle=90



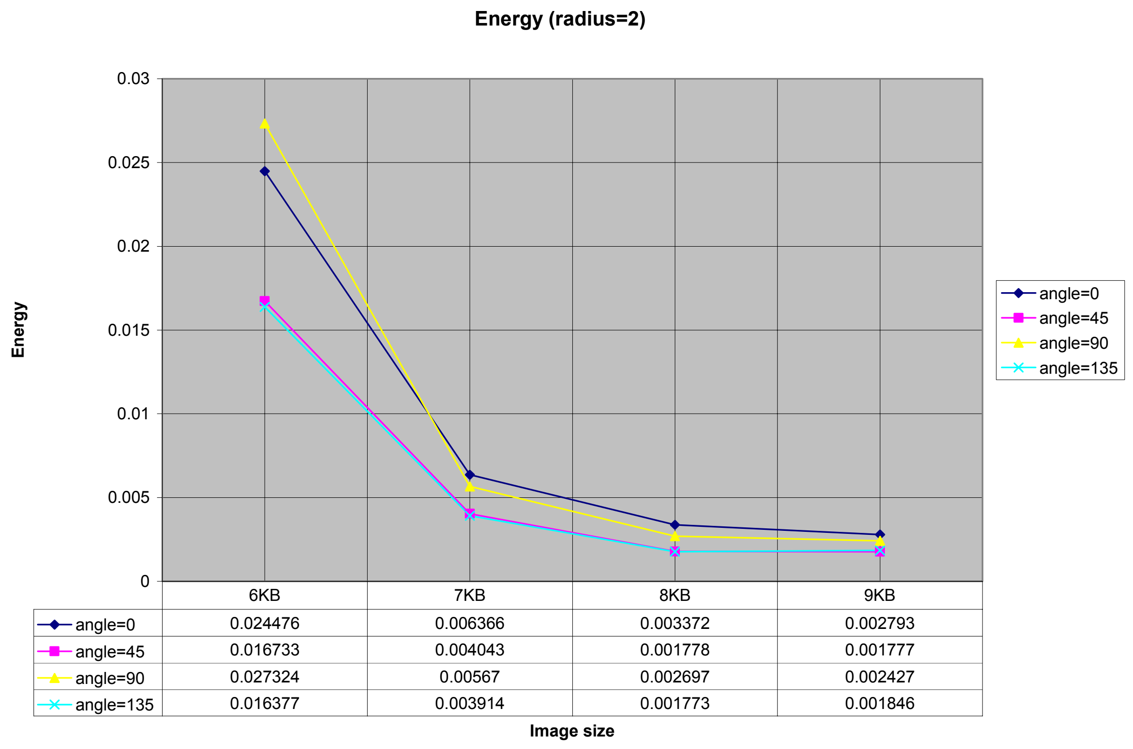
	6KB	7KB	8KB	9KB
radius=1	0.021021	0.005151	0.002385	0.002297
radius=2	0.016377	0.003914	0.001773	0.001846
radius=4	0.009319	0.002503	0.001362	0.001479
radius=8	0.00298	0.001413	0.001093	0.001211

Figure 26d. Energy angle=135



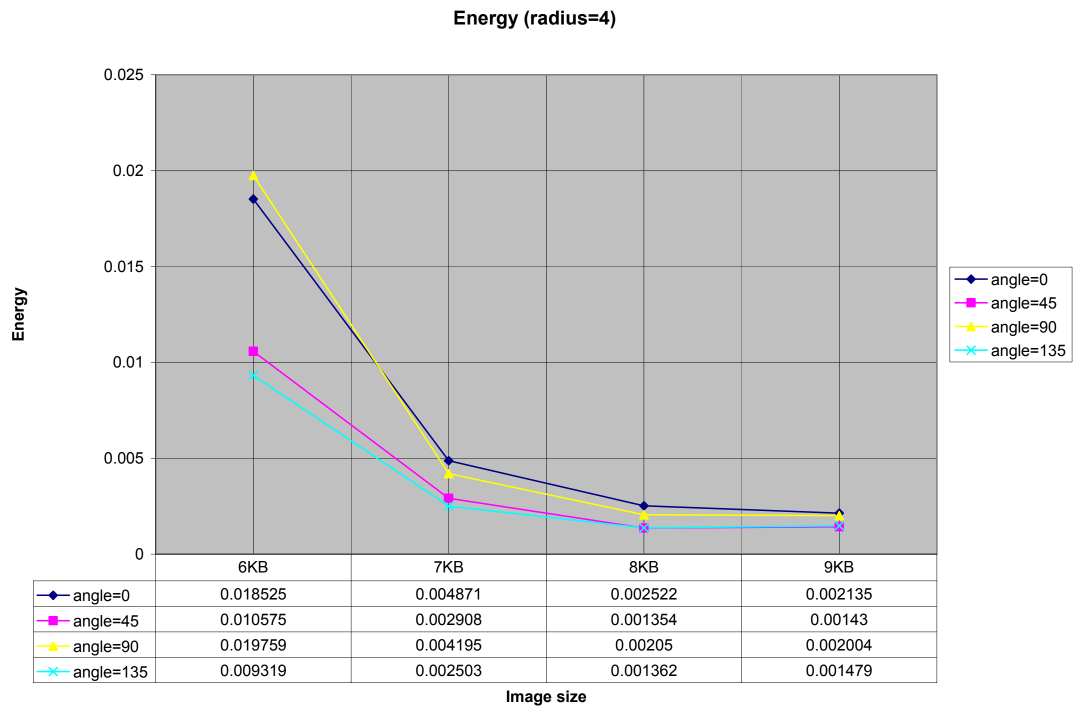
	6KB	7KB	8KB	9KB
angle=0	0.027379	0.007715	0.004008	0.003586
angle=45	0.021091	0.005217	0.002328	0.002308
angle=90	0.032086	0.007033	0.003731	0.00331
angle=135	0.021021	0.005151	0.002385	0.002297

Figure 26e. Energy radius=1



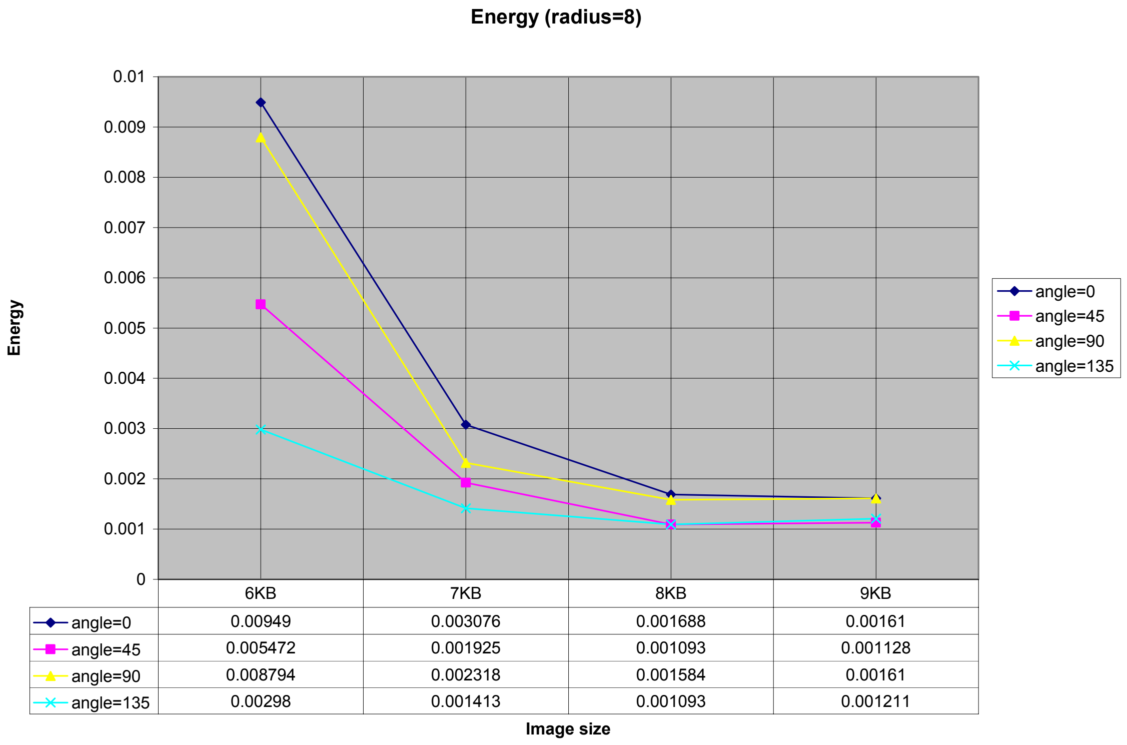
	6KB	7KB	8KB	9KB
angle=0	0.024476	0.006366	0.003372	0.002793
angle=45	0.016733	0.004043	0.001778	0.001777
angle=90	0.027324	0.00567	0.002697	0.002427
angle=135	0.016377	0.003914	0.001773	0.001846

Figure 26f. Energy radius=2



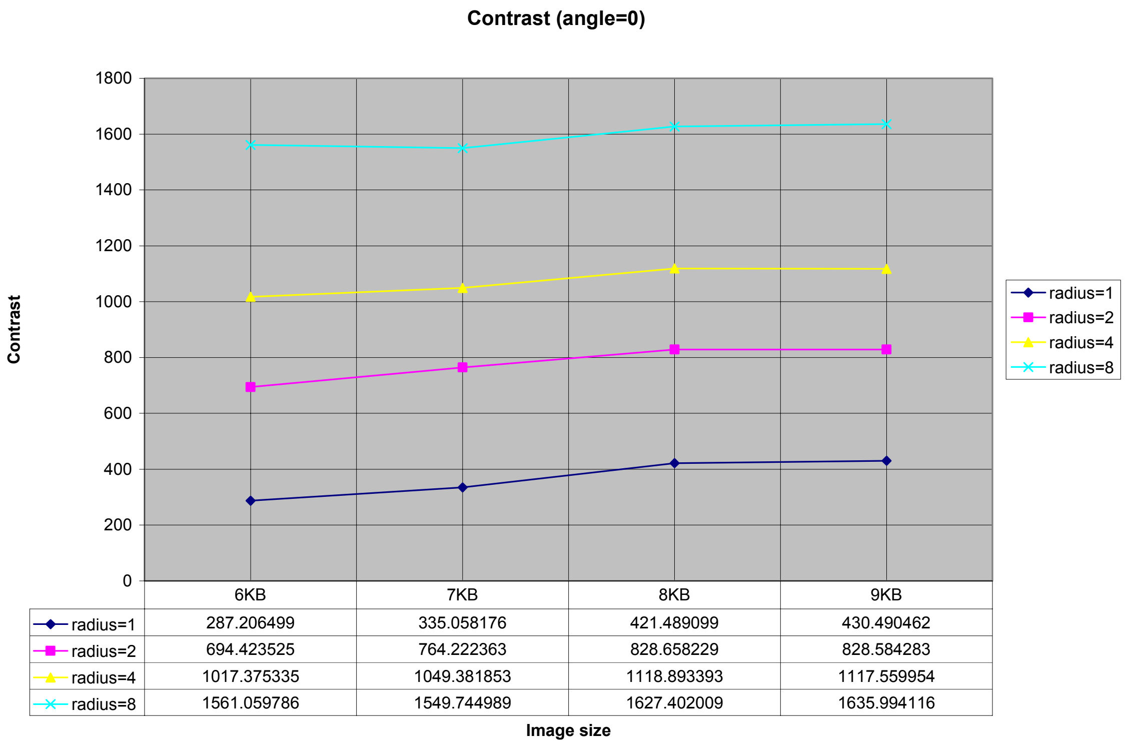
	6KB	7KB	8KB	9KB
angle=0	0.018525	0.004871	0.002522	0.002135
angle=45	0.010575	0.002908	0.001354	0.00143
angle=90	0.019759	0.004195	0.00205	0.002004
angle=135	0.009319	0.002503	0.001362	0.001479

Figure 26g. Energy radius=4



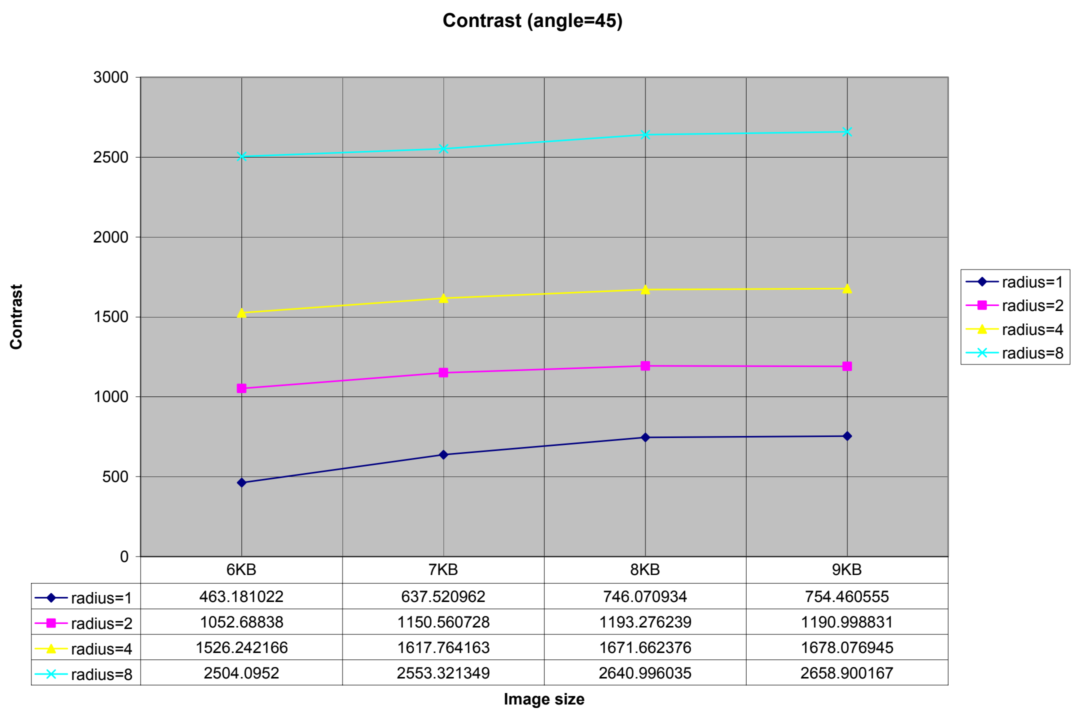
	6KB	7KB	8KB	9KB
◆ angle=0	0.00949	0.003076	0.001688	0.00161
■ angle=45	0.005472	0.001925	0.001093	0.001128
▲ angle=90	0.008794	0.002318	0.001584	0.00161
✕ angle=135	0.00298	0.001413	0.001093	0.001211

Figure 26h. Energy radius=8



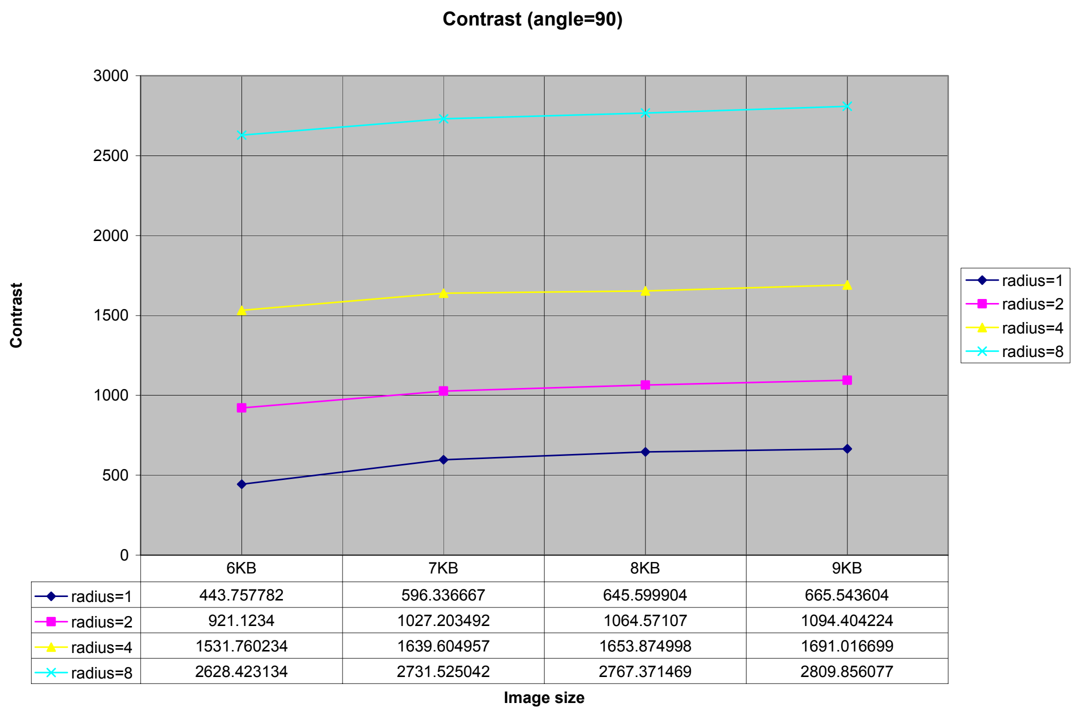
	6KB	7KB	8KB	9KB
◆ radius=1	287.206499	335.058176	421.489099	430.490462
■ radius=2	694.423525	764.222363	828.658229	828.584283
▲ radius=4	1017.375335	1049.381853	1118.893393	1117.559954
✕ radius=8	1561.059786	1549.744989	1627.402009	1635.994116

Figure 27a. Contrast angle=0



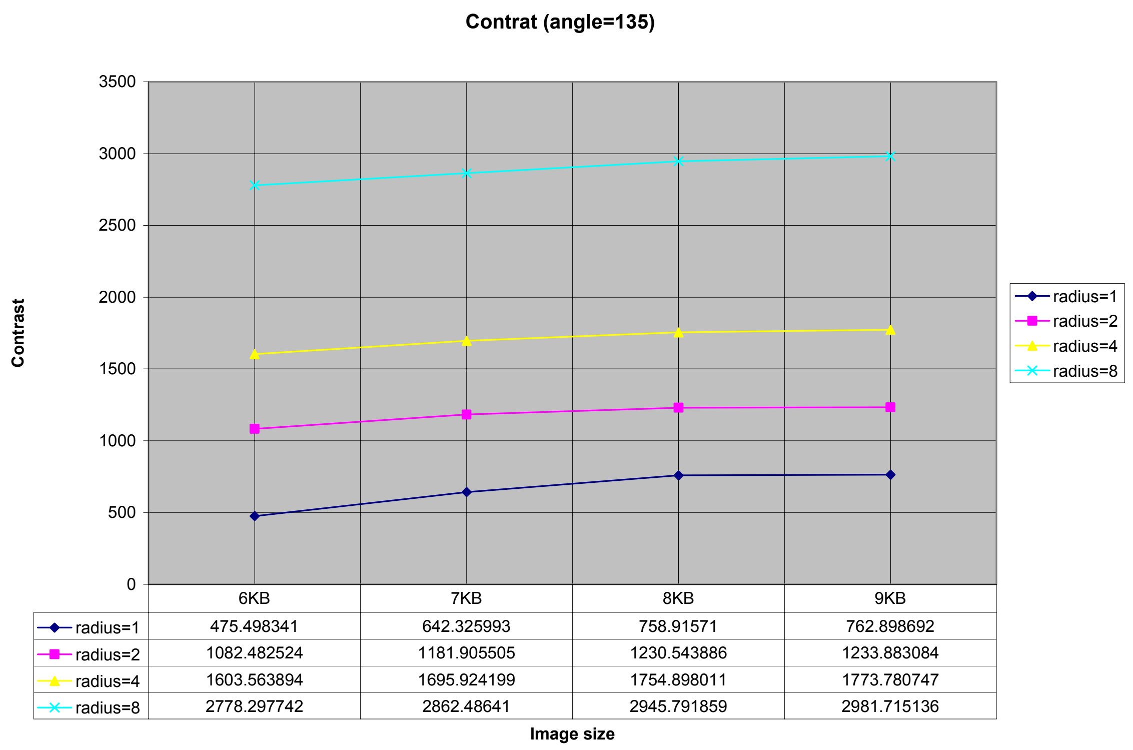
	6KB	7KB	8KB	9KB
radius=1	463.181022	637.520962	746.070934	754.460555
radius=2	1052.68838	1150.560728	1193.276239	1190.998831
radius=4	1526.242166	1617.764163	1671.662376	1678.076945
radius=8	2504.0952	2553.321349	2640.996035	2658.900167

Figure 27b. Contrast angle=45



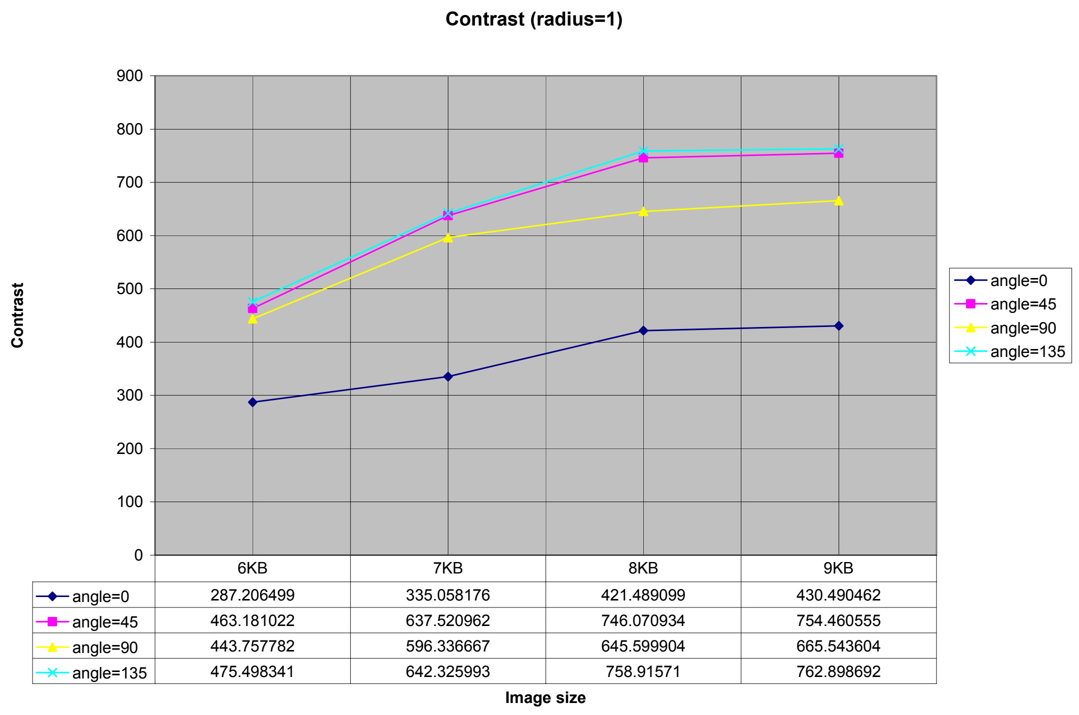
	6KB	7KB	8KB	9KB
radius=1	443.757782	596.336667	645.599904	665.543604
radius=2	921.1234	1027.203492	1064.57107	1094.404224
radius=4	1531.760234	1639.604957	1653.874998	1691.016699
radius=8	2628.423134	2731.525042	2767.371469	2809.856077

Figure 27c. Contrast angle=90



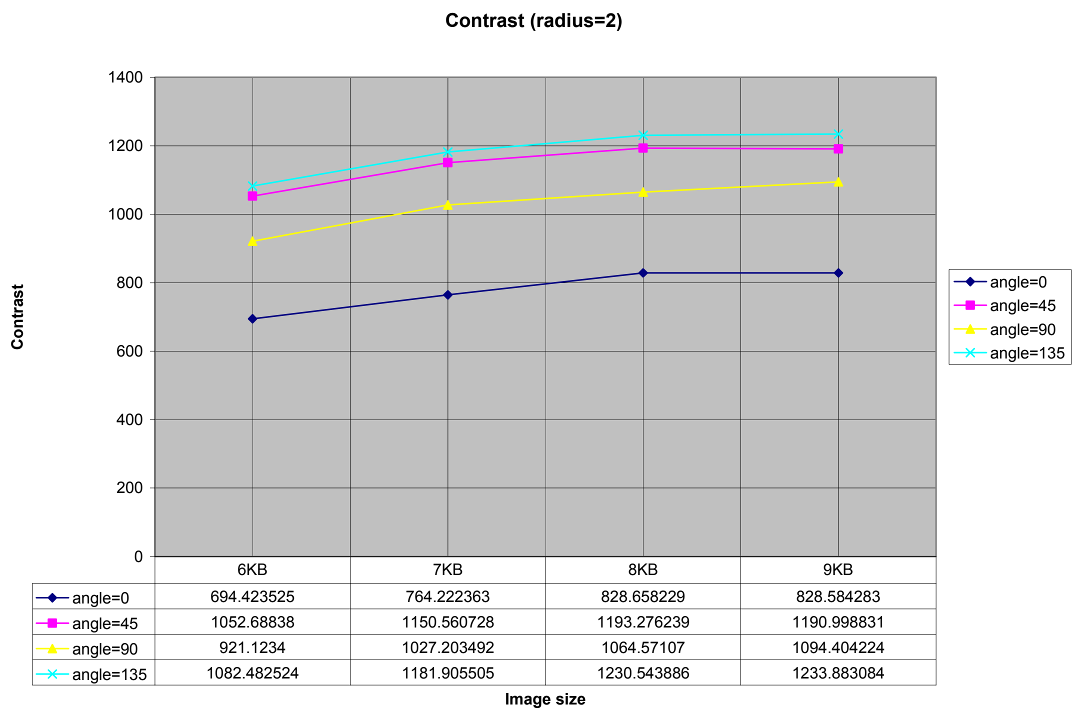
	6KB	7KB	8KB	9KB
radius=1	475.498341	642.325993	758.91571	762.898692
radius=2	1082.482524	1181.905505	1230.543886	1233.883084
radius=4	1603.563894	1695.924199	1754.898011	1773.780747
radius=8	2778.297742	2862.48641	2945.791859	2981.715136

Figure 27d. Contrast angle=135



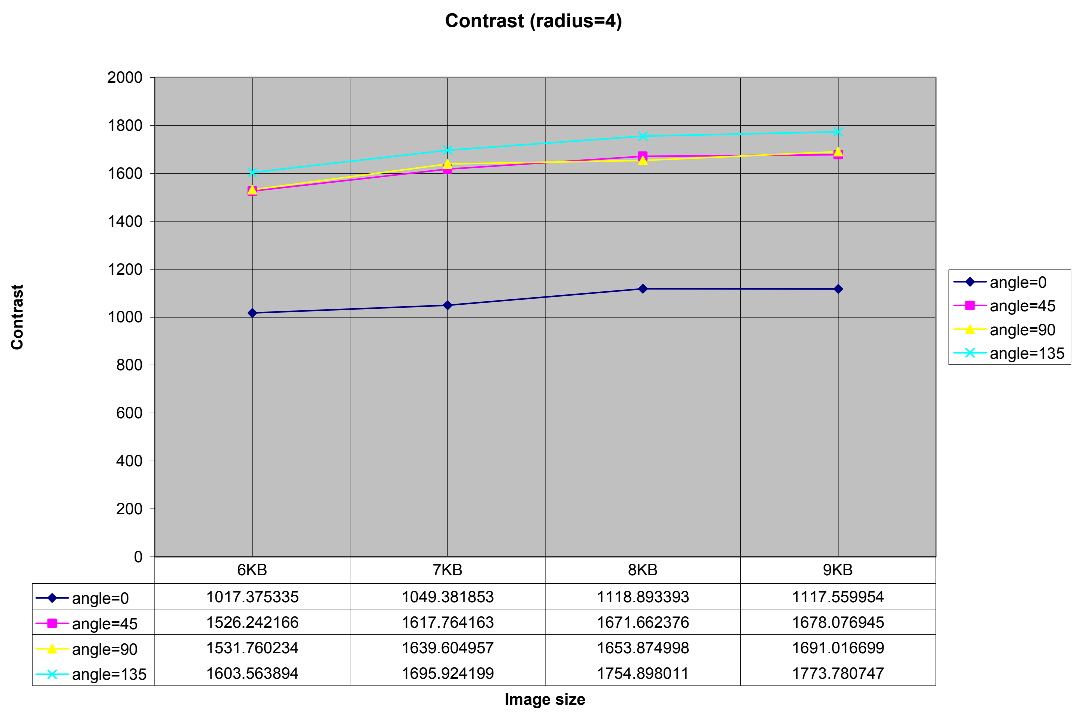
	6KB	7KB	8KB	9KB
angle=0	287.206499	335.058176	421.489099	430.490462
angle=45	463.181022	637.520962	746.070934	754.460555
angle=90	443.757782	596.336667	645.599904	665.543604
angle=135	475.498341	642.325993	758.91571	762.898692

Figure 27e. Contrast radius=1



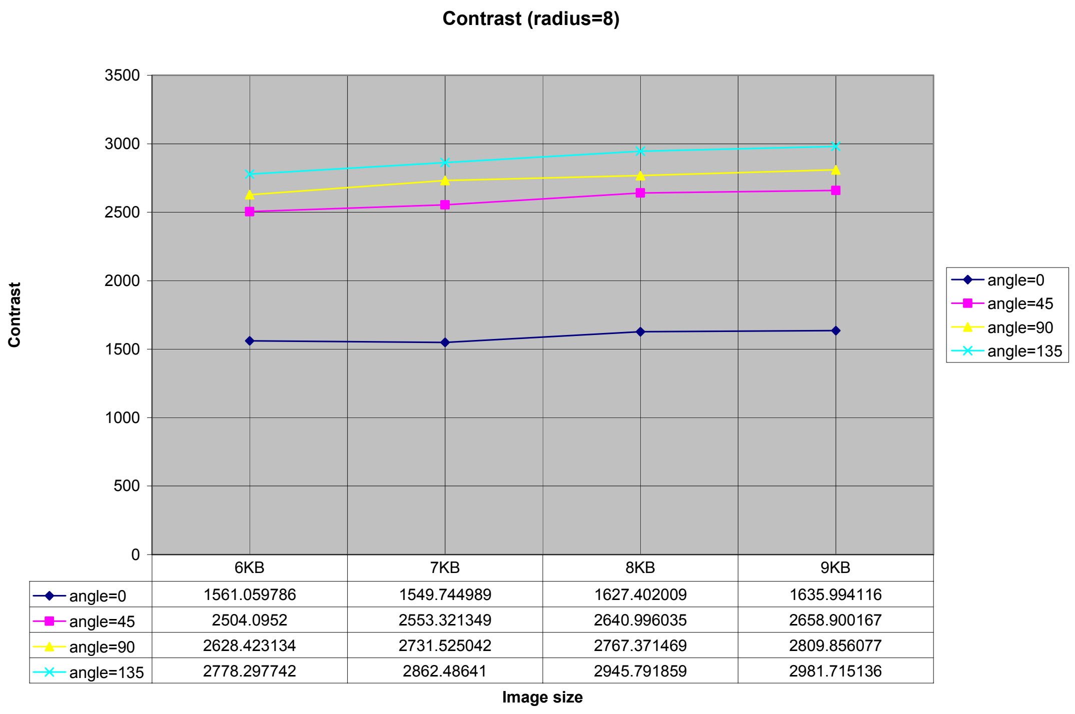
	6KB	7KB	8KB	9KB
angle=0	694.423525	764.222363	828.658229	828.584283
angle=45	1052.68838	1150.560728	1193.276239	1190.998831
angle=90	921.1234	1027.203492	1064.57107	1094.404224
angle=135	1082.482524	1181.905505	1230.543886	1233.883084

Figure 27f. Contrast radius=2



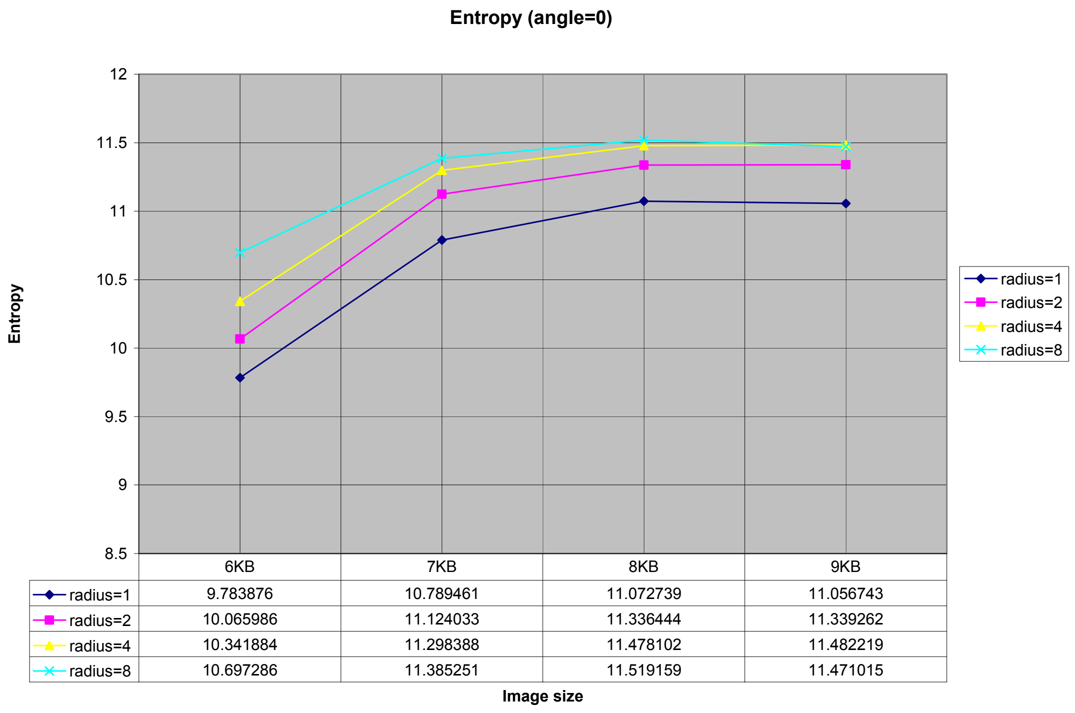
	6KB	7KB	8KB	9KB
angle=0	1017.375335	1049.381853	1118.893393	1117.559954
angle=45	1526.242166	1617.764163	1671.662376	1678.076945
angle=90	1531.760234	1639.604957	1653.874998	1691.016699
angle=135	1603.563894	1695.924199	1754.898011	1773.780747

Figure 27g. Contrast radius=4



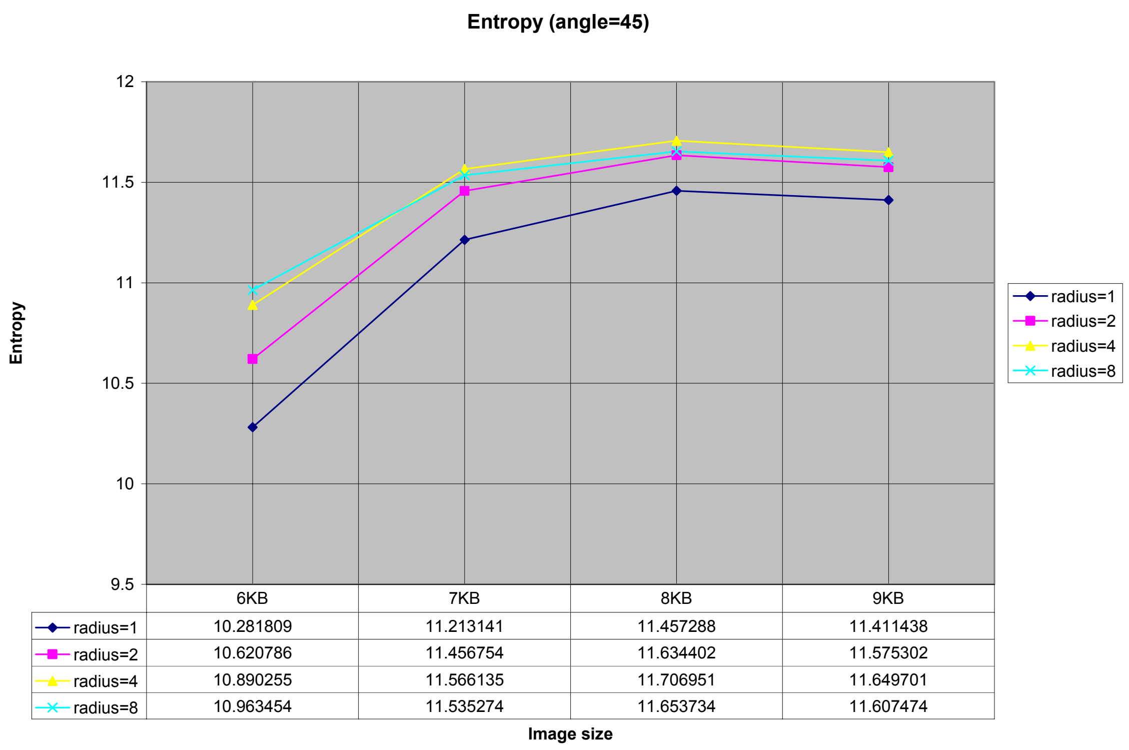
	6KB	7KB	8KB	9KB
angle=0	1561.059786	1549.744989	1627.402009	1635.994116
angle=45	2504.0952	2553.321349	2640.996035	2658.900167
angle=90	2628.423134	2731.525042	2767.371469	2809.856077
angle=135	2778.297742	2862.48641	2945.791859	2981.715136

Figure 27h. Contrast radius=8



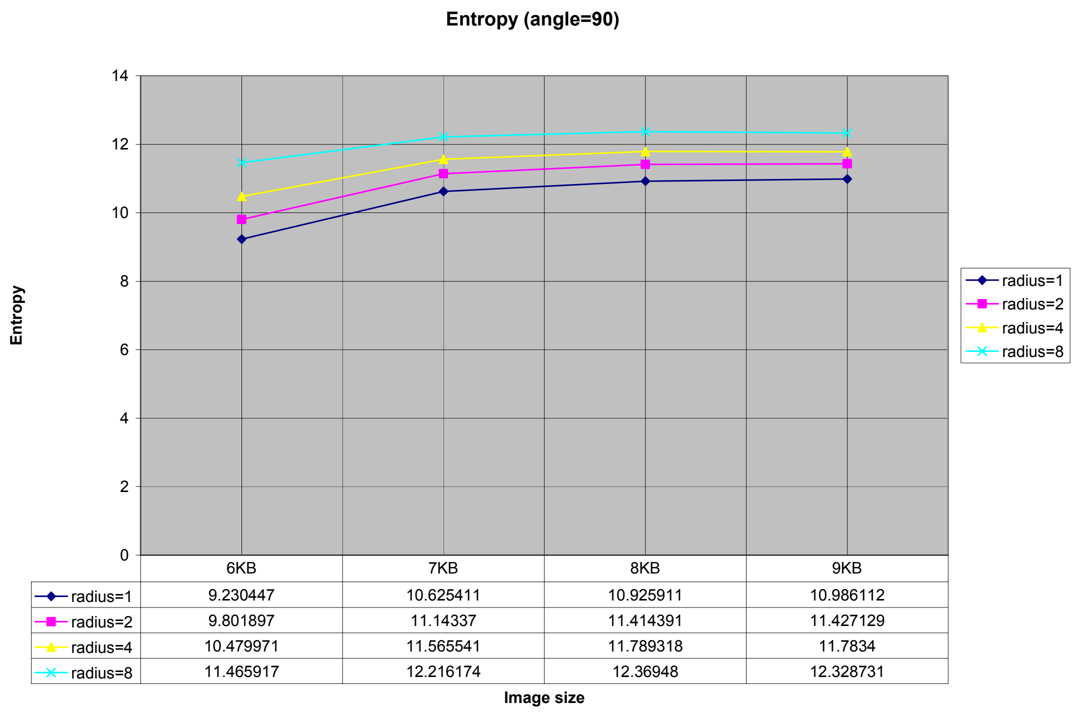
	6KB	7KB	8KB	9KB
radius=1	9.783876	10.789461	11.072739	11.056743
radius=2	10.065986	11.124033	11.336444	11.339262
radius=4	10.341884	11.298388	11.478102	11.482219
radius=8	10.697286	11.385251	11.519159	11.471015

Figure 28a. Entropy angle=0



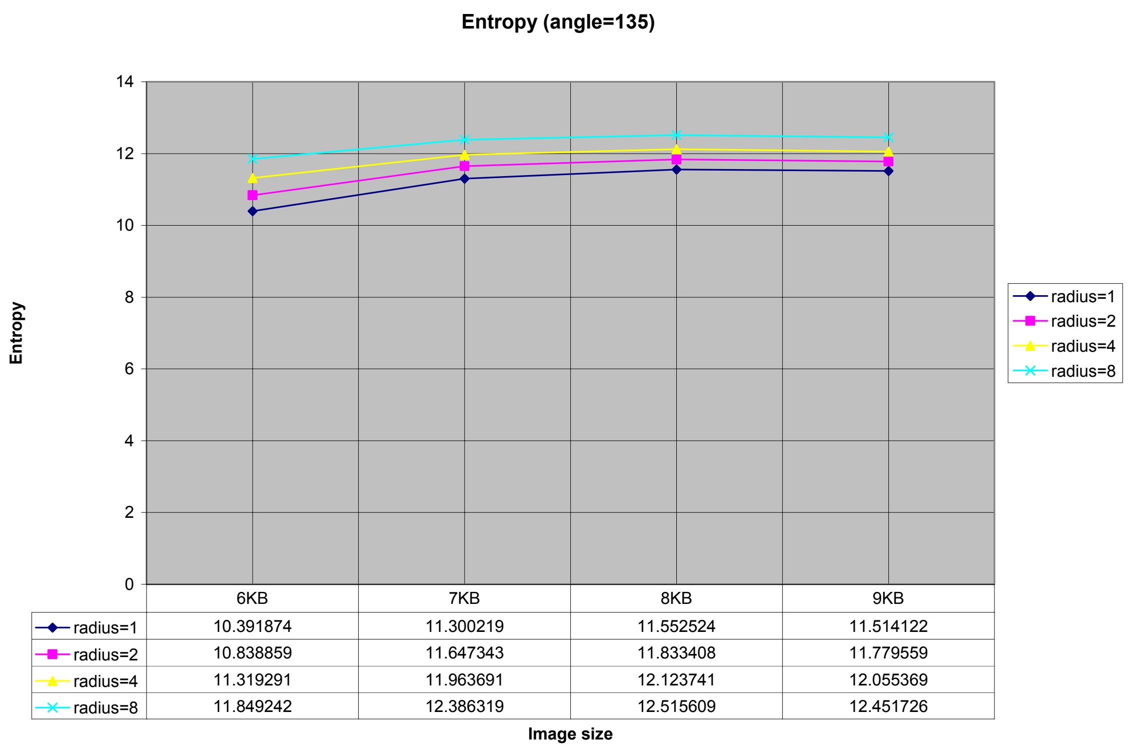
	6KB	7KB	8KB	9KB
radius=1	10.281809	11.213141	11.457288	11.411438
radius=2	10.620786	11.456754	11.634402	11.575302
radius=4	10.890255	11.566135	11.706951	11.649701
radius=8	10.963454	11.535274	11.653734	11.607474

Figure 28b. Entropy angle=45



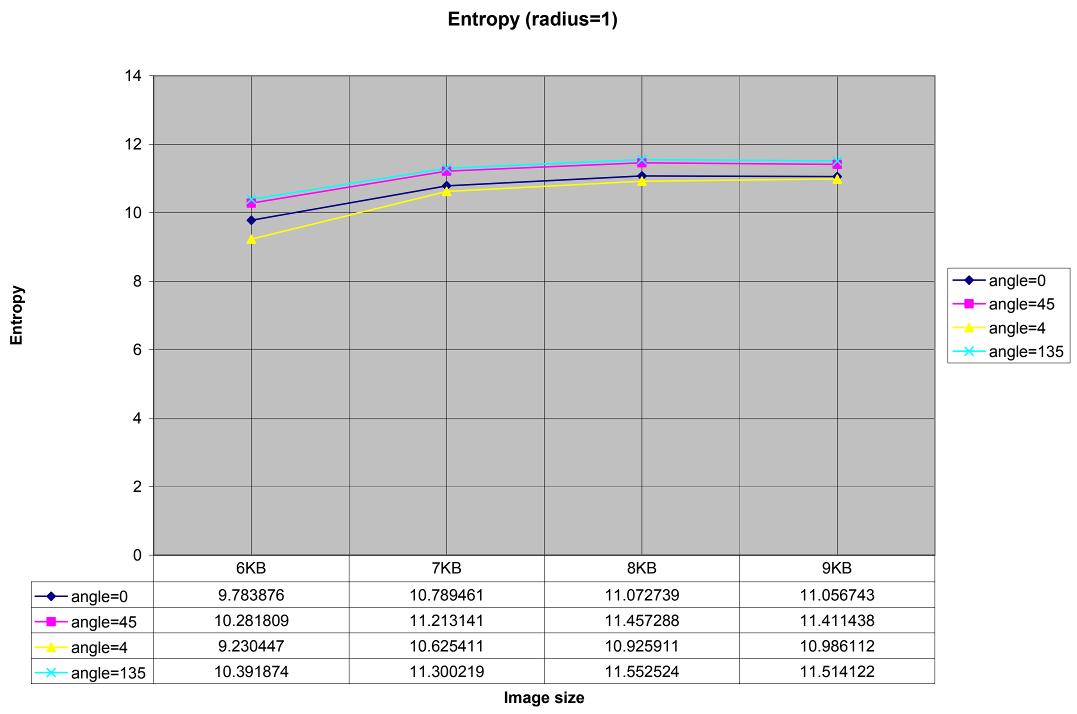
	6KB	7KB	8KB	9KB
radius=1	9.230447	10.625411	10.925911	10.986112
radius=2	9.801897	11.14337	11.414391	11.427129
radius=4	10.479971	11.565541	11.789318	11.7834
radius=8	11.465917	12.216174	12.36948	12.328731

Figure 28c. Entropy angle=90



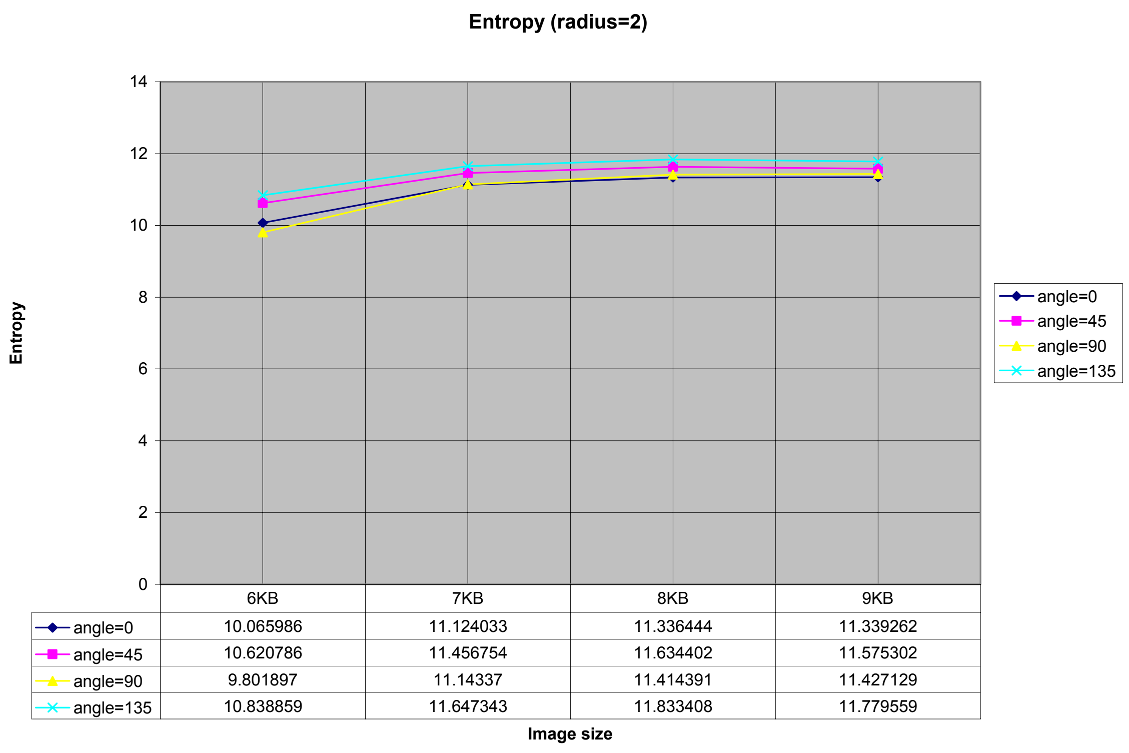
	6KB	7KB	8KB	9KB
radius=1	10.391874	11.300219	11.552524	11.514122
radius=2	10.838859	11.647343	11.833408	11.779559
radius=4	11.319291	11.963691	12.123741	12.055369
radius=8	11.849242	12.386319	12.515609	12.451726

Figure 28d. Entropy angle=135



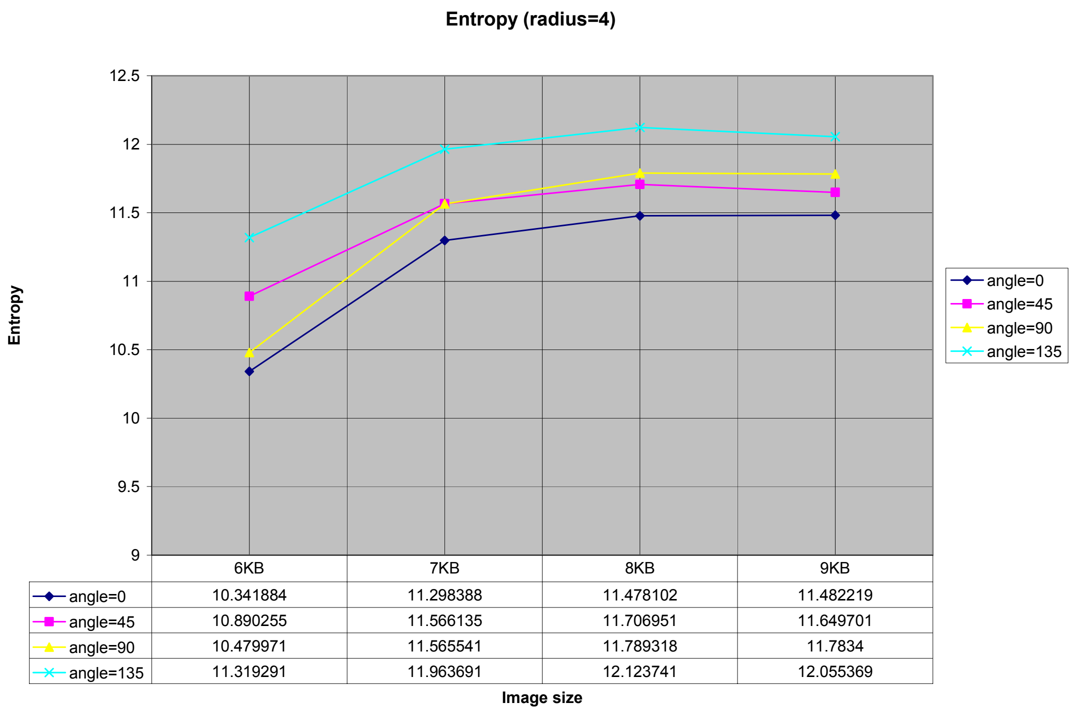
	6KB	7KB	8KB	9KB
angle=0	9.783876	10.789461	11.072739	11.056743
angle=45	10.281809	11.213141	11.457288	11.411438
angle=4	9.230447	10.625411	10.925911	10.986112
angle=135	10.391874	11.300219	11.552524	11.514122

Figure 28e. Entropy radius=1



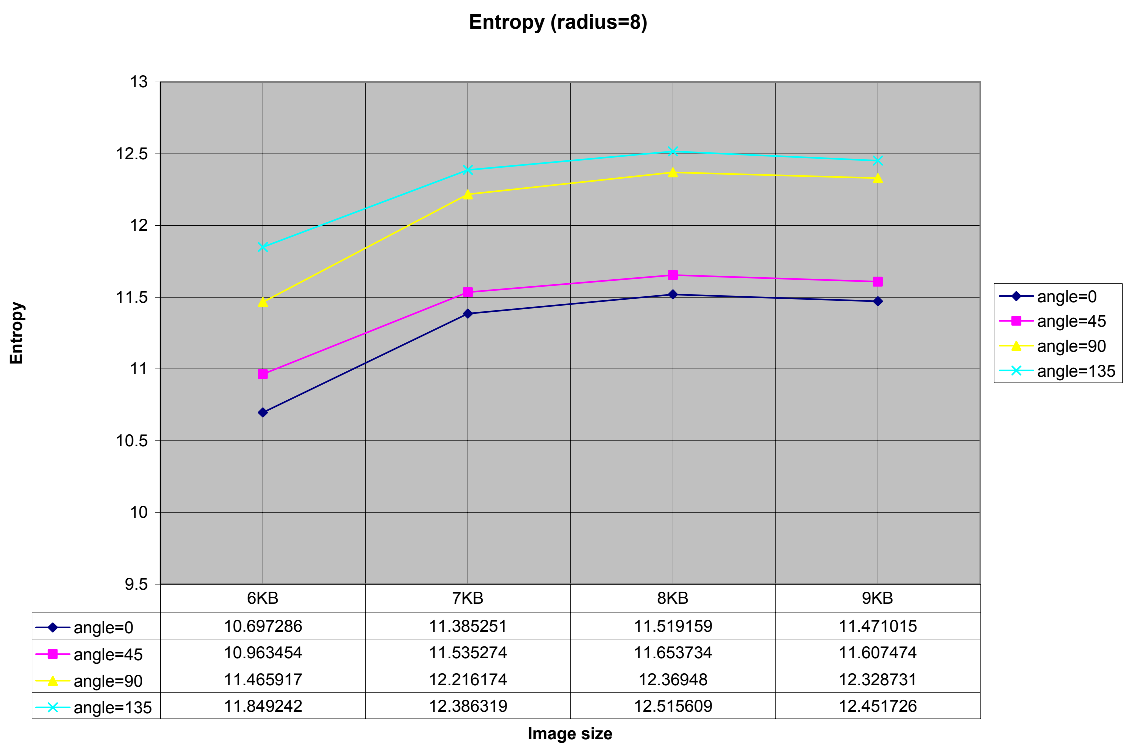
	6KB	7KB	8KB	9KB
angle=0	10.065986	11.124033	11.336444	11.339262
angle=45	10.620786	11.456754	11.634402	11.575302
angle=90	9.801897	11.14337	11.414391	11.427129
angle=135	10.838859	11.647343	11.833408	11.779559

Figure 28f. Entropy radius=2



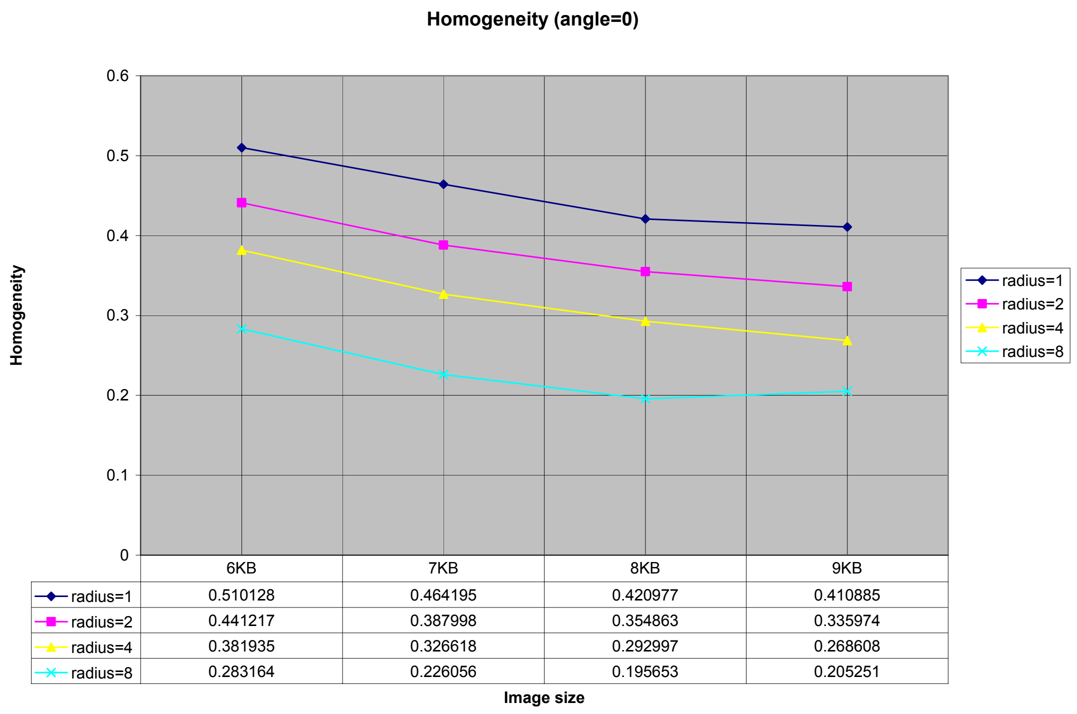
	6KB	7KB	8KB	9KB
angle=0	10.341884	11.298388	11.478102	11.482219
angle=45	10.890255	11.566135	11.706951	11.649701
angle=90	10.479971	11.565541	11.789318	11.7834
angle=135	11.319291	11.963691	12.123741	12.055369

Figure 28g. Entropy radius=4



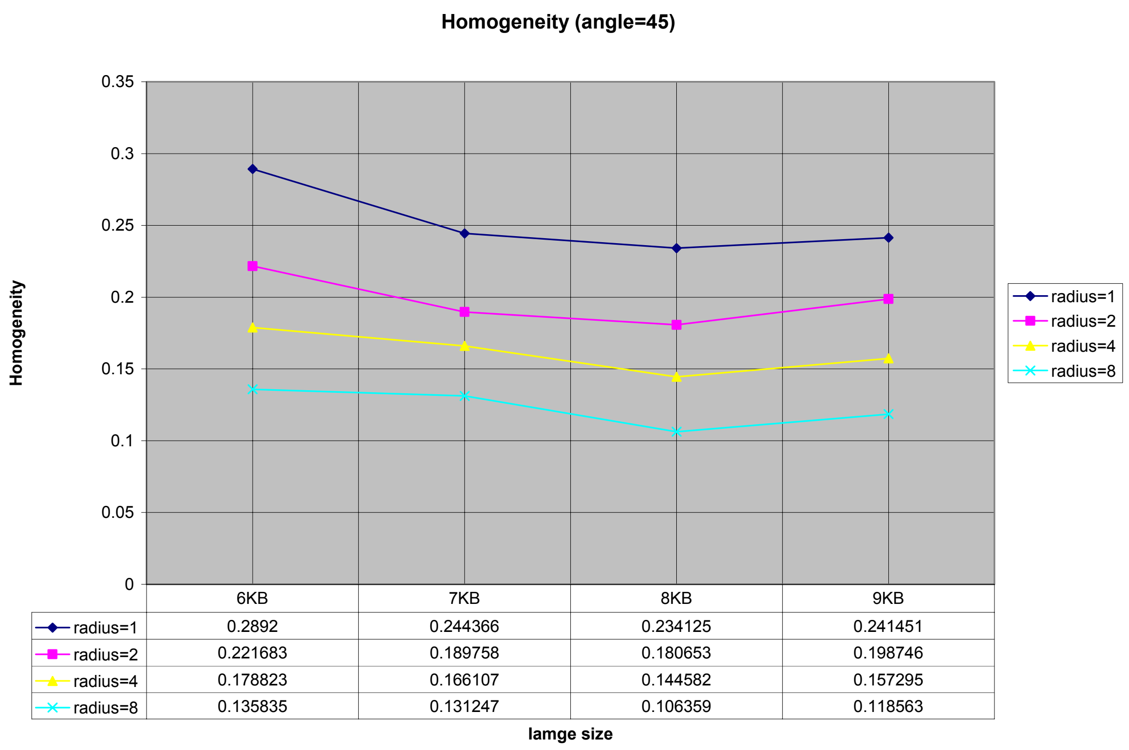
	6KB	7KB	8KB	9KB
angle=0	10.697286	11.385251	11.519159	11.471015
angle=45	10.963454	11.535274	11.653734	11.607474
angle=90	11.465917	12.216174	12.36948	12.328731
angle=135	11.849242	12.386319	12.515609	12.451726

Figure 28h. Entropy radius=8



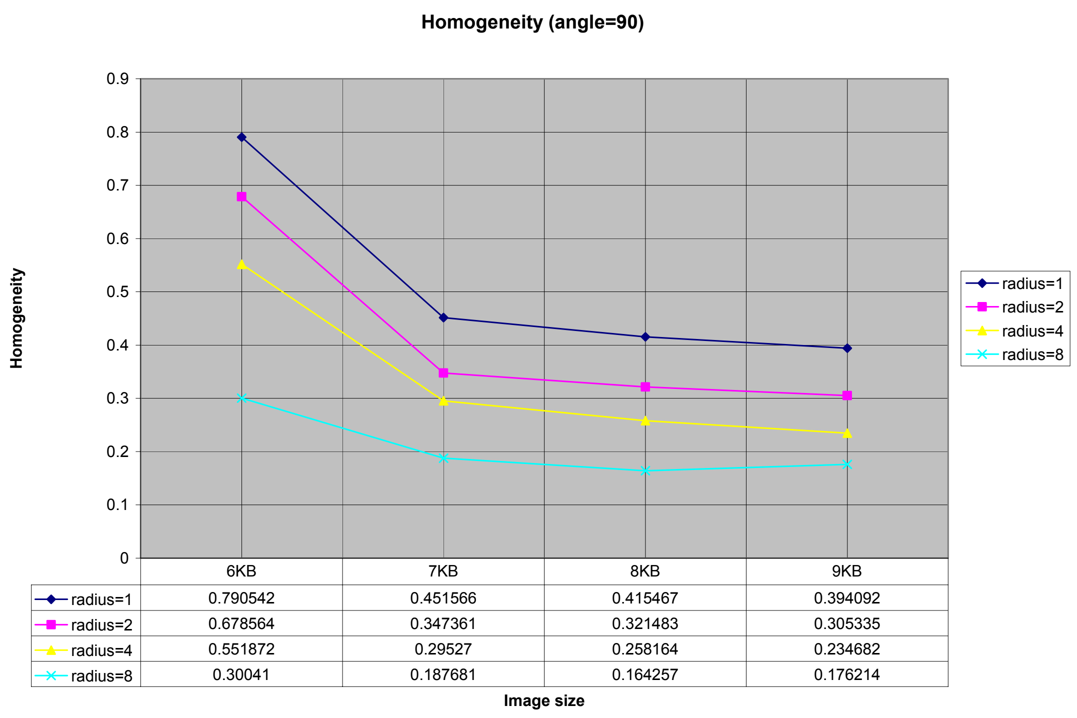
	6KB	7KB	8KB	9KB
radius=1	0.510128	0.464195	0.420977	0.410885
radius=2	0.441217	0.387998	0.354863	0.335974
radius=4	0.381935	0.326618	0.292997	0.268608
radius=8	0.283164	0.226056	0.195653	0.205251

Figure 29a. Homogeneity angle=0



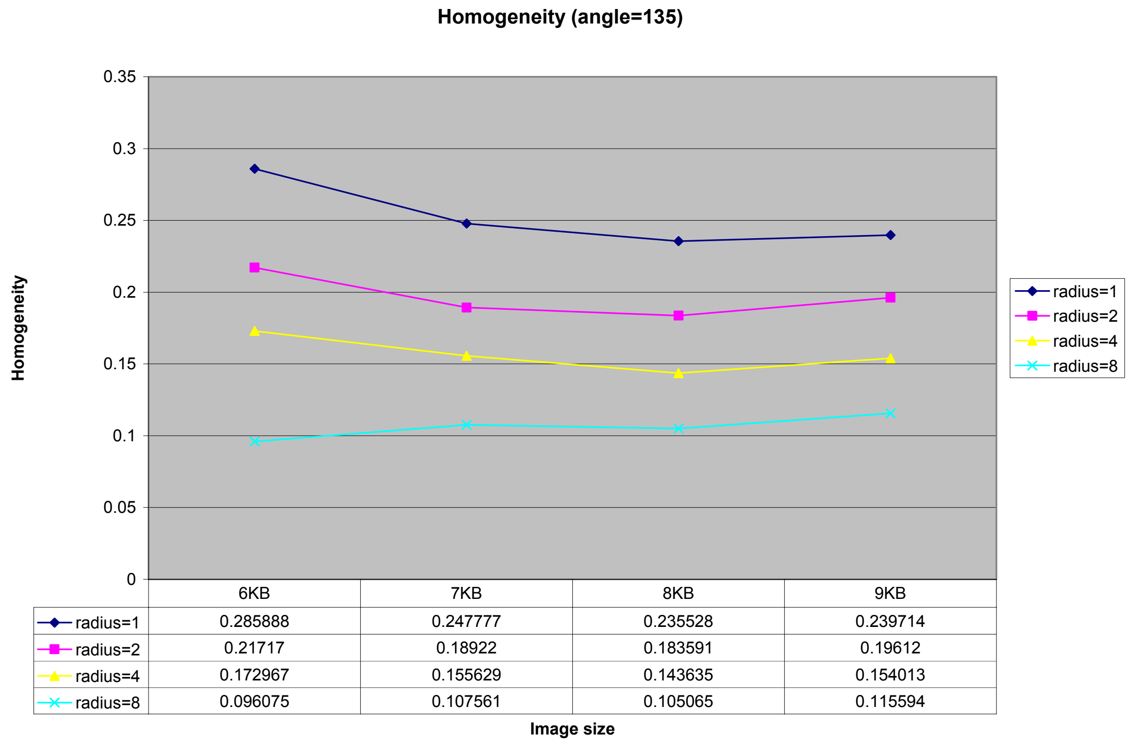
	6KB	7KB	8KB	9KB
radius=1	0.2892	0.244366	0.234125	0.241451
radius=2	0.221683	0.189758	0.180653	0.198746
radius=4	0.178823	0.166107	0.144582	0.157295
radius=8	0.135835	0.131247	0.106359	0.118563

Figure 29b. Homogeneity angle=45



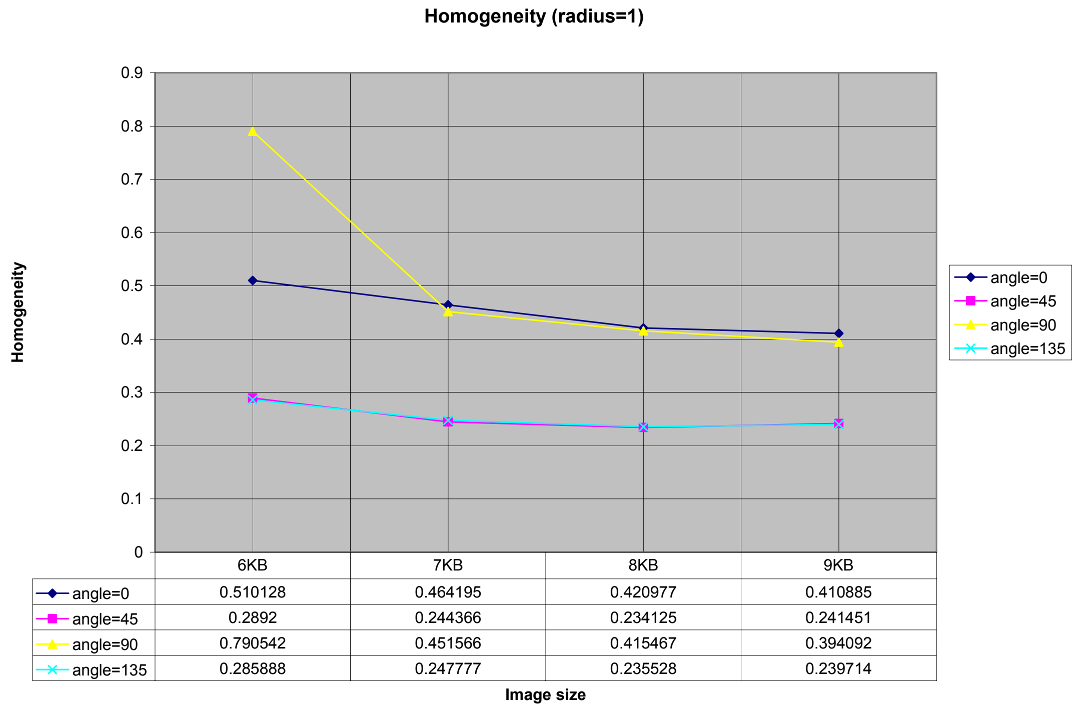
	6KB	7KB	8KB	9KB
radius=1	0.790542	0.451566	0.415467	0.394092
radius=2	0.678564	0.347361	0.321483	0.305335
radius=4	0.551872	0.29527	0.258164	0.234682
radius=8	0.30041	0.187681	0.164257	0.176214

Figure 29c. Homogeneity angle=90



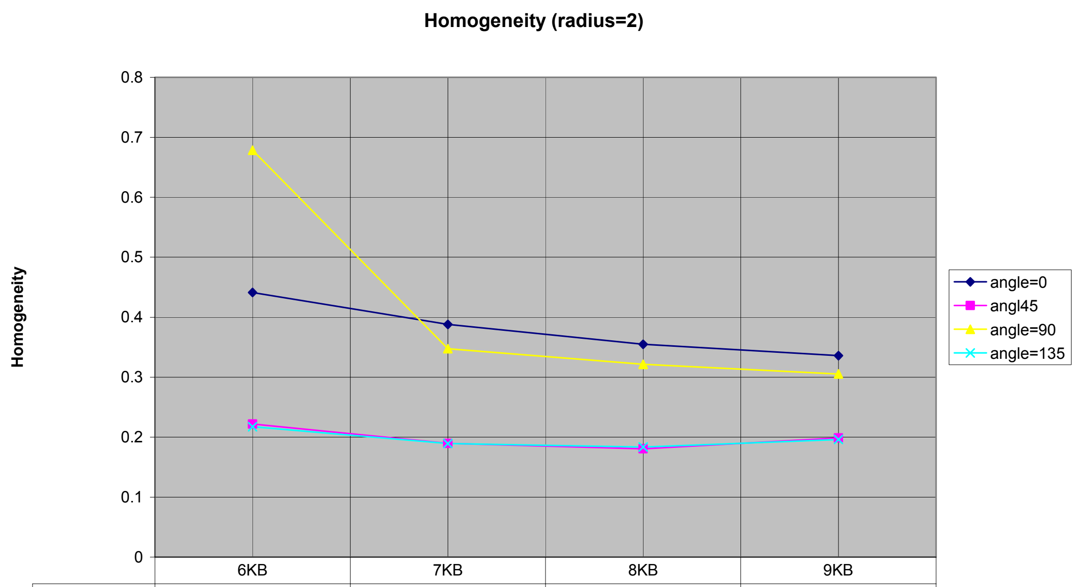
	6KB	7KB	8KB	9KB
radius=1	0.285888	0.247777	0.235528	0.239714
radius=2	0.21717	0.18922	0.183591	0.19612
radius=4	0.172967	0.155629	0.143635	0.154013
radius=8	0.096075	0.107561	0.105065	0.115594

Figure 29d. Homogeneity angle=135



	6KB	7KB	8KB	9KB
angle=0	0.510128	0.464195	0.420977	0.410885
angle=45	0.2892	0.244366	0.234125	0.241451
angle=90	0.790542	0.451566	0.415467	0.394092
angle=135	0.285888	0.247777	0.235528	0.239714

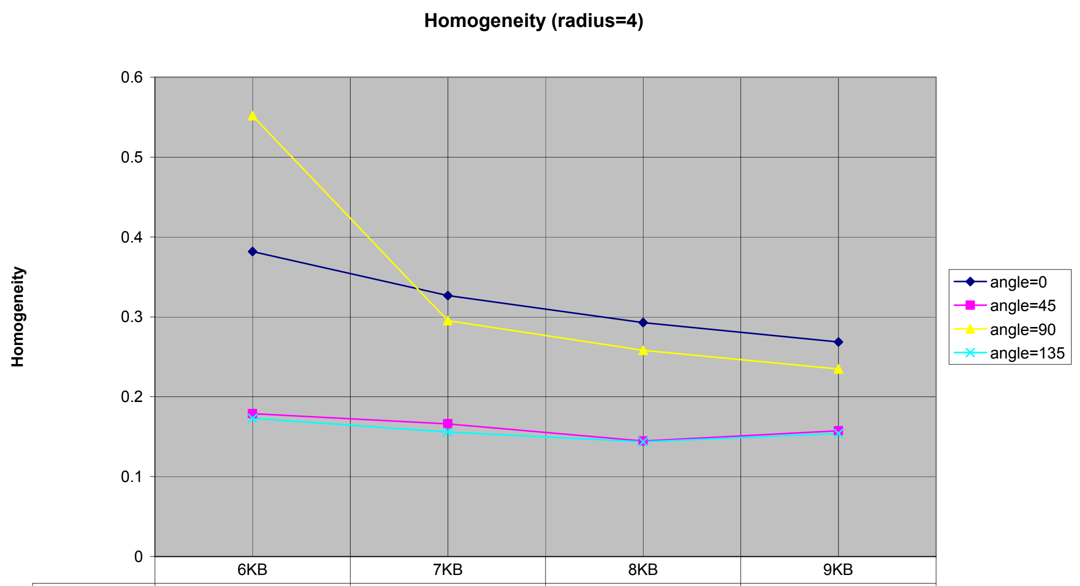
Figure 29e. Homogeneity radius=1



	6KB	7KB	8KB	9KB
angle=0	0.441217	0.387998	0.354863	0.335974
angl45	0.221683	0.189758	0.180653	0.198746
angle=90	0.678564	0.347361	0.321483	0.305335
angle=135	0.21717	0.18922	0.183591	0.19612

Image size

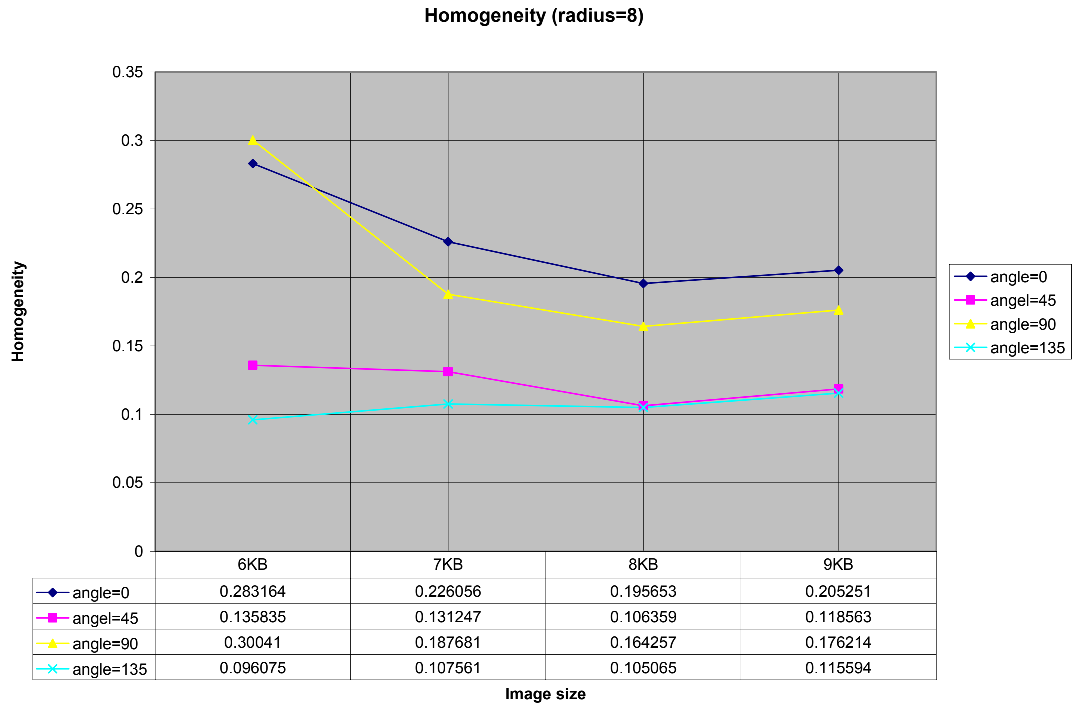
Figure 29f. Homogeneity radius=2



	6KB	7KB	8KB	9KB
angle=0	0.381935	0.326618	0.292997	0.268608
angle=45	0.178823	0.166107	0.144582	0.157295
angle=90	0.551872	0.29527	0.258164	0.234682
angle=135	0.172967	0.155629	0.143635	0.154013

Iamge size

Figure 29g. Homogeneity radius=4



	6KB	7KB	8KB	9KB
angle=0	0.283164	0.226056	0.195653	0.205251
angel=45	0.135835	0.131247	0.106359	0.118563
angle=90	0.30041	0.187681	0.164257	0.176214
angle=135	0.096075	0.107561	0.105065	0.115594

Figure 29h. Homogeneity radius=8

The analysis of all the 32 graphs shows that the expected trend is not observed for all the combinations of radius and angle. For instance, the homogeneity curve for 135° for all values of radius shows a change in the sign of first derivative. However there exists at least one single combination (in this case 2 and 90°) when all the textural parameters follow the desired trend. So the best way to analyze would be considering all the relevant combinations of radius and angle, and plotting the curves of textural features.

CHAPTER 5 OPTIMIZATION TECHNIQUES

Normally a typical GLCM would be a sparse matrix containing a few number of non-zero elements. Calculating GLCM is computationally intensive due to the humungous sizes of matrix involved. It leads to unnecessary calculations involving zero probabilities. Various optimization techniques have been proposed to overcome this problem and are discussed ahead in detail.

5.1 Gray level quantization

The number of gray levels is an important factor in the computation of GLCM as the dimensions of matrix equals the number of gray levels. The fewer the number of gray levels, faster would be the computation. The crucial decision is to decide how many levels are needed to represent a texture successfully. Some of the major quantization schemes are uniform quantization, Gaussian quantization and equal probability quantization. The uniform quantization scheme is the simplest in which gray levels are quantized into separate bins with uniform tolerance limits with no regard to the gray level distribution of the image. The Gaussian quantization technique finely quantizes a particular range of gray levels which might occur more frequently than others. In the equal probability quantization scheme, each bin has similar probability and it has been shown to represent accurate representation of the original image in terms of textural features based on GLCM [Conn78].

5.2 Windowing Technique

This method directly calculates the co-occurrence matrix parameters from the image and is slightly mathematically intensive [Arge90]. It is based on the fact that windows relative to adjacent pixels are mostly overlapping, so the features related to a pixel can be obtained by updating values already calculated. Consider *w(m, n)* to be window relative to pixel *(k, l)* and $w_1(m, n)$ to be relative to pixel *(k+1, l)*. Most occurrences of pixels separated by displacement δ in w can also be found in w_1. The co-occurrence matrix relative to w_1 is obtained by updating w. For instance, if $\delta = 1$ and $\theta = 135°$, g(i, j) corresponding to w_1 is obtained by decrementing by one the entries of *g(i, j)* corresponding to w, due to the pairs on the left hand side and incrementing by one due to the pairs on the right hand column. This algorithm is twice as fast as classical methods.

5.3 Gray level Co-occurrence link list (GLCLL)

Storing GLCM in a linked list can considerably reduce computation time [Zhao01]. A GLCLL stores only the non-zero co-occurring probabilities. A linked list is a data structure that allows rapid access from node to node using pointers. Each node of the GLCLL would consist of a pointer to the previous node, information node containing the co-occurring pair *(i, j)* and it's probability and a pointer to the next node. Hence, double summations over the entire GLCM are avoided and only single summations over the length of the linked list are required. Since the linked list length L is much smaller than the matrix size $N_g \; x \; N_g$, tremendous gains are achieved. The list needs to be kept sorted according to gray level pairs *(i, j)* for rapid searching of a co-occurring pair which compromises the efficiency of GLCLL. If the co-occurring pair is represented on the linked list, its probability value is updated. If the pair is not represented, then a node is inserted and initialized at the proper location in the list. Without a sorted list, it would be necessary to search the entire list for a particular gray level pair, which would be more time consuming. The major advantage of this technique is reduction in computational demands as compared to GLCM, although it results in additional computational overhead to sort the list.

5.4 Gray level co-occurrence hybrid structure (GLCHS)

GLCHS is based on an integrated hash table and linked list approach. It is faster as compared to GLCM as well as GLCLL [Zhao02]. The *listnode* structure defines two integer members to store the gray level pairs and two self-referential pointers to access previous and next *listnode*. In the hash table structure, one float member stores the gray-level co-occurrence probability and the other stores the linked list pointer. The hash table is dimensioned to the lower triangle size. Access to the hash table is provided using *(i, j)* as a unique key. Each entry in the hash table contains a pointer. A null pointer indicates that a particular co-occurring pair *(i, j)* does not have a representative node. Consequently, a new node would be created, inserted at the end of the linked list and its gray level values would be set. If the pointer is not null, then the probability of the existing corresponding node on the linked list is incremented. The GLCHS is built in the order in which the co-occurring pairs are encountered. The hash table allows rapid access to an *(i, j)* pair and the linked list provides a fast means to apply the statistics. The two main advantages are that there is no need of sorted linked list which allows easier addition, deletion and modification of probability associated with a node and lower computation time as compared to GLCLL. However it results in increased complexity in implementation due to a two dimensional hash table with longer linked list.

CONCLUSION AND FUTURE WORK

The research work attempted to investigate the use of GLCM textural parameters as an image quality metric. The proposed method discussed the relevance of radius and angle which happen to be the most crucial input parameters in GLCM processing. It can be concluded that the most appropriate value of radius for analysis would be one as closely spaced pixels are more likely to be correlated than those which are spaced far away. The radius which must be used in computing the GLCM may be obtained from the autocorrelation function of the image. The radius value at which the normalized autocorrelation function of the image becomes too small can serve as an upper bound on the value which may be used for computing the GLCM. No definite conclusion can be drawn regarding the value of angle. For most of the studies, it might be appropriate to calculate the textural parameters for all the four values of angle and use the average value. Thus GLCM happens to be a good discriminator in studying different images however no such claim can be made for image quality. The analysis of the results shows that the nature of the curve of textural parameter versus image size may not always follow a specific trend for chosen values of radius and angle. Performing exhaustive processing for all possible radius and angle values could be considered as an option and then choosing the most appropriate set of graphs. This however reduces the chances of automating the entire process. Hence the search for the best image quality metric continues.

Future research will include datasets that represent texture classes that differ more subtlely. Furthermore, it will be interesting to establish whether this finding holds true for computer generated images as well.

APPENDIX: C/OPENGL CODE

1) main.cpp: This file is used to loading the image and setting the opengl parameters.

```
// This is a compiler directive that includes libraries (For Visual Studio)
#pragma comment(lib, "opengl32.lib")
#pragma comment(lib, "glu32.lib")
#pragma comment(lib, "jpeg.lib")
#include "main.h"
#include "assert.h"
#include <math.h>
bool  g_bFullScreen = TRUE;                              // Set full screen as default
HWND  g_hWnd;                                  // This is the handle for the window
RECT  g_rRect;                                  // This holds the window dimensions
HDC   g_hDC;                            // General HDC - (handle to device context)
HGLRC g_hRC;        //General OpenGL_DC - Our Rendering Context for OpenGL
HINSTANCE g_hInstance; // Holds the global hInstance for UnregisterClass() in DeInit()
UINT g_Texture[MAX_TEXTURES];
        // This will reference to our texture data stored with OpenGL UINT is an unsigned
int (only positive numbers)
void Init(HWND hWnd)
{
        g_hWnd = hWnd;       // Assign the window handle to a global window handle
GetClientRect(g_hWnd, &g_rRect); // Assign the windows rectangle to a global RECT
InitializeOpenGL(g_rRect.right, g_rRect.bottom);    // Init OpenGL with the global rect
/////// * ////////// * ////////// * NEW * /////// * ////////// * ////////// *
```

```
/******** Load "Image.jpg" into OpenGL as a texture********************/
CreateTexture(g_Texture, "C:\\clarke\\harshal\\dataset1\\studentunion30.jpg", 0);
}
///////////////////////////////// MAIN GAME LOOP \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*
WPARAM MainLoop()
{
        MSG msg;
        while(1)                                            // Do our infinite loop
        {                                           // Check if there was a message
                if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
      {
                        if(msg.message == WM_QUIT) // If the message wasnt to quit
                                break;
        TranslateMessage(&msg);             // Find out what the message does
        DispatchMessage(&msg);              // Execute the message
      }
                else                                // if there wasn't a message
                {
                        RenderScene();              // Redraw the scene every frame
      }
        }
        DeInit();                           // Free all the app's memory allocated
        return(msg.wParam);                 // Return from the program
```

```
}
//        This function renders the entire scene.
void RenderScene()
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);        //   Clear   The
Screen And The Depth Buffer
glMatrixMode(GL_PROJECTION);              // Select The Projection Matrix
          glLoadIdentity();                         // Reset The Projection Matrix
          glOrtho(-1,1,-1,1,-1,1);
          glMatrixMode(GL_MODELVIEW);              // Reset The matrix
                    //          Position      View        Up Vector
          // Bind the texture stored at the zero index of g_Texture[]
          glBindTexture(GL_TEXTURE_2D, g_Texture[0]);
          glBegin(GL_QUADS);          // Display a quad texture to the screen
glTexCoord2f(0.0f, 0.0f);      glVertex3f(-1.0, 1.0, 0);        // Display the top left vertice
glTexCoord2f(0.0f, 1.0f);  glVertex3f(-1.0, -1.0, 0); // Display the bottom left vertice
glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0, -1.0, 0); // Display the bottom right vertice
glTexCoord2f(1.0f, 0.0f);  glVertex3f(1.0, 1.0, 0); // Display the top right vertice
glEnd();                                              // Stop drawing QUADS
          SwapBuffers(g_hDC);           // Swap the backbuffers to the foreground
}
//        This function handles the window messages.
```

```
LRESULT CALLBACK WinProc(HWND hWnd,UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
   LONG    lRet = 0;
   PAINTSTRUCT    ps;
   switch (uMsg)
         {
   case WM_SIZE:                                        // If the window is resized
                  if(!g_bFullScreen)              // Do this only if we are NOT in full screen
                  {
SizeOpenGLScreen(LOWORD(lParam),HIWORD(lParam));              //LoWord=Width,
HiWord=Height
GetClientRect(hWnd, &g_rRect);              // Get the window rectangle
                  }
      break;
 case WM_PAINT:                         // If we need to repaint the scene
                  BeginPaint(hWnd, &ps);                  // Init the paint struct
                  EndPaint(hWnd, &ps);          // EndPaint, Clean up
                  break;
case WM_KEYDOWN:
                  switch(wParam) {                                    // Check if we hit a key
                           case VK_ESCAPE:              // If we hit the escape key
                  PostQuitMessage(0);  // Send a QUIT message to the window
```

```
                    break;
            }
            break;
    case WM_CLOSE:                        // If the window is being closes
        PostQuitMessage(0);     // Send a QUIT Message to the window
        break;
         default:                                    // Return by default
        lRet = DefWindowProc (hWnd, uMsg, wParam, lParam);
        break;
    }
    return lRet;                                      // Return by default
}
```

2) init.cpp: This file contains the code for GLCM processing and calculation of textural parameters.

```
#include "main.h"
//      This decodes the jpeg and fills in the tImageJPG structure
void DecodeJPG(jpeg_decompress_struct* cinfo, tImageJPG *pImageData)
{       // Read in the header of the jpeg file
jpeg_read_header(cinfo, TRUE); // Start to decompress the jpeg file
jpeg_start_decompress(cinfo);
        // Get the image dimensions and row span to read in the pixel data
        pImageData->rowSpan = cinfo->image_width * cinfo->num_components;
        pImageData->sizeX   = cinfo->image_width;
        pImageData->sizeY   = cinfo->image_height;
        // Allocate memory for the pixel buffer
pImageData->data = new unsigned char[pImageData->rowSpan * pImageData->sizeY];
// Here we use the library's state variable cinfo.output_scanline as the
// loop counter, so that we don't have to keep track ourselves.
// Create an array of row pointers
        unsigned char** rowPtr = new unsigned char*[pImageData->sizeY];
        for (int i = 0; i < pImageData->sizeY; i++)
                rowPtr[i] = &(pImageData->data[i*pImageData->rowSpan]);
        // Now comes the juice of our work, here we extract all the pixel data
        int rowsRead = 0;
        while (cinfo->output_scanline < cinfo->output_height)
        {       // Read in the current row of pixels and increase the rowsRead count
rowsRead  +=  jpeg_read_scanlines(cinfo,  &rowPtr[rowsRead],  cinfo->output_height  -
rowsRead);
        }
delete [] rowPtr;        // Delete the temporary row pointers
jpeg_finish_decompress(cinfo);        // Finish decompressing the data
}
//        This loads the JPG file and returns it's data in a tImageJPG struct
tImageJPG *LoadJPG(const char *filename)
{
        struct jpeg_decompress_struct cinfo;
```

```
          tImageJPG *pImageData = NULL;
FILE *pFile; // Open a file pointer to the jpeg file and check if it was found and opened
          if((pFile = fopen(filename, "rb")) == NULL)
          { // Display an error message saying the file was not found, then return NULL
                    MessageBox(g_hWnd, "Unable to load JPG File!", "Error", MB_OK);
                    return NULL;
          }
          jpeg_error_mgr jerr; // Create an error handler
          // Have our compression info object point to the error handler address
          cinfo.err = jpeg_std_error(&jerr);
          jpeg_create_decompress(&cinfo);     // Initialize the decompression object
          jpeg_stdio_src(&cinfo, pFile);          // Specify the data source (Our file pointer)
          // Allocate the structure that will hold our eventual jpeg data (must free it!)
          pImageData = (tImageJPG*)malloc(sizeof(tImageJPG));
          // Decode the jpeg file and fill in the image data structure to pass back
          DecodeJPG(&cinfo, pImageData);
          // This releases all the stored memory for reading and decoding the jpeg
          jpeg_destroy_decompress(&cinfo);
          fclose(pFile);  // Close the file pointer that opened the file
          return pImageData;    // Return the jpeg data


}
//        This creates a texture in OpenGL that we can use as a texture map
```

```
void CreateTexture(UINT textureArray[], LPSTR strFileName, int textureID)
{
        if(!strFileName)        // Return from the function if no file name was passed in
                return;
tImageJPG *pImage = LoadJPG(strFileName); //GET PIXEL INFO OF JPEG IN pImage
FILE *target, *source; int i,j,k; unsigned char * gray;
gray = (unsigned char *) malloc (pImage->sizeX * pImage->sizeY);
target = fopen("C:\\clarke\\code\\jpeg\\Copy of TexturingIII\\results\\grayinfo.xls","w");
fprintf(target,"width=%d,\t height=%d rowspan=%d\n",pImage->sizeX, pImage->sizeY,
pImage->rowSpan);
short signed int radiusrow, radiuscol, radius, angle, width, height;
int x, y,   graylevels, xstretch, ystretch, row, col; //*image,
float   tempf;//, energy_nor, homogeneity_nor, inertia_nor, entropy_nor;
float *glcm, *buff,  no_of_pairs, counter=0.0;
double entropy, inertia, homogeneity, energy;
angle = 0; // SET INPUT PARAMETERS
radius = 1; height = pImage->sizeY;  width = pImage->sizeX;
switch(angle)
        {
        case 0:
                radiusrow = 0; radiuscol = radius;
                no_of_pairs = (float)(width-radius) * height;
                break;
```

```
case 180:
        radiusrow = 0; radiuscol = radius;
        no_of_pairs = (float)(width-radius) * height;
        break;
case 45:
        radiusrow = 0-radius; radiuscol = radius;
        no_of_pairs = (float)(height-radius) * (width-radius);
        break;
case 225:
        radiusrow = 0-radius; radiuscol = radius;
        no_of_pairs = (float)(height-radius) * (width-radius);
        break;
case 90:
        radiusrow = radius; radiuscol = 0;
        no_of_pairs = (float)(height-radius) * width;
        break;
case 270:
        radiusrow = radius; radiuscol = 0;
        no_of_pairs = (float)(height-radius) * width;
        break;
case 135:
        radiusrow = 0 - radius; radiuscol = 0 - radius;
        no_of_pairs = (float)(height-radius) * (width-radius);
```

```
                break;
        case 315:
                radiusrow = 0 - radius; radiuscol = 0 - radius;
                no_of_pairs = (float)(height-radius) * (width-radius);
                break;
        }
        for(i=0; i<pImage->sizeY; i++)
        {
                for(k=0,j=0; k<pImage->rowSpan/3; k++, j=j+3)
                {
*  (gray+i*pImage->sizeX+k)  =  (pImage->data[(i*pImage->rowSpan)+j]  +  pImage-
>data[(i*pImage->rowSpan)+j+1] + pImage->data[(i*pImage->rowSpan)+j+2]) / 3;
        fprintf(target,"%d\t", *(gray+i*pImage->sizeX+k));
                }
                fprintf(target,"\n");
        }
        fclose(target);
                xstretch = ystretch = graylevels = 256;
                glcm = (float *) malloc (8*graylevels * graylevels);
                buff = (float *) malloc (8*graylevels * graylevels);
                for(x=0; x<xstretch; x++) //CLEARING BUFF AND GLCM MATRICES
                {
                        for(y=0; y<ystretch; y++)
```

```
                    {
                        *(buff+x+y*xstretch) = 0; *(glcm+x+y*xstretch) = 0;
                    }
                }
for( row=0; row<xstretch ; row++)
        {
                for( col=row; col<ystretch; col++)
                {
                    counter = 0.0;
        for(x=0; x<height  ; x++)
                    {
                        for(y=0; y<width; y++)
                        {
if( (x+radiusrow)<0 || (x+radiusrow)>=width || (y+radiuscol)<0 || (y+radiuscol)>=height )
//BOUNDARY CONDITION
                            {
                            }
                            else
                            {
if(*(gray+x*width+y) == row && *(gray+((x+radiusrow)*width)+(y+radiuscol)) == col
|| *(gray+x*width+y) == col && *(gray+((x+radiusrow)*width)+(y+radiuscol)) == row)
                                {
                                    counter++;
```

```
                                                }
                                        }
                                }
                        }
          if(row==col)   *(buff+row*xstretch+col) = 2*counter;
          else           *(buff+row*xstretch+col) = counter;
          counter = 0.0;
                    }
          }
for( row=0; row<xstretch ; row++) //FILL LOWER TRIANGLE OF GLCM
                    {
                              for( col=0; col<row; col++)
                              {
                              *(buff+row*xstretch+col) = *(buff+col*xstretch+row);
                              }
                    }
target = fopen("C:\\clarke\\code\\jpeg\\Copy of TexturingIII\\results\\glcm.xls","w");
          for(i=0; i<ystretch; i++)
          {
                    for(j=0; j<xstretch; j++)
                    {
          *(glcm+i*xstretch+j) = ((*(buff+i*xstretch+j))/no_of_pairs);
          fprintf(target,"%f\t", *(buff+i*xstretch+j));
```

```
                    }
                    fprintf(target,"\n");
          }
          fclose(target);
energy = 0.0;   //ENERGY COMPUTATION
for(x=0; x<xstretch; x++)
          {
                    for(y=0; y<ystretch; y++)
                    {
                              energy = energy + pow(*(glcm+x+y*xstretch),2);
                    }
          }
inertia = 0.0; //INERTIA/CONTRAST COMPUTATION
for(x=0; x<xstretch; x++)
          {
                    for(y=0; y<ystretch; y++)
                    {
                              inertia = inertia + (pow((x-y),2) * (*(glcm+x+y*xstretch)));
                    }
          }
double tempdouble = 0.0;       //ENTROPY COMPUTATION
entropy = 0.0;
          for(x=0; x<xstretch; x++)
```

```
        {
                for(y=0; y<ystretch; y++)
                {
                        if(*(glcm+x+y*xstretch) != 0.0)
                        {
                                tempdouble = (double)*(glcm+x+y*xstretch);
                                tempdouble = log(tempdouble) * tempdouble;
                                entropy = entropy +  tempdouble;
                                tempdouble = 0.0;
                        }
                }
        }
        entropy = 0 - entropy;
tempf = 0.0;    // HOMOGENEITY COMPUTATION
homogeneity = 0.0;
        for(x=0; x<xstretch; x++)
        {
                for(y=0; y<ystretch; y++)
                {
                        tempdouble = pow((x-y),2) + 1;
                        tempdouble = 1 / tempdouble;
                        tempdouble = tempdouble * (*(glcm+x+y*xstretch));
                        homogeneity = homogeneity + tempdouble;
```

```
                    tempdouble = 0.0;
                }
            }
source = fopen("C:\\clarke\\code\\jpeg\\Copy of TexturingIII\\results\\glcminfo.txt",
"w");
fprintf(source, "\n\n ENERGY = %lf \n INERTIA/CONTRAST = %lf \n ENTROPY =
%lf \n HOMOGENEITY = %lf", energy, inertia, entropy,  homogeneity);
fclose(source);
fclose(target);
if(pImage == NULL)                          // If we can't load the file, quit!
                exit(0);
glGenTextures(1, &textureArray[textureID]); // Generate a texture with the associative
texture ID stored in the array
// Bind the texture to the texture arrays index and init the texture
glBindTexture(GL_TEXTURE_2D, textureArray[textureID]);
// Build Mipmaps (builds different versions of the picture for distances - looks better)
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, pImage->sizeX, pImage->sizeY, GL_RGB,
GL_UNSIGNED_BYTE, pImage->data);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MI
PMAP_NEAREST);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR_MI
PMAP_LINEAR);
// Now we need to free the image data that we loaded since OpenGL stored it as a texture
```

```
if (pImage)                                         // If we loaded the image
        {
                if (pImage->data)                   // If there is texture data
                {
                free(pImage->data);   // Free the texture data, we don't need it anymore
                }
                free(pImage);           // Free the image structure
        }
}
//      This changes the screen to FULL SCREEN
void ChangeToFullScreen()
{
        DEVMODE dmSettings;                         // Device Mode variable
memset(&dmSettings,0,sizeof(dmSettings));           // Makes Sure Memory's Cleared
// Get current settings -- This function fills our the settings
// This makes sure NT and Win98 machines change correctly
if(!EnumDisplaySettings(NULL,ENUM_CURRENT_SETTINGS,&dmSettings))
        {               // Display error message if we couldn't get display settings
                MessageBox(NULL, "Could Not Enum Display Settings", "Error",
MB_OK);
                return;
        }
dmSettings.dmPelsWidth      = SCREEN_WIDTH;         // Selected Screen Width
dmSettings.dmPelsHeight      = SCREEN_HEIGHT;// Selected Screen Height
        // This function actually changes the screen to full screen
        // CDS_FULLSCREEN Gets Rid Of Start Bar.
        // We always want to get a result from this function to check if we failed
        int result = ChangeDisplaySettings(&dmSettings,CDS_FULLSCREEN);
        // Check if we didn't recieved a good return message From the function
        if(result != DISP_CHANGE_SUCCESSFUL)
        {                   // Display the error message and quit the program
                MessageBox(NULL, "Display Mode Not Compatible", "Error", MB_OK);
                PostQuitMessage(0);
        }
}
//      This function creates a window, but doesn't have a message loop
HWND CreateMyWindow(LPSTR strWindowName, int width, int height, DWORD
dwStyle, bool bFullScreen, HINSTANCE hInstance)
{
        HWND hWnd;
        WNDCLASS wndclass;
memset(&wndclass, 0, sizeof(WNDCLASS));         // Init the size of the class
wndclass.style = CS_HREDRAW | CS_VREDRAW; // Regular drawing capabilities
wndclass.lpfnWndProc = WinProc; // Pass our function pointer as the window procedure
wndclass.hInstance = hInstance;                 // Assign our hInstance
wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION); // General icon
wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);    // An arrow for the cursor
wndclass.hbrBackground = (HBRUSH) (COLOR_WINDOW+1);  // A white window
wndclass.lpszClassName = "GameTutorials";          // Assign the class name
RegisterClass(&wndclass);                          // Register the class
        if(bFullScreen && !dwStyle) // Check if we wanted full screen mode
        {                               // Set the window properties for full screen mode
                dwStyle = WS_POPUP | WS_CLIPSIBLINGS | WS_CLIPCHILDREN;
                ChangeToFullScreen();                   // Go to full screen
                ShowCursor(FALSE);                      // Hide the cursor
        }
        else if(!dwStyle)       // Assign styles to the window depending on the choice
dwStyle     =     WS_OVERLAPPEDWINDOW     |     WS_CLIPSIBLINGS     |
WS_CLIPCHILDREN;
        g_hInstance = hInstance;// Assign our global hInstance to the window's hInstance
        RECT rWindow;
rWindow.left  = 0;                                 // Set Left Value To 0
        rWindow.right= width;               // Set Right Value To Requested Width
        rWindow.top     = 0;                       // Set Top Value To 0
        rWindow.bottom       = height;      // Set Bottom Value To Requested Height
AdjustWindowRect( &rWindow, dwStyle, false); // Adjust Window To True Requested
Size
// Create the window
hWnd = CreateWindow("GameTutorials", strWindowName, dwStyle, 0, 0,
rWindow.right    -  rWindow.left,  rWindow.bottom  -  rWindow.top,  NULL,  NULL,
hInstance, NULL);
if(!hWnd) return NULL;                    // If we could get a handle, return NULL
ShowWindow(hWnd, SW_SHOWNORMAL);          // Show the window
        UpdateWindow(hWnd);               // Draw the window
        SetFocus(hWnd);                   // Sets Keyboard Focus To The Window
        return hWnd;
}
//      This function sets the pixel format for OpenGL.
bool bSetupPixelFormat(HDC hdc)
{
   PIXELFORMATDESCRIPTOR pfd;
   int pixelformat;
  pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);// Set the size of the structure
   pfd.nVersion = 1;                      // Always set this to 1
// Pass in the appropriate OpenGL flags
pfd.dwFlags   =   PFD_DRAW_TO_WINDOW   |   PFD_SUPPORT_OPENGL   |
PFD_DOUBLEBUFFER;
pfd.dwLayerMask  =  PFD_MAIN_PLANE;//  We  want  the  standard  mask
pfd.iPixelType = PFD_TYPE_RGBA;           // We want RGB and Alpha pixel type
pfd.cColorBits = SCREEN_DEPTH;// Here we use our #define for the color bits
pfd.cDepthBits = SCREEN_DEPTH;// Depthbits is ignored for RGBA
pfd.cAccumBits = 0;                       // No special bitplanes needed
pfd.cStencilBits = 0;                   // We desire no stencil bits
// This gets us a pixel format that best matches the one passed in from the device
   if ( (pixelformat = ChoosePixelFormat(hdc, &pfd)) == FALSE )
   {
     MessageBox(NULL, "ChoosePixelFormat failed", "Error", MB_OK);
     return FALSE;
   }
        // This sets the pixel format that we extracted from above
   if (SetPixelFormat(hdc, pixelformat, &pfd) == FALSE)
   {
     MessageBox(NULL, "SetPixelFormat failed", "Error", MB_OK);
     return FALSE;
   }
   return TRUE;                          // Return a success!
}
//      This function resizes the viewport for OpenGL.
void SizeOpenGLScreen(int width, int height) // Initialize The GL Window
{
        if (height==0)                          // Prevent A Divide By Zero error
        {
                height=1;                       // Make the Height Equal One
        }
        glViewport(0,0,width,height);           // Make our viewport the whole window
```

```
glMatrixMode(GL_PROJECTION);            // Select The Projection Matrix
        glLoadIdentity();               // Reset The Projection Matrix
                // Calculate The Aspect Ratio Of The Window
        gluPerspective(45.0f,(GLfloat)width/(GLfloat)height, .5f ,150.0f);
        glMatrixMode(GL_MODELVIEW);             // Select The Modelview Matrix
        glLoadIdentity();                       // Reset The Modelview Matrix
}
//      This function handles all the initialization for OpenGL.
void InitializeOpenGL(int width, int height)
{
   g_hDC = GetDC(g_hWnd);                       // This sets our global HDC
  if (!bSetupPixelFormat(g_hDC))    // This sets our pixel format/information
      PostQuitMessage (0);              // If there's an error, quit
   g_hRC = wglCreateContext(g_hDC);// This creates a rendering context from our hdc
   wglMakeCurrent(g_hDC, g_hRC);// This makes the rendering context we just created
        // This allows us to use texture mapping, otherwise we just use colors.

        glEnable(GL_TEXTURE_2D);                // Enable Texture Mapping
SizeOpenGLScreen(width, height);        // Setup the screen translations and viewport
        }
//      This function cleans up and then posts a quit message to the window
void DeInit()
{
```

```
          if (g_hRC)
          {
                    wglMakeCurrent(NULL, NULL);
          // This frees our rendering memory and sets everything back to normal
                    wglDeleteContext(g_hRC);    // Delete our OpenGL Rendering Context
          }
          if (g_hDC)
                    ReleaseDC(g_hWnd, g_hDC); // Release our HDC from memory
                    if(g_bFullScreen)                   // If we were in full screen
          {
                    ChangeDisplaySettings(NULL,0);// If So Switch Back To The Desktop
                    ShowCursor(TRUE);                         // Show Mouse Pointer
          }
UnregisterClass("GameTutorials", g_hInstance);                  // Free the window class
          PostQuitMessage (0);            // Post a QUIT message to the window
}
//        This function handles registering and creating the window.
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hprev, PSTR cmdline, int
ishow)
{
          HWND hWnd;           // Check if we want full screen or not
          if(MessageBox(NULL, "Click Yes to go to full screen (Recommended)",
"Options", MB_YESNO | MB_ICONQUESTION) == IDNO)
```

```
                g_bFullScreen = false;

                // Create our window with our function we create that passes in the:

        // Name, width, height, any flags for the window, if we want fullscreen of not, and
the hInstance

        hWnd = CreateMyWindow("Texture Mapping JPEGs", SCREEN_WIDTH,
SCREEN_HEIGHT, 0, g_bFullScreen, hInstance);

        // If we never got a valid window handle, quit the program

        if(hWnd == NULL) return TRUE;

        Init(hWnd);                                     // INIT OpenGL

        // Run our message loop and after it's done, return the result

        return MainLoop();

}
```

REFERENCES

[Eski00] A. M. Eskicioglu, “Quality measurement for monochrome compressed images in the past 25 years”, Acoustics, Speech and Signal Processing, IEEE International Conference, Volume 6, June 2000.

[Wang02] Zhou Wang, Bovik A. C. and Ligang Lu, “Why is image quality assessment so difficult?”, Acoustics, Speech and Signal Processing, IEEE International Conference, Volume 4, May 2002.

[Joon01] Joonmi Oh, Sandra I. Woollley, Theodoros N. Arvanitis and John N. Townend “A Multistage Perceptual Quality Assessment for Compressed Digital Angiogram Images”, IEEE Transactions on Medical Imaging, Vol. 20, No. 12, December 2001

[Gonz02] R. C. Gonzalez and R. E. .Woods, “Digital Image Processing”, Second Edition 2002

[Rose01] C. Rosenberger and C. Cariou, “Contribution to Texture Analysis”, In Proc. International Conference on Quality Control by Artificial Vision,vol. 1, pp. 122-126, Le Creuzot, 2001

[Hara73] R. M. Haralick, K. Shanmugam and I. Dinstein “Textural features for Image Classification”, IEEE Transactions on Systems, Man and Cybernetics, Vol.3, pp. 610-621, November 1973

[Clau02] David A. Clausi, “An analysis of co-occurrence texture statistics as a function of gray level quantization”, Can. J. Remote Sensing, Vol. 28, No. 1, pp. 45-62, 2002

[Bara95] Baraldi Andrea and Parmigianni Flavio, “An Investigation of the Textural Characteristics associated with gray level co-occurrence matrix statistical parameters”, IEEE Transactions on Geoscience and Remote Sensing, vol. 33, No. 2, March 1995.

[Hadd93] J. F. Haddon and J. F. Boyce, “Co-occurrence matrices for Image Analysis”, IEE Electronics & Communication Engineering Journal, Volume 5, Issue 2, April 1993, Pages 71 – 83

[Kova96] V. Kovalev and M. Petrou, “Multidimensional Co-occurrence matrices for object recognition and matching”, Graphical models and image processing, vol. 58, No. 3, May 1996, article no. 0016

[Soh99] L. Soh and C. Tsatsoulis, “Texture Analysis of SAR Sea Ice Imagery using gray level co-occurrence matrices”, IEEE transactions on Geoscience and Remote Sensing, Vol. 37, No. 2, March 1999

[Lohm95] G. Lohmann, “Analysis and Synthesis of Textures: A Co-occurrence based approach”, Computer and Graphics, Vol. 19, No. 1, pp. 29 – 36, 1995

[Palm03] C. Palm, “Color Texture Classification by integrative co-occurrence matrices”, Pattern Recognition, September 2003.

[Coop04] G. R. J. Cooper, “The textural analysis of gravity data using co-occurrence matrices”, Computers and Geosciences 30 (2004) 107 – 115

[Lati00] A. Latif-Amet, A. Ertuzun, A. Ercil, “An efficient method for texture defect detection: sub-band domain co-occurrence matrices”, Image and Vision Computing 18 (2000) 543 – 553

[Arge90] F. Argenti, L. Alparone and G. Benelli, “Fast Algorithms for texture analysis using co-occurrence matrices”, IEE Proceedings, Vo. 137, No. 6, December 1990.

[Clau98] D. A. Clausi and M. E. Jernigan, “A fast method to determine co-occurrence texture features”, IEEE Transactions on Geoscience and Remote Sensing 36 (1), 298-300, January 1998

[Zhao01] Yongping Zhao and David A. Clausi, “Rapid Determination of Co-occurrence Texture Features”, Geoscience and Remote Sensing Symposium, 2001. IGARSS '01. IEEE 2001 International ,Volume: 4 , 9-13 July 2001 Pages:1880 - 1882 vol.4 IEEE 2001

[Zhao02] Yongping Zhao and David A. Clausi, “Rapid extraction of image texture by co-occurrence using a hybrid data structure”, Computers and Geoscicences 28 (2002) 763-774

[Conn78] R. W. Conners and C. A. Harlow, “Equal probability quantizing and texture analysis of radiographic images”, Computer Graphics Image Processing, Vol. 8, pp. 447 – 463, 1978.

[Daly93] S. Daly, “The visible differences predictor: an algorithm for the assessment of image fidelity”, in Digital Images and Human Vision, A. B. Watson, Ed., pp. 179-206, MIT Press, Cambridge 1993.

[Karu96] S. A. Karunasekera and N. G. Kingsbury, “A distortion measure for blocking artifacts in images based on human visual sensitivity”, Visual Communications and Image Processing, Proc. SPIE 2094, 474-486 1996

[Dung98] Le Phu Dung, Srinivasan Bala, Mohammed Salahadin, Kulkarni Santosh and Wilson Campbell, “A Measure for Image Quality”, Proceedings of 1998 ACM Symposium on Applied Computing, 513 – 519, 1998

[Irfa04] http://www.irfanview.com/