Dimension reduction of image and audio space

Blaine Lee Hagstrom

University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation

Hagstrom, Blaine Lee, "Dimension reduction of image and audio space" (2000). UNLV Retrospective Theses & Dissertations. 3098.
https://digitalscholarship.unlv.edu/rtds/3098

This Dissertation is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0500

UMI®

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
DIMENSION REDUCTION OF IMAGE
AND AUDIO SPACE

by

Blaine Lee Hagstrom

Associate of Science
Bismarck State College
1991

Bachelor of Arts
University of North Dakota
1993

Bachelor of Science
University of North Dakota
1993

Master of Science, Computer Science
University of Nevada, Las Vegas
1995

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy Degree
Department of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
May 2000

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The Dissertation prepared by

Blaine Lee Hagstrom

Entitled

Dimension Reduction of Image and Audio Space

is approved in partial fulfillment of the requirements for the degree of

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative
ABSTRACT

Dimension Reduction of Image and Audio Space

by

Blaine Lee Hagstrom

Dr. Evangelos Yfantis, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

The reduction of data necessary for storage or transmission is a desirable goal in the digital video and audio domain. Compression schemes strive to reduce the amount of storage space or bandwidth necessary to keep or move the data. Data reduction can be accomplished so that visually or audibly unnecessary data is removed or recoded thus aiding the compression phase of the data processing. The characterization and identification of data that can be successfully removed or reduced is the purpose of this work. New philosophy, theory and methods for data processing are presented towards the goal of data reduction. The philosophy and theory developed in this work establish a foundation for high speed data reduction suitable for multimedia applications. The developed methods encompass motion detection and edge detection as features of the systems. The philosophy of energy flow analysis in video processing enables the consideration of noise in digital video data. Research into noise versus motion leads to an efficient and successful method of identifying motion in a sequence. The research of the underlying statistical properties of vector quantization

iii

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
provides an insight into the performance characteristics of vector quantization and leads to successful improvements in application. The underlying statistical properties of the vector quantization process are analyzed and three theorems are developed and proved. The theorems establish the statistical distributions and probability densities of various metrics of the vector quantization process. From these properties, an intelligent and efficient algorithm design is developed and tested. The performance improvements in both time and quality are established through algorithm analysis and empirical testing. The empirical results are presented.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>v</td>
</tr>
<tr>
<td>CHAPTER 1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>Significant Background</td>
<td>2</td>
</tr>
<tr>
<td>Research Questions</td>
<td>5</td>
</tr>
<tr>
<td>CHAPTER 2 COLOR IMAGERY</td>
<td>7</td>
</tr>
<tr>
<td>Background</td>
<td>8</td>
</tr>
<tr>
<td>Color Spaces</td>
<td>9</td>
</tr>
<tr>
<td>Error Metrics</td>
<td>16</td>
</tr>
<tr>
<td>Methods</td>
<td>19</td>
</tr>
<tr>
<td>Data Structures</td>
<td>20</td>
</tr>
<tr>
<td>Naive vs. Exhaustive Methods</td>
<td>21</td>
</tr>
<tr>
<td>Partitioning</td>
<td>23</td>
</tr>
<tr>
<td>Merging</td>
<td>37</td>
</tr>
<tr>
<td>Iterative Improvement</td>
<td>42</td>
</tr>
<tr>
<td>Image Sequences (Video)</td>
<td>49</td>
</tr>
<tr>
<td>CHAPTER 3 Image Data Characterization</td>
<td>53</td>
</tr>
<tr>
<td>Theory</td>
<td>53</td>
</tr>
<tr>
<td>Video Methods</td>
<td>61</td>
</tr>
<tr>
<td>Still Methods</td>
<td>68</td>
</tr>
<tr>
<td>Summary</td>
<td>73</td>
</tr>
<tr>
<td>CHAPTER 4 VECTOR QUANTIZATION</td>
<td>76</td>
</tr>
<tr>
<td>Background</td>
<td>77</td>
</tr>
<tr>
<td>Codebook Design</td>
<td>77</td>
</tr>
<tr>
<td>Quantization Methods</td>
<td>79</td>
</tr>
<tr>
<td>Problem</td>
<td>83</td>
</tr>
<tr>
<td>Development</td>
<td>84</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Typical RGB noise distribution of a block</td>
<td>62</td>
</tr>
<tr>
<td>3.2</td>
<td>Typical YIQ noise distribution of a block</td>
<td>62</td>
</tr>
<tr>
<td>3.3</td>
<td>Typical YUV noise distribution of a block</td>
<td>62</td>
</tr>
<tr>
<td>3.4</td>
<td>Motion differences from a RGB block</td>
<td>63</td>
</tr>
<tr>
<td>3.5</td>
<td>Motion differences from a YIQ block</td>
<td>63</td>
</tr>
<tr>
<td>3.6</td>
<td>Motion differences from a YUV block</td>
<td>63</td>
</tr>
<tr>
<td>3.7</td>
<td>Autocorrelation distribution from RGB non-motion</td>
<td>64</td>
</tr>
<tr>
<td>3.8</td>
<td>Autocorrelation distribution from YIQ non-motion</td>
<td>64</td>
</tr>
<tr>
<td>3.9</td>
<td>Autocorrelation distribution from RGB motion</td>
<td>64</td>
</tr>
<tr>
<td>3.10</td>
<td>Autocorrelation distribution from YIQ motion</td>
<td>65</td>
</tr>
<tr>
<td>3.11</td>
<td>Motion detection in sample frame using RGB space</td>
<td>65</td>
</tr>
<tr>
<td>3.12</td>
<td>Motion detection in sample frame using YIQ space</td>
<td>67</td>
</tr>
<tr>
<td>3.13</td>
<td>Standard color test image “Lenna”</td>
<td>69</td>
</tr>
<tr>
<td>3.14</td>
<td>Blue band results from featureless region</td>
<td>70</td>
</tr>
<tr>
<td>3.15</td>
<td>Green band results from featureless region</td>
<td>71</td>
</tr>
<tr>
<td>3.16</td>
<td>Red band results from featureless region</td>
<td>71</td>
</tr>
<tr>
<td>3.17</td>
<td>Blue band results from feature region</td>
<td>71</td>
</tr>
<tr>
<td>3.18</td>
<td>Green band results from feature region</td>
<td>72</td>
</tr>
<tr>
<td>3.19</td>
<td>Red band results from feature region</td>
<td>72</td>
</tr>
<tr>
<td>3.20</td>
<td>Y band results from featureless region</td>
<td>73</td>
</tr>
<tr>
<td>3.21</td>
<td>Y band results from feature region</td>
<td>73</td>
</tr>
<tr>
<td>3.22</td>
<td>Still image feature results of Lenna</td>
<td>74</td>
</tr>
<tr>
<td>3.23</td>
<td>Sample frame from video sequence</td>
<td>74</td>
</tr>
<tr>
<td>3.24</td>
<td>Still image feature results of video frame</td>
<td>75</td>
</tr>
<tr>
<td>4.1</td>
<td>Histogram of square Euclid distances</td>
<td>89</td>
</tr>
<tr>
<td>4.2</td>
<td>Histogram of Euclid Distances</td>
<td>89</td>
</tr>
<tr>
<td>4.3</td>
<td>Histogram of minimum distances</td>
<td>89</td>
</tr>
<tr>
<td>4.4</td>
<td>Maximum sign matching distribution</td>
<td>92</td>
</tr>
<tr>
<td>4.5</td>
<td>Sign matching of a single vector</td>
<td>93</td>
</tr>
<tr>
<td>4.6</td>
<td>Sign matching of a single vector</td>
<td>93</td>
</tr>
<tr>
<td>4.8</td>
<td>Histogram of occurrences</td>
<td>97</td>
</tr>
<tr>
<td>4.9</td>
<td>Histogram of occurrences found</td>
<td>98</td>
</tr>
<tr>
<td>4.10</td>
<td>Histogram of probability of finding k amount of</td>
<td>98</td>
</tr>
<tr>
<td>4.11</td>
<td>Wavelet compression results of Lenna</td>
<td>104</td>
</tr>
<tr>
<td>4.12</td>
<td>Brute force search VQ results</td>
<td>104</td>
</tr>
<tr>
<td>4.13</td>
<td>Full search magnitude over 1 VQ results</td>
<td>105</td>
</tr>
<tr>
<td>4.14</td>
<td>Short circuit search magnitude over 1 VQ results</td>
<td>105</td>
</tr>
<tr>
<td>4.15</td>
<td>Rejection method VQ results</td>
<td>106</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

The existence of this work is testament to the hard work and dedication of many incredible people. I take this space to acknowledge those people located both near and far temporally and spatially. Everyone I have ever known personally has either inspired me or driven me to greater heights.

I would like to specifically thank Dr. Yfantis for chairing my committee and advising this work. I would also specifically thank Dr. Larmore, Dr. Bein, Dr. Jeffery and Dr. Stubberud for serving on the committee.

I would like to thank the entire faculty of the Computer Science Department of the University of Nevada, Las Vegas for the formal education in class and personal friendship in general.

I would also like to thank my wife Barbara for her patience, sacrifice and support.

I would like to dedicate this work to the memory of my father Garry Lee Hagstrom and his wife, my step-mother, Sandra Hagstrom whose sacrifice and support enabled me to achieve this work.
CHAPTER 1

INTRODUCTION

The level of computer power now available at the casual user level has reached a point where the data being processed and presented is at a very high volume. Both image and audio data are more common than ever due to the available processing speed and the sophistication level of output peripherals. Large color monitors or displays and audio devices such as compact disk readers, coupled to sound cards which in turn drive high performance loud speaker systems, have opened up a new area for computer application. At the consumer level, home entertainment in the form of game playing was the earliest exploitation of the new advances. This was quickly advanced to very sophisticated interactive applications and the term “multimedia” has become common place. Even with the high performance of today’s existing hardware and the hardware in the near future, the amount of data necessary for high quality imaging, either still or moving, and audio will always be larger than the systems capability. Data compression methodologies help to relieve some of the storage and transfer difficulties but the problem will remain due to increases in data size. The demand for more will always remain. An inherent fact of compression schemes is that the data must be reconstructed into a presentable form. This places demands on the system in addition to the presentation. Dimension reduction, or more commonly, quantization, has been discussed briefly in the texts and literature of the early years of computer science [53]. Technically we are addressing the re-quantization of digital data and not the problem of sampling or quantizing the continuous domain (analog to digital conversion). Briefly, the problem is one of emulating high quality performance.
with limited data. The defined or accepted standard data domain or space is reduced to representative data which still allows acceptable presentation.

**Significant Background**

The increase in amount and type of data being processed and transmitted in today's computer world has provided many problems that may be resolved by current computer science methodology. Image and audio data processing place high demands on resources found in today's computer systems and their inter-connection networks. Reducing the work demands on the hardware relieves the resources for better and faster processing. The processing is not limited to computer systems alone. The digital processing of image and audio data now reaches to the home entertainment arena. Digital Video Disk (DVD), compact disk (CD) in audio, video and computer formats, Internet access and transfer (web browsers, cable modems), and others. The processing of image and audio data also reaches into new and expanding areas such as medicine, finance, security, document and image storage and retrieval along with others, placing very high demands for performance on their systems.

The significance of image processing is obvious in the area of medicine. There are numerous types of medical imaging systems. Nuclear Magnetic Resonance Imaging (NMR or NMRI), Magnetic Resonance Imaging (MRI), Computed Axial Tomography (CAT scan or CT scan), Positron Emission Tomography (PET), Single Photon Emission Computed Tomography (SPECT), Computer Electroencephalograph Tomography (CET) along with X-ray, Ultrasound and others make up a rich and diverse image space domain. One main concern of the medical field is storage and transmission of the data of these images. The storage alone is a problem requiring large resources and now remote access of the data storage is an area of interest and concern for the health professional and the people responsible for the data. Many of the imaging methodologies require specialists in interpreting the data and providing the results to
the attending physician. The field of computer science has provided increased accuracy in the interpretation of these images and further work with the image space and its processing lends itself to the advancement of the use of these imaging systems.

From the area of finance, take the document storage and retrieval area. A typical checking account will serve to illustrate. With the reluctance of banking institutions to cover the cost of returning canceled checks to their customers there is the problem of proof that a check existed if the original is not retained. Microfiche has been a typical answer to the problem but the creation and storage of that medium is presenting problems. Electronic imaging of the document and storage on high volume devices provides an attractive solution both in ease and use. Data retrieval techniques ease the look up situation and the ability to access these data bases through the Internet makes the system attractive. A customer may help themselves after security precautions are satisfied. The ability of the customer to take advantage of such a system would relieve the institution of manpower demands. The problem remaining is that of resolution. A full color scan of each and every check that exists for each and every customer would still be over-taxing. Reduction of the image space to grey scale or bi-level imaging reduces the storage and transmission demands. The quantization to the smaller domain would need to be accurate enough to provide a viable, recognizable and provable image. Thus a reduction solution is necessary.

The finance area also blends into and among the security area. High speed processing of images lends itself to security concerns. Document recognition and proofing would be aided with an image space reduction that preserves the unmistakable and unforgeable features of the document. Audio recognition or voice recognition methodology relies on identifying characteristic patterns within the audio signal. There exist unique, unforgeable, unmistakable characteristics in a voice pattern which exist as a reduced data set of the total space. The characteristics or patterns are referred to as the unvoiced portion of audio as opposed to the voiced portion. Efficient reduc-
tion of the audio space to these patterns or signals would advance the use of such systems in security areas. Finally, the ability to reduce digital video data to a point where normal transmission mediums (phone lines and modems) could be used allows for security applications in every location imaginable. Small digital camera devices capable of high data transmission are an emerging industry demand. The utility of digital video is enticing to those who now use standard video methods.

A common theme in the above applications is data transmission. The recent advent and publicity of the World Wide Web has brought the electronic data domain into the homes of even the very computer illiterate. It has also opened the imagination of the computer literate and the professional in all forms of enterprise. The potential to access large amounts of data has become a focus for many and there is no shortage of information being provided. Internet commerce has provided the ability to open electronic store fronts from the home or existing store. Image data provides a powerful communication medium beyond text description. The psychological impact of seeing what you're buying, even without the tactile feel of the product is still a driving force. The tactile feel of a product has been lost through secure packaging and thus buying from image is a natural extension or evolution in purchasing akin to catalog shopping with the added interactive ability of direct communication with the seller at your finger tips. As this phenomena becomes more common the data traffic demands will grow. Add this on top of the desire to work remotely from the central office and its central data stores by accessing across the open Internet and transmission demands grow even larger. A private closed intra-net can benefit due to the limited bandwidth common in all but the very expensive networks and even a state of the art wide bandwidth network will eventually be over-taxed. A reduction in data would relieve some of the demand. Smaller, perceivably identical image and audio data would transmit quicker and be presented faster.
Research Questions

The dimension reduction is a difficult problem since we are dealing with a multi-dimensional domain. Optimization in multi-dimensions has been shown to be NP-complete and thus optimal solutions, though they are possible, may be infeasible to compute depending on the data sample size. One ambitious study performed a limited exhaustive search of the solution space [60]. Spaulding et al. ran a state of the art tightly coupled multi-processor machine for hours in an attempt to achieve good results. Their methodology is entirely infeasible for the present and near future. The problem solution will necessarily balance quality and time, with quality a priority. A time efficient solution is still desirable for on-line reduction for transfer or presentation on insufficient remote hardware. Preliminary research has demonstrated that superior algorithm development and data structures can improve the time necessary for existing methods. Advancement in computer science since the initial problem formation has produced various improvements, for example the work of Bentley [7] [25]. The quality and efficiency of solutions is still an active research area.

Chapter 2 outlines the current knowledge base in the context of a conceptually tractable application. It includes all aspects necessary to provide a background in data reduction, vector quantization as well as theoretical and mathematical standards that exist in the field. It provides a background in data reduction, vector quantization and the theoretical and mathematical standards. It also provides a background in digital image color space and the nature of the images.

Chapter 3 addresses the problems of video and still image processing more directly. Energy flow analysis is consulted to outline the behavior of lighting in a video sequence. Understanding the complexity of digital imagery at its natural base level is necessary for the development of intelligent processing methods. With the understanding of the imagery in hand, a new method of processing video and still image data is presented and demonstrated. The method developed is useful not only for data
reduction but also for motion detection in video and is extended to feature identification in still image methods. The results of this chapter provide the characterization of data into necessary and unnecessary or redundant.

Chapter 4 directly addresses the vector quantization methods. A theoretical analysis of the vector quantization method is performed and tested in simulation. The statistical properties of the vector quantization process are explored and three theorems are developed and proven. The analysis provides characterizations of the method which indicates the performance behavior. This behavior is exploited to develop improved methods in terms of both efficiency and quality. The results of the simulation are presented and an application based on the findings is developed. Results of the application of the developed methods are presented.

Chapter 5 provides a summary and future research plan. The Appendix contain selected source code from the applications developed and presented.
CHAPTER 2

COLOR IMAGERY

The creation and use of color images on computer systems has a long and rich history. This dissertation is focused on digital images of real events either recorded with digital equipment or transferred from analog images. Even with that restriction, the knowledge base is vast. This chapter presents the most relevant material to provide the background for the research advancements described in the following chapters. The focus of the research is to provide improvements to the existing methods of using and storing digital images and sound. The inherent nature of quality digital images and sound is that they require large amounts of data. Since the supply of data is infinite, improvement comes from reduction of the data while maintaining quality.

Quantization may be a form of compression. However, compression can occur without quantization and always involves de-compression. Quantization will always cause compression, if done properly, but may not involve de-quantization. The most successful quantization compression is a vector quantization (VQ) which processes groups of data at a time. The paradigm is to match a vector of test data to a vector from a fixed set of vectors contained in a codebook. This matching is based on a nearness metric. The data is coded as the index to the best vector in the codebook. Reduction occurs by keeping the number of codebook vectors limited so that the index space is much smaller than the vector space. A simple vector of length 1, a scalar, is typically very easy to match. Added length or dimensions complicate the procedure. Typical vector lengths are becoming increasingly long in the push for high data reduction. Dimensions beyond 3 are difficult to envision. Fortunately great
interest lies in data reduction for color image data where 3 dimensions are typical due to the color dimension, red, green and blue. A large amount of study has been performed in reducing the data in the color dimensions and from that we can get a good background understanding. The following is a synopsis of color quantization methods and theories gleaned from the body of literature available at the time of this writing. The details are extensive and are reflected here in a brief manner. The formal notation of the various publications differs and an attempt to normalize notation is included. Although the discussion is specific to color quantization, the techniques apply to other domains and the theory scales to higher dimensions.

Background

Color quantization denotes a specific method of the more general vector quantization. A subset of vectors of appropriate length is used to represent the desired space. Actual data vectors, test vectors, are mapped to these representatives in a closest possible approximation manner. The design of the codebook itself is a rich area of study and many concerns are illustrated in the process of color quantization.

There are existing transforms which aid in processing and characterization of data. Various color space metrics and transforms from one to another exist. Spaces such as Red-Green-Blue (RGB), National Television standard (YIQ or YUV), Com­mission Internationale de l’Eclairage standards (CIE Luv and Lab) and others. Colorimetry is itself a science that has a rich history, see for example [71].

The advent of digital images presented many problems for both capturing an image and displaying that image. Various hardware problems on the capturing end were overcome to provide a large enough color space so that the images could provide more color than the human eye could detect. There exist various color spaces in which data may be represented. All spaces used in color quantization consist of three major dimensions. Many of these spaces are referred to as tristimulus, most notably the red,
green, blue space commonly referred to as RGB. In order to provide a large enough space to cover the human visual system (HVS), a size of 256 levels for each of the dimensions was adopted. This provides for \(2^{24} = 16,777,216\) colors possible. Research estimates of the HVS range from less than 350,000 colors \([65]\) to as many as 10,000,000 colors \([69]\). Thus a 24 bit color space is ought to be more than adequate to provide high quality images for human viewers. A problem arises from the fact that 24 bits or 3 bytes of computer storage must be provided for each element or pixel of an image. With images increasing in size and number, this becomes a burden on the computer memory, transmission, storage, display and processing systems. At a VGA display size of 640 by 480 pixel resolution, this demands 921,600 bytes of resource. Even with the cost of storage, main memory and video adapters falling and the increase of processing power, storage access and transmission speed (bandwidth), resources can still be overtaxed by animation and multiple display. Standardized compression methods like JPEG for still images and MPEG for motion sequences (movies and animation), have helped to reduce the transmission and storage problems by placing more burden on the processing systems. The problem still remains of presenting the images on restricted display systems. To address the display problem the concept of color palette or lookup tables was developed and adopted. In this system an index from a much smaller domain is used to reference the palette or table which contains representative values from the larger image domain. The problem is to find the best representative colors to place in these tables so that the displayed image is as accurate as possible. Thus the original color space is quantized into the smaller space. This methodology is an exact application of vector quantization.

Color Spaces

Color science has roots dating to the early part of the twentieth century. It is a very broad science and goes far beyond the scope of this work. We shall provide a
very brief introduction. Color measurement is based on energy wavelength. A color space basis standard was established in 1931 by the Commission Internationale de l'Éclairage (CIE) to address the problems caused by the color matching based on wavelengths [71]. The CIE standard established a continuous function system $x_\lambda$, $y_\lambda$, and $z_\lambda$ with the three components XYZ to replace red, green and blue.

$$X = k \int P(\lambda) x_\lambda d\lambda, \quad Y = k \int P(\lambda) y_\lambda d\lambda, \quad Z = k \int P(\lambda) z_\lambda d\lambda,$$

where $k$ is 680 lumen/watt for self luminous devices or is based on a normalizing function for reflective items so that bright white has values of 100. XYZ components are weighted and added together to match any given color and the use of it in computer systems is usually through a look up table since these are continuous functions. A problem with this color space is that a small and equal change in dimensions on two colors from this space may not be perceived as equal in amount. In 1976 the CIE LUV ($L^*, u^*, v^*$) space and the CIE LAB ($L^*, a^*, b^*$) space were developed to address this problem. The formulae for LUV are readily available [24].

$$L^* = 116 \left( \frac{Y}{Y_n} \right)^{1/3} - 16, \quad \frac{Y}{Y_n} > 0.01$$

$$u^* = 13L^*(u' - u'_n)$$

$$v^* = 13L^*(v' - v'_n)$$

$$u' = \frac{4X}{X + 15Y + 3Z}, \quad v' = \frac{9Y}{X + 15Y + 3Z}$$

$$u'_n = \frac{4X_n}{X_n + 15Y_n + 3Z_n}, \quad v'_n = \frac{9Y_n}{X_n + 15Y_n + 3Z_n}$$

where $(X_n, Y_n, Z_n)$ is the coordinate defined as white. A more complete set of formulae is also available along with diagrams of the color space [71].

$$L^* = \begin{cases} 
116 \left( \frac{Y}{Y_n} \right)^{1/3} - 16, & \frac{Y}{Y_n} > 0.008856 \\
903.29 \left( \frac{Y}{Y_n} \right), & \frac{Y}{Y_n} \leq 0.008856 
\end{cases}$$

$$u^* = 13L^*(u' - u'_n)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[ v^* = 13L^*(v' - u'_n) \]

\[ u' = \frac{4X}{X + 15Y + 3Z}, \quad u_n' = \frac{4X_n}{X_n + 15Y_n + 3Z_n}, \quad v' = \frac{9Y}{X + 15Y + 3Z}, \quad v_n' = \frac{9Y_n}{X_n + 15Y_n + 3Z_n} \]

The formulae for LAB space is also listed in two forms [71].

\[ L^* = 116 \left( \frac{Y}{Y_n} \right)^{\frac{1}{3}} - 16 \]

\[ a^* = 500 \left[ \left( \frac{X}{X_n} \right)^{\frac{1}{3}} - \left( \frac{Y}{Y_n} \right)^{\frac{1}{3}} \right] \]

\[ b^* = 200 \left[ \left( \frac{Y}{Y_n} \right)^{\frac{1}{3}} - \left( \frac{Z}{Z_n} \right)^{\frac{1}{3}} \right] \]

\[ \frac{Y}{Y_n}, \frac{X}{X_n}, \frac{Z}{Z_n} \geq 0.01 \] (2.4)

and

\[ L^* = \begin{cases} 
116 \left( \frac{Y}{Y_n} \right)^{\frac{1}{3}} - 16, & \frac{Y}{Y_n} > 0.008856 \\
903.3 \left( \frac{Y}{Y_n} \right), & \frac{Y}{Y_n} \leq 0.008856 
\end{cases} \]

\[ a^* = 500 \left[ f \left( \frac{X}{X_n} \right) - f \left( \frac{Y}{Y_n} \right) \right] \]

\[ b^* = 200 \left[ f \left( \frac{Y}{Y_n} \right) - f \left( \frac{Z}{Z_n} \right) \right] \]

\[ f \left( \frac{X}{X_n} \right) = \left( \frac{X}{X_n} \right)^{\frac{1}{3}}, \quad \frac{X}{X_n} > 0.008856 \]

\[ f \left( \frac{X}{X_n} \right) = 7.787 \left( \frac{X}{X_n} \right) + \frac{16}{116}, \quad \frac{X}{X_n} \leq 0.008856 \]

\[ f \left( \frac{Y}{Y_n} \right) = \left( \frac{Y}{Y_n} \right)^{\frac{1}{3}}, \quad \frac{Y}{Y_n} > 0.008856 \]

\[ f \left( \frac{Y}{Y_n} \right) = 7.787 \left( \frac{Y}{Y_n} \right) + \frac{16}{116}, \quad \frac{Y}{Y_n} \leq 0.008856 \]

\[ f \left( \frac{Z}{Z_n} \right) = \left( \frac{Z}{Z_n} \right)^{\frac{1}{3}}, \quad \frac{Z}{Z_n} > 0.008856 \]

\[ f \left( \frac{Z}{Z_n} \right) = 7.787 \left( \frac{Z}{Z_n} \right) + \frac{16}{116}, \quad \frac{Z}{Z_n} \leq 0.008856 \] (2.5)
Because the focus of color quantization is on producing acceptable images on a color computer monitor, we shall work from its color space and show the transforms to other color spaces including the CIE standard.

The two major image display systems are either color monitor displays which work with RGB phosphors on the screen and thus use the additive primary system of RGB, or hard copy or printing devices which work with cyan, magenta and yellow (CMY) as subtractive primaries. Note that many printing devices also include a black due to the limits of blending CMY to create black. These spaces are a simple cube shape and are easily represented as a 3D space with RGB axes. The two systems are related by a simple inverse computation.

\[
\begin{bmatrix}
C \\
M \\
Y
\end{bmatrix} = \begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix} - \begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

(2.6)

Although an image palette is eventually coded in RGB space, since monitor phosphors are RGB, there exist many other color spaces which are either linearly related to RGB or otherwise obtainable from RGB through non-linear transformation or table look up methods. These spaces provide different advantages to processing. The most notable reason to work in another color space is to achieve a uniform color space so that the Euclidean distance metric is a better measure of error between original data and quantized data. Another reason is to provide a system where an image may be produced on a monochrome device. A brief outline of the various popular color spaces is provided with the transforms to these spaces. The simplest transforms are linear of the form

\[
T_1 = C_{11}R + C_{12}G + C_{13}B \\
T_2 = C_{21}R + C_{22}G + C_{23}B \\
T_3 = C_{31}R + C_{32}G + C_{33}B
\]

(2.7)

where \(C_{ij}\) is a constant and \(R, G, B\) are the RGB components. Another common transformation is based on a luminance or brightness component. The RGB components are used to extract a gray scale value which indicates the relative brightness of
the pixel and the coloration of the pixel is carried in two chromaticity components. These transforms are generally of the form

\[
Y = C_1 R + C_2 G + C_3 B \\
t_1 = \frac{T_1}{T_1 + T_2 + T_3} \\
t_2 = \frac{T_2}{T_1 + T_2 + T_3}
\]

(2.8)

A remaining less popular type is some general non-linear invertible form that differs from the above type [52].

The foremost alternate color space is the National Television System Committee (NTSC) transmission space. This is the U.S. color television transmission standard developed in 1951 to allow monochrome or black-and-white televisions to receive and display the signal. The conversion is a linear transform from RGB. For the normalized space with \(0 \leq R, G, B \leq 1\) the conversion is given, [52]

\[
\begin{bmatrix}
Y \\
I \\
Q
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
0.596 & -0.274 & -0.322 \\
0.211 & -0.253 & 0.312
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

(2.9)

or as [24]

\[
\begin{bmatrix}
Y \\
I \\
Q
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
0.596 & -0.275 & -0.321 \\
0.212 & -0.528 & 0.311
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}.
\]

(2.10)

A closely related space is referred to as YUV with identical Y band transform

\[
\begin{bmatrix}
Y \\
U \\
V
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
0.701 & -0.587 & -0.114 \\
-0.299 & -0.587 & 0.886
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}.
\]

(2.11)

A second proposed television standard from the Society of Motion Picture and Television Engineers is referred to as SMPTE and is given as [6]

\[
Y = 0.299R + 0.587G + 0.114B \\
C_R = 0.713(R - Y) + 128 \\
C_B = 0.564(B - Y) + 128
\]

(2.12)

where the domain is [0,255] for each dimension in RGB.

The direct conversion from RGB to XYZ is a transform of the form

\[
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} =
\begin{bmatrix}
X_R & X_G & X_B \\
Y_R & Y_G & Y_B \\
Z_R & Z_G & Z_B
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

(2.13)
where $X_i, Y_i, Z_i$ are the appropriate weight for a given device. These are device specific. They are either provided by the manufacturer or obtainable by measurement with a colorimeter or a spectroradiometer. There are methods for determining the values by working with a photometer and known color values [24]. Due to the device dependency, these transforms are more exacting than is useful for general color quantization implementation. There exists an easily implemented CIE tristimulus uniform chromaticity scale developed empirically and given as [52]

\[
\begin{bmatrix}
U \\
V \\
W
\end{bmatrix} =
\begin{bmatrix}
0.405 & 0.116 & 0.133 \\
0.299 & 0.587 & 0.114 \\
0.145 & 0.827 & 0.627
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}.
\] (2.14)

Note that $V = Y$ between the YIQ and CIE-UVW. This conversion provides for just noticeable changes in chromaticity for incremental changes in the tristimulus values. A further computation results in a another CIE uniform space

\[
Y = 0.299R + 0.587G + 0.114B
\]

\[
u = \frac{U}{U+V+W}
\]

\[
v = \frac{V}{U+V+W}
\] (2.15)

and a general non-linear invertible transform based in CIE

\[
U^* = 13W^*(u - 0.201)
\]

\[
V^* = 13W^*(v - 307)
\]

\[
W^* = 25(100Y)^\frac{1}{3} - 17, \ 0 \leq Y \leq 1
\] (2.16)

There are also color spaces based on Hue (the basic color), Saturation (the distance from gray of equal intensity) and Lightness or Brightness (the perceived intensity). Two major forms are Hue, Saturation and Value (HSV) also called Hue, Saturation and Brightness (HSB) and Hue, Lightness and Saturation (HLS) also called Hue, Saturation and Lightness (HSL). We shall use HSV and HSL notation. The spaces are hexagonal cone shaped and use a polar coordinate type system. Algorithms to convert from RGB to HSV or HSL and back are given in the reference [24] along with the diagrams: A closed form for HSV would be

\[
V = \max(R, G, B)
\]
\[ t = \min(R, G, B) \]
\[ S = \begin{cases} \frac{V-t}{V}, & V \neq 0 \\ 0, & V = 0 \end{cases} \]
\[ H = \begin{cases} 
0, & S = 0 \\
60 \left( \frac{G-B}{V-t} \right), & R = V, G \geq B \\
60 \left( \frac{G-B}{V-t} \right) + 360, & R = V, G < B \\
60 \left( 2 + \frac{B-R}{V-t} \right), & G = V, B \geq R \\
60 \left( 2 + \frac{B-R}{V-t} \right) + 360, & G = V, B < R, \left| \frac{B-R}{V-t} \right| > 2 \\
60 \left( 4 + \frac{R-G}{V-t} \right), & B = V, R \geq G \\
60 \left( 4 + \frac{R-G}{V-t} \right) + 360, & B = V, R < G, \left| \frac{R-G}{V-t} \right| > 4 
\end{cases} \] (2.17)

A closed form for HSL would be

\[ A = \max(R, G, B) \]
\[ C = \min(R, G, B) \]
\[ L = \frac{A+C}{2} \]
\[ S = \begin{cases} 0, & A = B \\
\frac{A-B}{A+B}, & L \leq 0.5 \\
\frac{B-A}{B}, & L > 0.5 \end{cases} \]
\[ H = \begin{cases} 
0, & A = C \\
60 \left( \frac{G-B}{A-B} \right), & R = A, G \geq B \\
60 \left( \frac{G-B}{A-B} \right) + 360, & R = A, G < B \\
60 \left( 2 + \frac{B-R}{A-B} \right), & G = A, B \geq R \\
60 \left( 2 + \frac{B-R}{A-B} \right) + 360, & G = A, B < R, \left| \frac{B-R}{A-B} \right| > 2 \\
60 \left( 4 + \frac{R-G}{A-B} \right), & B = A, R \geq G \\
60 \left( 4 + \frac{R-G}{A-B} \right) + 360, & B = A, R < G, \left| \frac{R-G}{A-B} \right| > 4 \end{cases} \] (2.18)

There is a cylindrical color space model with a closed form for HSL given also [40], however the formulation cannot produce a cylinder span.

\[ L = 0.299R + 0.587G + 0.114B \]
\[ S = \left[ \left( \frac{R-L}{1.14} \right)^2 + \left( \frac{B-L}{2.03} \right)^2 \right]^\frac{1}{2} \]
\[ H = \arctan \left( \frac{0.562B-L}{R-L} \right) \] (2.19)

Extensive coverage of the color spaces and color science exist and may be found in the literature. Early works addressing quantization of signals such as Stenger [61]
deal more with the interdependencies of the various color space components and their
effect in television and on HVS along with the non-linearity inherent in the systems.

Error Metrics

The nature of the solution to the quantization problem is that an approximation
is made to the true data or image. Because of this approximation, there naturally
must be some form of error measurement. The difference between the approximation
and the true image is an error. The qualitative measurement of this error is a difficult
problem since there exists no known metric for the HVS. There are several measure­
ments based on various existing space metrics and some work has been done in an
attempt to provide a qualitative method for perceived quality. A list of the standard
space metrics and the computations used to provide an error metric are presented.
The perceived quality computation methods will follow.

The 3 dimensional nature of color space naturally lends itself to normal 3 di­
dimensional calculation based on distance. The two most common metrics used are
Euclidean distance

\[ D_E (x_i, x_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \]  

(2.20)

and Manhattan or grid distance

\[ D_M (x_i, x_j) = |x_i - x_j| + |y_i - y_j| + |z_i - z_j| \]  

(2.21)

A common efficiency method is to avoid the square root calculation of \(D_E\) and simply
use the sum of squares distance

\[ D_S (x_i, x_j) = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \]  

(2.22)

There is also a specific distance or distortion measure for the YUV color space [36]

\[ D = \frac{5}{8} (y_i - y_2)^2 + \frac{3}{16} (u_1 - u_2)^2 + \frac{3}{16} (v_1 - v_2)^2 \]  

(2.23)
We shall simply use $D$ to indicate any distance measure in the following equations and identify the specific method where needed. Based on these distance measures, there are various error calculations that may be employed. Let $N$ be the total number of color values of a given image. Let $K$ be the number of representative colors that are determined by the quantization process ($K < N$). Let $Q$ be the quantization function and $q_k = Q(x)$ where $q$ and $x$ may be scalar or vector quantities. Let $P(x)$ be the discrete density function where $x$ is again scalar or vector. Let $E$ be the error function. We shall use bolding to indicate vector or matrix variables, i.e., $x = (x_i, x_j, x_k)$ with $i, j, k$ representing color space dimensions. A very typical color quantization process is to partition the space into $K$ regions or subsets $s_i$ such that $S = \{s_1, s_2, \ldots, s_k\}$ and

$$q_k = \frac{\sum_{x_i \in s_k} x_i}{|s_k|} \quad (2.24)$$

where $|s_k|$ is the size or number of elements in $s_k$. It is the partitioning of the color space where the difficulty is encountered. Once the partitions are determined, the mean or centroid of a partition causes the least error.

The most popular error metric is the Mean Square Error (MSE) also referred to as Mean Absolute Error (MAE) [21]

$$MSE = \frac{1}{N} \sum_{i=1}^{N} D(x_i, Q(x_i)) \quad (2.25)$$

which is an average error term for the entire image or data set. A modified version of this is used to avoid the square root calculation and is referred to as Total Square Error (TSE) [43][49][9][5], Total Sum of Squared Error (TSSE) [74] and confusingly Mean Square Error (MSE) [21]

$$TSE = \frac{1}{N} \sum_{i=1}^{N} D(x_i, Q(x_i))^2 \quad (2.26)$$

There is also a slight variation of these called Quantization Mean Square Error and Quantization Root Mean Square Error [57]

$$QMSE = \left[\frac{1}{N} \sum_{i=1}^{N} D(x_i, Q(x_i))^2 \right]^{\frac{1}{2}} \quad (2.27)$$
A modification to all these equations is to rewrite them using the probability density function and define the error $E$ as

$$E = \sum_{i=1}^{N} P(x_i) D(x_i, Q(x_i))$$  \hspace{1cm} (2.28)

A common metric that does not appear much in color quantization but is used extensively in image processing is the peak signal to noise ratio. It is included here for completeness. This value is usually employed to demonstrate the quality of the proposed method.

$$PSNR = 10 \log_{10} \frac{\max(x) - \min(x)}{MSE}$$ \hspace{1cm} (2.29)

These all have a distance measure as a basis of error calculation and thus the distance between colors in a given space becomes a concern. The ideal situation is that an equal distance between any two colors causes an equal change in the perceived image. Another desired property is that a unit change in a color space causes a just perceivable change in the image. These are goals for color space research.

There are problems with designing a solution based on minimizing values calculated by the above formulae. The foremost is that the calculations have no basis in HVS. They are valid in comparing the performance between different methods but fall short on giving a valid performance standard for images. Two major problems with optimizing based on these measures are gross quantization of exceptional colors and contouring or Mach banding.

The first, gross quantization of exceptional colors, is how colors that occur infrequently are handled. These are usually highlights caused by specular reflection or are colors that represent very important small details such as eye color in portrait images. Since the colors occur infrequently $P(x)$ is very small and thus plays a minor part in the error calculations above. The problem is that importance is not based on visual quality. The second, contouring or Mach banding, is cause by smooth gradient areas quantizing to a few representative colors and a noticeable band or step in the
color is produced. Once again, because importance is based on occurrence rather than on visual quality, perceived image quality suffers. There has been some work done towards addressing this problem but the lack of a qualitative system based on the HVS hinders this type of problem solution. The solutions addressing these problems are context oriented. A local block is used around a pixel to weight its importance or to give it a value which indicates how it is used. These are pre-processing methods. Post-processing methods such as dithering help the contouring problem.

Methods

The manner of quantizing from the large original space into the small representative space may be very simple and quick or complex and slow. The design of the method may be based on different error priorities. The largest and most difficult problem is that there is no existing metric to quantify the HVS. The final presented image is subject to viewer critique and no known method of computing quality according to perceived image exists. Since attempting to model or account for HVS and perceived quality is the most difficult problem it is addressed in the more complex methods. We shall start from the simplest and move to the more involved methods.

The field of vector quantization codebook design may be crudely divided by basic method. Two major method categories are data independent and data dependent design. The advantages of either method are obvious. In independent methods all the work is done once and used repeatedly. Independent methods may be very naive intuitive approaches or may include some intelligent methodology such as statistical argument, although examples of this are rare in the literature. Independent methods do not introduce a processing burden at display time and thus are very fast. Dependent methods make use of the actual image itself during its quantization. Dependent methods are more accurate since resources are not provided for unused solution space. Dependent methods may also be naive intuitive approaches or may be very involved.
and complex. Since there is a quality advantage in dependent methods, all known existing research takes this approach. A major division in these research and design methods also exists. Two basic approaches used are either a divisive methodology or an amalgamative methodology. The basic approach is to either start with one large group and split into smaller groups recursively until the goal is attained or start with all individual elements and combine groups recursively until the goal is attained. In either case similar decision processes may be employed. Another major division in methodologies is in data handling. One approach works with the raw data directly and the other works from a histogram of the data.

Most of the research in vector quantization is aimed at codebook design or error control. Since higher compression is possible with longer vectors, the field is moving in that direction. Most common practices in high data reduction are centered around linear predictor methods with the error of the prediction being subject to vector quantization. A field that explores codebook design in great detail is color quantization research. The general methods of vector quantization are used along with special techniques adapted to the color quantization problem. An understanding of color space is also provided by studying the color quantization methods.

Data Structures

Before presenting the various methods for quantization a short coverage of common data structures is appropriate. These are efficient ways of handling the data during the quantization solution search. These structures may hold the raw data itself or histogram information on the data. One of the most common is a structure called a k-dimensional (k-d) tree. This is a binary tree that partitions a k-dimensional space with (k-1)-dimensional hyper-planes. The internal nodes contain the decision factor that divides the data across the hyper-plane. The structure was developed for vector data and a node causes a division based on one component of the vector.
Implementation is user defined but a general practice is to use the vector components in a cyclic fashion. Another practice for quantization methods that aim to minimize error functions is to use components in a manner based on data variance. The idea being to spread out the data in the leaves thus providing a better chance that they are close in their space. The leaves of the tree contain the data itself. Building the tree usually consists of taking the entire data set and placing it at the root. The set is divided into left and right children recursively until the tree contains the desired number of leaves, i.e., the set is partitioned into the desired number of regions. A problem with using these structures for the design of an algorithm is that they provide localized optimal solutions. They are greedy algorithms that make a best choice at each given step and therefore do not necessarily provide the globally optimal solution.

A second very common data structure is a 2-dimensional array of tree or linked list pointers. The 2 dimensions are two of the data dimensions and the linked list or tree is used to hold the data arranged according to the third dimension. In this manner a direct lookup based on two of the data dimensions is possible and a short search for the third dimension to locate the value. This relieves the memory burden of an entire third dimension. For example, 24 bit image data may be stored in $O(256^2)$ instead of $O(256^3)$, a significant savings. A problem with this method is that is does not scale up to higher dimensions.

**Naive vs. Exhaustive Methods**

The absolute simplest vector quantization is uniform quantization. In an independent manner, this is simply dividing the total space into equal sized sub-spaces and placing the image data into the sub-space that brackets the datum value. The datum bits are shifted right so that the final bit length is the predetermined dimension size. For example, from $2^{24}$ color space, by shifting each dimension 5 places right you achieve $2^9$ space or 512 colors. To reach 256 colors, one dimension is shifted one bit.
more. Typically the blue dimension is shifted farther due to the lessened sensitivity in the HVS to the blue color space [23] or the non-uniform aspect of the RGB color space [37]. This method of bit shifting or space dividing has been used to pre-quantize data to reduce storage requirements for more advanced algorithms. The reduction is usually on the order of 2 or 3 bits per dimension resulting in 262144 and 32768 colors respectively and the claim is that the HVS does not perceive more than this many levels [59]. Note that creating a histogram from the data is also a data storage reduction method. The simple bit shifting process, performed for whatever reason, is entirely too crude to preserve image quality without extensive processing of the resulting data. Too much information is simply cast away that could aid in better quantization. An extensive coverage of uniform image independent quantization in various color spaces is provided by Gentile et al., [27].

Perhaps the most fundamental vector quantization method is known as the LBG algorithm from the principal designers, Linde, Buzo and Gray. It also called the K-means algorithm. It is directly applicable to color quantization and iteratively improves a quantization codebook based on some error metric. It is simple in design. First an initial codebook or quantization is selected

\[ \{q_n : 1 \leq n \leq K \} \] (2.30)

then \( K \) clusters are formed according to the quantization

\[ C_k = \{x_n : ||x_n - q_k|| \leq ||x_n - q_j|| , 1 \leq j \leq K \} \] (2.31)

The codebook is recomputed based on the clusters

\[ q_k = \frac{1}{|C_k|} \sum_{x_k \in C_k} x_k \] (2.32)

The cluster forming step and codebook re-computation steps are repeated until no more error improvement occurs or a threshold in error is reached. This method is similar to hill climbing or steepest descent methods from AI theory and suffers from
the same problems. A local optimum will be obtained that may or may not be
globally optimum. A poor initial selection may cause prohibitive time demands to
convergence. However, this method may be used as a post-processing scheme once a
quantization is determined by other methods.

Partitioning

Although Heckbert [34] is usually given as a reference for the beginning of color
quantization, the mathematical theory of vector quantization and solutions date much
earlier along with older color quantization focus. Pratt has addressed color quantiza-
tion as either a topic directly or a part of a compression scheme years before Heckbert.
Pratt’s earlier work [52] dealt with transform coding of images for compression and
includes a good general coverage of transform methods between color spaces. His
book [53] deals with all aspects of digital image processing and mentions quantiza-
tion of monochrome and color image data. It covers quantization in general based
on probability density for the scalar case and briefly describes the difficulty with the
vector case. For the scalar case quantization may be performed in the uniform man-
ner or in a non-uniform manner. Non-uniform may be either directly with decision
levels based on the distribution or through a non-linear transform and then a uniform
quantization. For the vector case an iterative method for obtaining a sub-optimal
solution is given. Since the methods rely heavily on distribution assumptions they
may be categorized as independent methods.

Heckbert’s work is a simple approach in RGB space and uses $D_S(x_i, Q(x_i))$ as
the error metric. To conserve memory he implements a pre-quantization step that
shifts each color dimension 3 bits left. His first work was the implementation of a
Popularity algorithm developed independently by two groups in 1978: Tom Boyle and
Andy Lippman at MIT and Ephraim Cohen at the New York Institute of Technology.
Heckbert’s implementation was the publication of the MIT development and the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
latter is unpublished. The algorithm works from the image histogram and simply uses the most frequently occurring colors as the quantization values and maps image colors to the closest quantization value by setting decision points half way between quantization values. This implementation is $O(NK)$. Since sparsely occurring values suffer from this method some modification to the basic method were developed. One modification was to reduce the frequency of nearby values when a quantization value was determined thus allowing less frequent colors a better chance of being represented. Heckbert proposed another method in the first work and presents it in the second. The Median Cut Algorithm works from the color histogram and divides the color space orthogonal to the dimensions into regions that contain roughly equal numbers of values. It may be implemented recursively starting with the smallest space that encloses the image data and dividing along the longest axis. Until the desired number of regions is reached, all subspaces are considered and the one with highest pixel count is then divided along its longest axis. A correct recursive implementation divides the desired region number in half on each phase. A failure of a recursive implementation is that equal numbers of pixels may not fall on each side of the division. The algorithm has a check for splitting a single pixel subspace. The centroids of the resulting regions are then chosen as the quantization values and nearest neighbor rules are used to quantize the data. An iterative procedure is mentioned to improve the result of both the Popularity algorithm and the Medium Cut algorithm. This is the LBG vector quantizer. Wu [69] outlines Heckbert's work and identifies it as a $k - d$ tree recursive cutting method which Heckbert acknowledges but did not fully investigate before publishing. Wu also ties Heckbert's work to histogram equalization and identifies Stevens et al [62] work with Peano curves as a histogram equalization method. The histogram equalization suffers from the infrequent occurrence problem.

A compression scheme was proposed by Campbell et al., [11] that performs color quantization as part of the compression. The image is divided into 4 X 4 blocks and for
each pixel the luminance computation from RGB is performed. The mean luminance for each block is used to partition the data in the block into two sets, those pixels that have a luminance above the mean and those below or equal to the mean. These two sets are then processed to produce a mean RGB value for each block partition. The entire set of means from all the image blocks are then bit shifted left 3 bits. Finally all the partition representative colors are processed with Heckbert’s algorithm to choose 256 best colors and produce a look up table. The 24 bit partition representative colors are then replaced with an 8 bit look up table index. Since quantization was the goal, the colors are handled in a cruder manner than Heckbert’s original algorithm and thus cannot produce better perceived images.

Voloboj [65] makes observations on Heckbert’s Median Cut algorithm and proposes an improvement. One observation is that Heckbert only used $D_S(x_i, Q(x_i))$ as an error metric and that another metric should be considered in conjunction with the original. The new metric is a maximum deviation measure

$$E_D = MAX (\max(\Delta r, \Delta g, \Delta b))$$

(2.33)

where $MAX$ is the maxima of all pixels of the image and max is the maxima coordinate deviation of a pixel. The other observation is that the HVS does not distinguish more than 350,000 colors as noted above and that implies about 48 gradations on average for each basic color. This value may range between 70 to 35 gradations with good results. The point being that it is pointless to further divide a subspace below this level of diameter. The proposed algorithm divides a subspace orthogonally based on the arithmetic average of the three coordinates and counts quantities on points on each side of each average coordinate. The axis is chosen so that the split places pixels on either side as close to equal as possible. This addresses the $D_S(x_i, Q(x_i))$ metric. To address the $E_D$ metric, Voloboj suggests an initial step that uniformly divides the entire color space into 64 parts, 4 parts per dimension. This places a limit on $E_D$ of 0.25. The remainder of the paper contains implementation considerations.
A simple method of space splitting is a greedy tree growing algorithm proposed by Liu and Chang [43] where the quantization is based on a binary tree method and an error minimization method. Working from TSE they derive the equation which stipulates the change in TSE when a set is split.

\[ E_n = N_{nl} \| \bar{X}_{nl} - \bar{X}_n \|^2 + N_{nr} \| \bar{X}_{nr} - \bar{X}_n \|^2 \]  

(2.34)

where \( N_{nl} \) and \( N_{nr} \) are the number of elements in the left and right set respectively, \( \bar{X}_i \) is the centroid of node \( i \). Note that \( n \) splits into \( nl \) and \( nr \) as left and right children in a binary tree. The algorithm checks each axis distribution and the resulting split based on the mean as a division point. The axis with the largest contribution to error reduction is the chosen one. All leaf nodes contain a calculation of their contribution to error reduction and the maximum is chosen for the split and the children's contribution calculated. The centroids of the leaves are the quantization values. This results in an \( O(N) \) time complexity. Their results are compared with LBG at 20 iterations using \( RMSE \). The algorithm is presented again [44] with acknowledgment of contouring and proposed improvements through morphological operations and error diffusion. Basically the error diffusion is performed where edges are detected with an erosion type operation. This corrective methodology was proposed at an earlier date by Bouman and Orchard and is discussed later with their algorithm.

Joy and Xiang [37] propose modifications to Heckbert's Center Cut algorithm. The first is to base the next subspace chosen for division on the maximum longest dimension of all subspaces. The second modification is to use a center cut instead of a median cut to divide the subspace, presumably through the longest axis. The last modification is to change the 3 bit uniform shift performed to conserve memory to a 3-2-4 bit shift for RGB respectively. This results in the same memory requirements but partially compensates for the non-uniform nature of the RGB color space. It also correlates with the luminance computation. They do warn that this will cause problems because the subspace divisions are now biased towards green foremost and
red secondly. They recommend calculating subspace centroids based on unabridged data to overcome this problem.

Balasubramanian et al., [6] along with Pei and Cheng [50] developed in parallel a method of dividing the color space by sequential scalar quantization. The method processes each dimension separately. They work in SMPTE color space as given above. After transforming from RGB to YCrCb, the least significant bit of Cr and Cb is dropped (right shift 1 bit). To further reduce computational load they subsample by 2 in each spatial direction. They use a 2D array of linked lists for histogram storage for both speed and size considerations. The data structure has become a common storage method for image quantization methods. The objective of the algorithm is to choose how many bits to allocate for each dimension based on image statistics. The method quantizes the first dimension into \( N_1 \) regions \( B_{2j} \) based on the marginal distribution \( p(x_1) \). This gives columnar regions in \( R^2 \). It then quantizes each region \( B_{2j} \) along the second dimension into \( n_2 \) regions \( B_{3kJ} \) based of the conditional distribution \( p(x_2|B_{2j}) \). This results in \( N_2 \) regions in \( R^2 \) and columnar regions in \( R^3 \). Finally it quantizes along the third dimension in each \( B_{3kJ} \) based on the conditional distribution \( p(x_3|B_{3kJ}) \) to produce \( N_3 = N \) regions. It then chooses the centroid as the quantization value for each region. The method attempts to minimize MSE with a proper choice of \( N_1 \) and \( N_2 \). The development of their method used asymptotic theory which assumes that there is a large number of quantization regions and that the distribution of the data is relatively smooth. From this they reach an optimal set of equations

\[
N_1 = N^{\frac{1}{4}} \left( \frac{\alpha^2}{\beta \eta} \right)^{\frac{1}{8}}, \quad N_2 = N^{\frac{3}{4}} \left( \frac{\alpha \beta}{\eta^2} \right)^{\frac{1}{4}}
\]  
(2.35)

The actual values of \( \alpha, \beta, \eta \) are approximated from the image. The values for the optimal allocation of the number of quantization levels within a region, \( n_{ij} \) is given as

\[
n_{ij} = N_i \left( \frac{r_j}{\sum_{m=1}^{N_{i-1}} r_m} \right); \quad i = 2, 3; \quad j = 1, \cdots, N_{i-1}
\]  
(2.36)
where
\[ r_j = P(B_{ij})^{\frac{1}{2}} \int p(x_i|B_{ij})^{\frac{1}{4}} dx_i \] (2.37)

and \( P(B_{ij}) \) is the probability that a data vector falls in region \( B_{ij} \). Since \( n_{ij} \) is real values and the quantization is integer valued care must be taken when mapping to integer values. The authors include a method for adjusting the integer values so that the correct number of levels is obtained and the error is minimized. Using an initial guess for \( N_1 \) and \( N_2 \) the data is processed to produce a set of quantization values.

Then an error metric is used to determine the difference
\[ d_i = \frac{1}{3} \sum_{i=1}^{N} \sum_{x_j \in S_i} [x_{ij} - y_i]^2, \quad i = 1, 2, 3 \] (2.38)

where \( x_j = [x_{1j}, x_{2j}, x_{3j}] \) is the \( j^{\text{th}} \) input color and \( S_i \) is the quantization region with quantize value \( y_i = [y_{1i}, y_{2i}, y_{3i}] \). The initial values for \( N_1 \) and \( N_2 \) are then used to determine empirical values for \( \alpha, \beta \) and \( \eta \)
\[ \alpha_e = N_1^2 d_1, \quad \beta_e = \frac{N_2^2}{N_1^2} d_2, \quad \eta_e = \frac{N_2^2}{N_2^3} d_3 \] (2.39)

These values are placed back into (2.35) to obtain values for \( N_1 \) and \( N_2 \) for use in the final quantization. The authors make an argument for quantizing the chrominance bands first and then the luminance last based on the "intuitively appealing idea" of assigning hue first and then providing shading. From experiment, the authors determined \( N_1 = 18 \) and \( N_2 = 30 \). They also note that the luminance is predicted closely enough from the asymptotic theory to be used and thus the actual preliminary quantization is not performed. The value for \( d_3 \) is obtained from
\[ d_3 = \frac{1}{3} \sum_{j=1}^{N_2} \sum_{12n_3j} \left[ p(x_3|B_{3j})^{\frac{1}{4}} \right]^3 P(B_{3j}) \] (2.40)

They also account for the HVS sensitivity to luminance errors by weighting the luminance component in the computations with a value \( K \)
\[ N_1 = N_1^{\frac{1}{4}} \left( \frac{\alpha^2}{\beta \eta} \right)^{\frac{1}{4}}, \quad N_2 = \frac{N_2^{\frac{3}{4}}}{K^{\frac{1}{4}}} \left( \frac{\alpha \beta}{\eta^2} \right)^{\frac{1}{4}} \] (2.41)
They report a value of $K = 4$ to be able to reduce objectionable contouring. They go further by adding a spatial activity weight to luminance values to account for smooth versus noisy regions of color. The image is processed in 8 X 8 blocks and an average gradient $\alpha_l$ is computed based on a gradient measure in luminance

$$|\nabla_{mn}| = |Y_{m,n} - Y_{m,n-1}| + |Y_{m,n} - Y_{m-1,n}| \tag{2.42}$$

The inverse is used since the smooth regions need more weight, $\omega_l = \frac{1}{\alpha_l}$. The weights are used during the quantization of the luminance dimension. This amounts to modifying equation (2.36) by replacing $r_j$ with $r'_j = \omega'_j r_j$ where $\omega'_j$ is the average of the subjective weights $\omega_x$ of all colors $x$ in $B_{3j}$. The actual pixel mapping is done through 3 look up tables produced during quantization. The algorithm is $O(N)$ but involves many passes through the data and a transform to and from $YCrCb$.

Wu and Witten [70] proposed an improvement to Heckbert's method. The region cutting decision was based on color quotas and variance. In a later work, Wu [69] states that a region $\Omega$ is cut into $\Omega_1$ and $\Omega_2$ orthogonal to the axes and attempts to minimize an error measure

$$E(\xi) = \sum_{c \in \Omega_1} P(c) [c_j - \mu_{1j}(\xi)]^2 + \sum_{c \in \Omega_2} P(c) [c_j - \mu_{2j}(\xi)]^2 \tag{2.43}$$

where the $\mu_i$ are the marginal means of the $\Omega_i$ on axis $j$ where $\Omega$ has the widest extent. Marginal variances are used to speed computation and an estimator is used for $\xi \cong \xi_{opt} \approx \mu_j + 0.2(\mu_j - \tau_j)$ where $\mu_j$ and $\tau_j$ are the $j$ coordinates of the mean and geometric center of the region $\Omega$ respectively. The region chosen for cutting is based on color quota where $K$ color quotas are originally assigned to the entire space and when split the new regions are assigned quotas $K_1$ and $K_2$ according to

$$K_1 = \left[ \frac{K \sum_{c \in \Omega_1} P(c) [c - \mu(\xi_{opt})]^2}{\sum_{c \in \Omega_1} P(c) [c - \mu_1(\xi_{opt})]^2 + \sum_{c \in \Omega_2} P(c) [c - \mu_2(\xi_{opt})]^2} \right], \quad K_2 = K - K_1 \tag{2.44}$$
which is again approximated to avoid variance computations by

$$K_1 = \left[ \frac{K |S_1| L(\Omega_1)}{|S_1| L(\Omega_1) + |S_2| L(\Omega_2)} \right], \quad K_2 = K - K_1$$

(2.45)

with $|S|$ as the size of the set and $L(\Omega)$ the average of the extents of the region. The authors called this an adaptive bi-split algorithm. It was interpreted and reported by Wan et al., [66] and called a mean-split algorithm. The interpretation states that the color quota assignment during the split follows the formula

$$K_i = K - \left( \frac{n_i}{n_1 + n_2} + (1 - q) \frac{V_i}{V_1 + V_2} \right), \quad (i = 1, 2)$$

(2.46)

where $n_i$ is the number of elements and $V_i$ is the volume of the region $i$. The parameter $q$ is reported as heuristic and restricted to a range $0.5 \leq q \leq 0.7$. They also note that a minimum region size is specified as a cutting limit. The original report was unavailable for the current author.

Wan et al., propose a clustering algorithm [66] based on mean and variance of the region chosen for cutting. They define a weight, mean and variance for region $l$ as

$$W_l = \sum_{x \in \Omega_l} p(x)$$

(2.47)

$$\mu_l = \sum_{x \in \Omega_l} \frac{p(x)}{W_l} x$$

(2.48)

$$\sigma_l^2 = \sum_{x \in \Omega} ||x - \mu_l||^2 \frac{p(x)}{W_l}$$

(2.49)

where $x$ and $\mu$ are vectors. They also define a weighted variance

$$\tilde{\sigma}_l^2 = W_l \sigma_l^2 = \sum_{x \in \Omega} ||x - \mu_l||^2 p(x)$$

(2.50)

They propose that each division occurs on the region with the highest weighted variance. This division is orthogonal to the axes and is placed perpendicular to the axis determined to cause the maximum reduction in variance. The algorithm works on one dimensional distributions obtained by projecting the data unto the axes in the region.
Each dimension is checked and the optimal position from all possible is chosen. The optimum placement of the cut is given as

$$t_{opt} = \arg \max_i \left[ \sigma^2 - (w_1 \sigma_1^2(t)) + (w_2 \sigma_2^2(t)) \right]$$

(2.51)

where \( w_i \) and \( \sigma_i^2 \) are the weight and variance of the \( i \)th interval. They provide a simplifying equivalence

$$\sigma^2 - (w_1 \sigma_1^2(t)) + (w_2 \sigma_2^2(t)) = \frac{w_1}{w_2} [\mu - \mu_1(t)]^2$$

(2.52)

where \( \mu_1(t) \) is the mean of the first interval and ultimately use

$$t_{opt} = \arg \max_{\mu_{lower}} \max_{\frac{\mu}{2} \leq t \leq \frac{\mu}{2} + \text{upper}} \left[ \frac{w_1}{w_2} [\mu - \mu_1(t)]^2 \right]$$

(2.53)

to determine the optimal cutting position through exhaustive search. Note the lower and upper are the boundaries of the distribution. The centroids of the determined regions are used as the quantization value for the region. The algorithm may be implemented using a spatial-storage scheme requiring \( O(N) \) space or a frequency-storage scheme where the constant of \( O(N) \) would be typically reduced. They allow for memory efficiency by bit shifting the data by 3 bits. The algorithm is \( O(KN) \).

Wu adds another modification to the basic divisive methodology [69] and breaks with the traditional orthogonal cutting plane. A cutting plane perpendicular to the principal axis is used in this algorithm. It may be implemented in a blind recursive manner or based on the minimization of variance. Wu reports that the minimization method performs better. At a given point of execution of the algorithm when there are \( k \) regions, the next region \( \Omega_t, 1 \leq t \leq k \leq \Omega, \) to divide is

$$t = \arg \max_{1 \leq i \leq k} \left\{ E(\Omega_i) - E(\Omega_{j,1}) - E(\Omega_{j,2}) \right\}$$

(2.54)

where \( E(\Omega) \) is the error for the region \( \Omega \). To implement division along the principal axis, the covariance matrix of a region must be computed

$$C = E\{cc^T\} - E\{c\} [E\{c\}]^T$$

(2.55)
For a three dimensional space like RGB for example, there are nine covariances in the 3 X 3 matrix

\[ c_{ij} = \frac{\sum_{c \in \Omega_i} c_i c_j}{|\Omega_i|} - \mu_i \mu_j, \; 1 \leq i, j \leq 3 \]  

(2.56)

where RGB is indexed as 123 for simplicity. From these the largest eigenvalue and then the largest eigenvector, which determines the principal axis, are computed for each region. Once the principal axis of the region is known, the data is projected onto the axis and sorted according to

\[ c_1 \leq c_2 \text{ if } c_1^T v \leq c_2^T v \]  

(2.57)

The author notes that this is performed with a linear bucket sort into 512 buckets. This is based on the observation that the domain of the input data is \([0, 255]\). This sorted data is used to determine the optimum cutting position by sweeping through the sorted data

\[ t_{opt} = \arg \max_t \left\{ \sum_{k=1}^{3} \left[ \frac{\left( \sum_{c \in \Omega_1(t)} c_k \right)^2}{|\Omega_1(t)|} + \frac{\left( \sum_{c \in \Omega_2(t)} c_k - \sum_{c \in \Omega_1(t)} c_k \right)^2}{|\Omega_2(t)|} \right] \right\} \]  

(2.58)

Updates to the sums may be performed incrementally. The use of a two-means clustering algorithm is added to improve the quantization value. The principal axis partition is used as a starting point. The data division provides two tentative centroids \(c_1\) and \(c_2\). With a midpoint of a line connecting the centroids, \(c_m\), we have

\[ (c - c_m)^T (c_2 - c_1) = 0 \]  

(2.59)

By placing a color point \(c \in \Omega\) into the equation and checking the sign, positive is closer to \(c_2\) and negative is closer to \(c_1\), we determine cluster membership. The computation iterates until convergence is determined and the author reports an average of 4 iterations to convergence. The algorithm complexity is \(O(|\Omega| \log K)\) expected time. In a later work, Wu \([68]\) augments this method, suggesting that a luminance, chromaticity color space should be used because it is more uniform than RGB space.
The algorithm design is aimed at providing a more global optimization. The luminance component tends to dominate and the principal axis of data sets in these spaces tends to be very close to the principal axes in the partitioned space. Thus the algorithm makes the first $t$ partitions along the main principal axis of the data set. The principal axes of the partitions are checked and when they vary enough from the main principal the algorithm shifts into a local partitioning scheme as outlined above. Through pre-computation and recording of results the author claims $O(|\Omega|)$ time.

The principal axis splitting method has been incorporated into other work. Bouman and Orchard [9] use the principal axis method and add weightings to help address the perceived image quality. They outline a gradient based weight but due to time complexity of the weight calculations, do not fully develop it. A weighting method based on erosion methods is further developed. The algorithm performs principal axis splitting based on $TSE$ error metric for a predetermined number of partitions. It then calculates a weight for all partitions based on the erosion like method where a fixed lattice of points is used to count the number of times that lattice will fit entirely inside a group of equal valued data. The principal eigenvectors for each partition are then weighted by this measure and splitting resumes based on the weighted eigenvectors. In this manner, smooth regions are allocated more quantization values and thus contouring may be reduced. In a very similar work [49] the authors acknowledge the LBG algorithm but discount it due to time complexity and usefulness. They note that for sub-optimal quantization it will improve the results but on optimal or near-optimal quantization it does little to improve the result. They also note that the LBG algorithm complicates the data mapping step because it breaks the decision tree like structure of the data. Akarun, Ozdemir and Yalcin [1] offer a modification to Orchard and Bouman's methods that offsets the quantization value for a region from the regular mean of the region. They claim that it improves results when used with dithering methods. They offset the values in opposite directions along the principal
axis of the region being split. The offset is determined by an empirically determined value $0 \leq \mu \leq 2$. The reference is very terse and the rationale behind the modification is not discussed. Work combining two major methods outlined above is presented by Balasubramanian et al., [5] where the splitting is performed along the principal axis and the weighting is based on a gradient activity measure.

Yang and Lin [74] provide another partition scheme based on a Radius Weighted Mean (RWM). The mean of a data set is computed

$$
\overline{r} = \frac{1}{|\Omega|} \sum_{i=1}^{|\Omega|} r_i, \quad \overline{g} = \frac{1}{|\Omega|} \sum_{i=1}^{|\Omega|} g_i, \quad \overline{b} = \frac{1}{|\Omega|} \sum_{i=1}^{|\Omega|} b_i
$$

(2.60)

and a weighted mean is calculated

$$
r' = \frac{1}{W} \sum_{i=1}^{|\Omega|} r_i w_i, \quad g' = \frac{1}{W} \sum_{i=1}^{|\Omega|} g_i w_i, \quad b' = \frac{1}{W} \sum_{i=1}^{|\Omega|} b_i w_i
$$

(2.61)

where

$$
W = \sum_{i=1}^{|\Omega|} w_i = \sum_{i=1}^{|\Omega|} \sqrt{(r_i - \overline{r})^2 + (g_i - \overline{g})^2 + (b_i - \overline{b})^2}
$$

(2.62)

The division plane is placed through the RWM and normal to a line through the RWM and the data centroid. If the RWM and the centroid are the same point, (a rare occurrence), the plane is placed through the point and perpendicular to the color space axis with largest variance. After the initial split the variance of each region is checked and the region with the largest variance is the next to be split. This repeats until the desired number of regions is obtained. The centroid of each region is then the quantization value. The algorithm performance is significantly faster than the principal axis methods since the covariance and eigenvector computations are avoided and the method has a simpler implementation for real time applications. To gain speed the partitions are perpendicular to the color space axis. The largest variance axis is identified and the partition plane placed through it. The computations are only performed on the component along the highest variance axis.

$$
\overline{c} = \frac{1}{|\Omega|} \sum_{i=1}^{|\Omega|} c_i, \quad c' = \frac{\sum_{i=1}^{|\Omega|} |c_i - \overline{c}|}{\sum_{i=1}^{|\Omega|} |c_i - \overline{c}|}
$$

(2.63)
The authors also show research results from using RWM, Median-cut, uniform and mean-split results as input to the LBG algorithm. The time factor of their algorithm is shown as significantly faster that the other methods when joined with LBG thus implying that the result of RWM is closer to the optimal result of LBG.

A family of space division algorithms exist that are closely related in philosophy. The algorithm was originally developed by Braudaway [10] and later improved by Gentile et al., [27]. The color space is uniformly quantized and the number of values counted in each region. Thus a coarse histogram value $h_c(i, j, k)$ is obtained. Next the region with the highest value is selected and the centroid used as a quantization value for the region. Then all coarse histogram values are reduced to prevent a next choice too close to the first choice. The reduction is based on distance

$$h_c'(i, j, k) = \left(1 - e^{-\alpha r^2}\right) h_c(i, j, k)$$  \hspace{1cm} (2.64)

$$r^2 = (i - i')^2 + (j - j')^2 + (k - k')^2$$  \hspace{1cm} (2.65)

and the value of $\alpha$ was chosen so that the histogram was reduced by $\frac{1}{4}$ at a distance of $r = \frac{M}{4}$ by Braudaway and $\frac{1}{4}$ at a distance of $r = \frac{M}{8}$ by Gentile et al. due to the larger output space. $M$ is the dimension of the color space. The algorithm then repeats until the number of quantization values is determined. A final step is the implementation of the LBG algorithm to help improve results.

A close resemblance to the Braudaway algorithm is found in an algorithm presented by Chang and Liu [14] where the original coarse quantization is not included but a transform to YCrCb space is performed and quantization done in that color space. The algorithm initially works with the luminance component and is driven by the color histogram. The $M$ most frequent colors are computed and if $M \leq K$ these are used as the quantization values with decision boundaries set half way between any two consecutive values. If $M > K$ then the histogram is processed in decreasing frequency. The maximum frequency value in the histogram is chosen. If the distance
from this value to those already chosen as quantization values is less than a prede-
termined threshold it is discarded and a new maximum value chosen. If the value
passes the threshold test it is included as a quantization value and the algorithm
loops back to choose again. The algorithm repeats until $M = K$. Once luminance is
quantized the two chrominance signals are quantized. This is a uniform quantization
along each of these axes. A conditional probability histogram is used for each axis
with the uniform regions and the centroid of each range chosen for the quantization
value for that component. Thus there is a square grid in two axes of varying thickness
along the third axis with membership of the region determining the centroid along
each axis direction. Chang and Liu compare results with the LBG algorithm and
their advantage is speed. This algorithm should perform in $O(N)$. They quantize
the luminance into 16 levels and each chrominance into 4 yielding 256 levels.

A somewhat similar method to Braudaway is presented by Sakauchi et al. [58]. A
histogram and a data structure called a BD-tree of their design is employed. The most
frequent color is chosen as a quantization value and removed from competition. All
colors with a spherical radius of predetermined size are eliminated from competition
and the algorithm repeats. If candidates are exhausted before enough quantization
values are determined, the algorithm replaces removed candidates that fell into two
or more spheres and then continues. If the number of quantization values is reached
before the candidates are exhausted, the remaining candidates are included through
nearest neighbor calculations. They implement a user feedback (interactive) algo-
rithm that allows the user to correct area colors. They do not elaborate the method
but indicate that the feedback is an effect to the histogram.

A final algorithm developed very early in the history of color quantization is
that of Houle and Debois [35] now commonly referred to as the MinMax algorithm.
The design is very simple. A histogram is created of the data. An initial color is
chosen as a quantization value. Either the most common color or the most extreme
color in dimension terms i.e., black. A next value chosen as a quantization value is searched for from among a group of colors which pass an unspecified frequency threshold test. The color whose minimum distance to all the colors selected so far as quantization values is maximum is chosen and added to the quantization values set. This methodology insures that a wide spread of quantization values is used to span the gamut of image colors. However, because of the threshold test, this method can suffer from the loss of rare but important colors as mentioned previously.

Merging

The algorithms of this section differ from the last in that they tend to group or merge data into clusters or regions rather than split data into regions. Note that these are dual to the above methods. The intuitive objective is clear. Take a set of data, find two datum that are closest together in some metric of the space and group them by replacing them with their centroid. That first step is easy. Once the data set contains grouped data represented by the group’s mean, things get more difficult. The distance metric now acts on a mean value and closest may not be optimal. The other difficulty is that merging a large group implies a heavy cost because the error is computed for each element of the group. An early work by Equitz [20] introduced a fast Pairwise Nearest Neighbor (PNN) algorithm. The algorithm uses a k-d tree to organize the data. Since the image was 256 gray scale, 4 X 4 blocks of data were used as vectors. The decision metric in the internal nodes was based on the coordinate with greatest variance. Once the tree is built, each leaf is considered and the two closest data become candidates for merging. A threshold is used to limit the amount of merges on each iteration based on the observation that a leaf may not contain data that is relatively close. After the predetermined percentage of merges is performed the tree is re-balanced. Leaves with many data are split and leaves with few data are joined with neighbor leaves. The process repeats until a predetermined number of clusters
is formed or until an error threshold is violated. Equitz developed this algorithm to
deal with the difficulties of the LBG algorithm. The LBG algorithm iterates until
convergence with $O(\Omega^2)$ time complexity. The PNN of Equitz is dominated by the
tree construction and results in $O(\Omega \log \Omega)$ time complexity.

Balasubramanian and Allebach [4] took up where Equitz left off. They worked
with color images and thus each element was a vector (RGB). They also worked from
histogram data rather than raw data. They noted that the algorithm suffered from
contouring and proposed two modifications to address this issue. The first is a pre­
quantization step. They divide the image into 8 X 8 blocks and compute an activity
measure based on the block mean

$$a_k = \frac{1}{64} \sum_{x \in \Omega_k} ||x - \bar{x}_k||_1$$

The activity measure is assigned to every image vector of the block. Since the same
vector may occur in different blocks with different activity, they assign the global min­
imum activity measure to all vectors of equal value. Based on these activity measures
they create three categories for the vectors; low, medium and high activity. Based
on empirically established thresholds the vectors are quantized into small, medium
and large size regions. The thresholds for the activity measure were established at
12 and 20. Activity below 12 places data into a small cell, between 12 and 20 places
it into a medium sized cell and above 20 places it into a large cell. The data was
placed in a Binary Search Tree (BST) to be used to create the k-d tree. The second
modification was the incorporation of a gradient weighting as discussed previously in
Balasubramanian's work above (this work pre-dates that above). Gradient weighting
is included in the distance function and causes high activity regions to be merged
ahead of low activity regions thus avoiding contouring. The pre-quantization step
reduces the overall time complexity from Equitz' algorithm. The data set is reduced
from $O(\Omega)$ to $O(\Omega')$ by the pre-quantization and this value is image dependent. The
pre-quantization simply reduces the total number of values from $\Omega$ to $\Omega'$ by grouping
values together. The improvement is mainly in the contouring solution.

An algorithm by Fletcher [22][23] uses boxes in the color space to bound the data. Initially $K$ boxes are formed for the first $K$ distinct colors of length 1 on each side. As the data is processed, a check is made to determine if the datum falls into one of the existing boxes. If the datum does not fall into an existing box, a new box is formed and the set of boxes searched for a closest pair which are then merged into a larger box. A merged box is created by forming a box based on the two most distant corners of the merging boxes. Once all the data has been processed, the centroids of the boxes become the quantization values. The second paper’s focus is on parallel implementation of this method and parallel implementation of support methods like dithering. Thus there is no time complexity analysis for the sequential method although a mention of expensive table bookkeeping exists. The work also mentions that boxes may overlap each other and so box membership is not the best quantization to use but rather a nearest quantization value to the data should be found. Due to the extensive amount of searching inherent in the method its use in a sequential machine is questionable. Through the bookkeeping data structures presented in the first paper and a limiting search method the time bound is improved but still appears to be about $O(KN)$. The parallel version offers no help in the searching but only offers more processors to tackle the problem. Although a parallel machine may be used for off-line processing for storage or transmission the benefits of this method would need comparison with parallel versions of the other quantization methods to show its value. A similar algorithm is described by Xiang and Joy [72] where the implementation is sequential and boxes are merged with a limit on size. They account for RGB properties by limiting the size ratio to 2:1:4 and mention that $L^*u^*v^*$ implementation would have cubical boxes due to the more uniform property in that color space. The algorithm has a minimum error consideration during the choice of boxes to merge. There is an ordering placed on the data through the use of
a 2-dimensional array of pointers and the linking of all data into a chain. Candidates for merging are chosen relatively close to each other in this chain. When two nodes of the chain merge, one is removed from the chain and placed as a child of the other node and all data fields are updated. Once the chain reaches the desired size the algorithm stops. Allowances are made for more than one merge per iteration when the chain is much larger than the goal. This is the dual to their splitting algorithm.

The algorithm presented by Dixit [19] is dated by the hardware commonly available at the time. The algorithm generates a histogram from a random sampling of the data in order to reduce memory demands. The data is processed in vector fashion and the histogram is stored in an ascending order of frequency. The representative vectors in the histogram are then merged pairwise until the desired number of quantization values is obtained. The frequency of the vectors is used as a weight to skew the merge towards the higher frequency vector. Distance is also weighted in this manner

\[
d_{ij} = \left\{ \frac{(r_i - r_j)^2 + (g_i - g_j)^2 + (b_i - b_j)^2}{n_i n_j} \right\}^{1/2}
\]

(2.67)

\[
\begin{align*}
    r_{ij}' &= \frac{(r_i n_i + r_j n_j)}{n_i + n_j} \\
    g_{ij}' &= \frac{(g_i n_i + g_j n_j)}{n_i + n_j} \\
    b_{ij}' &= \frac{(b_i n_i + b_j n_j)}{n_i + n_j}
\end{align*}
\]

(2.68)

The frequency of the new vector is set to \((n_i + n_j)\). The vectors are merged pairwise such that a merged pair is not considered for merging again until all that remains are merged vectors. They indicate that the weighting biases the merging so that low frequency vectors are merged first, then low and high frequency vectors and lastly high frequency vectors are merged in that order. Since a merged pair is removed from consideration on an iteration, the sorted histogram need not be re-sorted. They do not mention if re-sorting is done between iterations although it seems prudent. A nearest-neighbor algorithm is used to map the data to the final quantization vectors formed. Since only a random sample histogram is used and current hardware and data structures allow complete histogram use, there is the question of how well this works
with a complete histogram of an image. However, since there is no error minimization explicit in the algorithm and no consideration of the HVS for items such as contouring, this algorithm falls into the quick and dirty implementation category.

One of the most successful developments along the merging philosophy is that of Gervautz and Purgathofer [29] called the Octree Quantization. This is both a methodology and a data structure. The octree itself was developed elsewhere but is perfectly suited for quantization. The octree is a tree structure where each internal node has 8 children. For RGB data the individual components are used to build and navigate the tree. At the root the most significant bit of each component is used to determine a value. Since \(2^3 = 8\) the node has 8 children. At the next level down, the next most significant bit set is used. In this manner a tree of height 8 can completely span a 24 bit color space. Typically the tree is not created in its entirety but is grown as data is processed and pruned when the number of leaves exceeds the quantization amount desired. The tree also acts as a dynamic histogram. A count of the number of vectors that map to a leaf is kept in the node. The pruning occurs as follows. When the number of leaves exceed the threshold the lowest level of non-leaves is searched and one node chosen for reduction. The reduction is to remove all the leaves and sum the leaf counts. This sum is stored in the parent node which now becomes the quantization value as a centroid of the leave values. The decision of which node to reduce may be either the one with the lowest children count, thus minimizing error, or the one with the largest count, thus allowing contouring but preserving anti-aliasing and shading. To quantize the image, a second pass through the data is made and the tree traversed until a leaf node is reached. The leaf node contains the centroid value or the actual data value if the reduction never occurred. The tree traversal is \(O(8) = O(1)\). A very similar algorithm is described by Chang and Chang [13]. Implementations may be found in [2] and [54].

A hybrid algorithm combining the pre-quantization of Balasubramanian and
Allebach [4] with an octree quantization and a final LBG algorithm refinement is presented by Chaddha, Tan and Meng [12]. They use equation (2.66) in a modified way to obtain a weighted mean square error

\[ D_{W\text{MSE}} = \frac{1}{64} \sum_{i=1}^{64} \frac{e_i^2}{\alpha^2} \]  

(2.69)

where \( e_i \) is the norm of the error vector. Instead of thresholds of 12 and 20 they determined thresholds of 9 and 11. These thresholds are used in an octree to determine a split of a node. Initially the color space is divided into 8 equal size regions. Then the members of each region are checked against the thresholds. If members carry a weight indicating lower activity than the current region is allowed, the region is split into 8 equal sized regions. Activity measures above 11 cause no further division. Values from 9 to 11 cause the second division into 8 regions. Any secondary region with activity weighted values below 9 are divided again into 8 regions. This is the coarse quantization phase. The regions are then mapped into voronoi regions of the color space by splitting with a tree structured vector quantizer (TSVQ). The centroids of the course quantization are used as training vectors to the TSVQ algorithm. Initially all vectors are placed together and a generalized Lloyd algorithm (GLA) is used to determine the weighted centroid. The node is then split with the centroid going to one child and a random perturbation of the centroid going to the other child. The GLA is then run again on the child and future children with maximum weighted distortion. Due to the iterative GLA and the greedy TSVQ a rough complexity measure is \( O(N \log N) \).

Iterative Improvement

The algorithms discussed so far have attempted to provide relatively quick solutions. It has already been mentioned that there are algorithms that iterate to converge
to a locally optimal solution. These types of algorithms are computationally expensive due to their time factor. Usually an initial guess is made for a solution and this is improved upon through repeated re-computation. Depending on how accurate the guess was, the algorithms may need to iterate many times before convergence is established. The attraction of these algorithms may be the intuitive improvement they offer along with the simplicity of implementation. If ultimate quality is the goal, these algorithms have merit. However, due to the extremely long running times, they are not viable in most applications. There is some research based on retaining the philosophy of the methods while providing time improvement through heuristics, shortcuts and optimizations that may not preserve the accuracy of the original algorithm but do provide relatively quick quality results. There is also the issue that these methods may be implemented as a post-processing improvement to one of the quantization algorithms discussed above. The LBG algorithm has been applied by many researchers to the results of their algorithms. There are two important reasons why this works. If the proposed algorithm provides a quality result, the iterative improvement algorithm will do little to change that result. If the proposed algorithm provides a fair guess the iterative improvement algorithm will converge very quickly. The former validates the proposed algorithm and the latter completes the proposed algorithm performance.

The algorithms discussed above are referred to as pre-clustering in nature while the following are referred to as post-clustering.

The LBG algorithm has been discussed above. This algorithm is the most popular in the color quantization literature. This algorithm comes from vector quantization theory. Another area of theory that has provided some methodology is the neural network arena. Dekker presents an algorithm based on Kohonen self-organizing neural networks or maps [18]. Kohonen's work is discussed in the paper and further details may be found there. Dekker provides the information necessary for the implementation of the algorithm. The algorithm works on training a set of weight vectors in
RGB space \{\langle R_i, G_i, B_i \rangle \}. Dekker mentions that Kohonen suggests starting with a set from the center on the space cube but his work showed that the diagonal with \( R_i = G_i = B_i = i \) worked better. It is not noted but this is a uniform set along the gray scale axis which would align with the luminance component and thus makes intuitive sense. As the data is scanned the 'best' weighted vector is determined and updated. Dekker uses the Manhattan distance metric and notes that convergence to zero in the Manhattan metric implies convergence to zero in the Euclidean metric. The updating is performed by

\[
\langle R_i, G_i, B_i \rangle = \alpha \langle R, G, B \rangle + (1 - \alpha) \langle R_i, G_i, B_i \rangle
\]

(2.70)

where \( \langle R, G, B \rangle \) is the current value scanned. The parameter \( \alpha \) is initially 1 and decreases with time. Kohonen suggests a linear decrease but Dekker reports an exponential decrease performs better. The algorithm cycles 100 times and \( \alpha \) decreases to 0.05 by the last cycle based on time \( t \) as

\[
\alpha = e^{-0.03t}
\]

(2.71)

This breaks from the theory presented by Kohonen but Dekker reports that it performs “extremely well”, requiring less training than usual. The network is considered “elastic”; when one vector moves its neighbors also move. This movement is limited to a neighborhood \( j \) based on a radius \( r \) with \( i - r \leq j \leq i + r \). The parameter \( r \) is also decreased with time. The updating function now contains a parameter function.

\[
\langle R_j, G_j, B_j \rangle = \alpha \rho_{(i,j,r)} \langle R, G, B \rangle + (1 - \alpha \rho_{(i,j,r)}) \langle R_j, G_j, B_j \rangle
\]

(2.72)

The decrease in \( r \) is also exponential from the initial value of 32 to less than 2 at cycle 86 and is computed at time \( t \) as

\[
r = 32e^{-0.0325t}
\]

(2.73)

With the definition of \( \rho_{(i,j,r)} \) given as

\[
\rho_{(i,j,r)} = \begin{cases} 
1 & \text{if } j = i \\
1 - \left( \frac{|j - i|}{r} \right)^2 & \text{if } j \neq i
\end{cases}
\]

(2.74)
the update to \((R_i, G_i, B_i)\) is the same as in equation (2.70) and for the last 14 iterations the update only occurs to \((R_i, G_i, B_i)\). To distribute the training, \(\frac{N}{100}\) data values are used in each cycle and chosen from the image set at \(P\) intervals where \(P\) is a prime near 500 and not a factor of \(N\). Dekker used \(P \in \{487, 491, 499, 501\}\). This causes 5 scans across the data set on each cycle and after 100 cycles each datum is used once. Dekker reports that over-sampling does not improve performance and sub-sampling gives a poorer result. One final piece of the algorithm remains, that of ‘best’ weighted vector determination. In order for less frequently occurring data to get better treatment, the distance metric is biased by \(b_i\) in

\[
D = |R - R_i| + |G - G_i| + |B - B_i| - b_i \tag{2.75}
\]

where

\[
b_i = \gamma \left( \frac{1}{256} - f_i \right) \tag{2.76}
\]

with \(\gamma\) a constant and \(f_i\), the frequency estimate, initially at \(\frac{1}{256}\). The parameters are updated with

\[
f_i = \begin{cases} f_i - \beta f_i + \beta, & \text{if vector } i \text{ is closest} \\ f_i - \beta f_i, & \text{for all other vectors} \end{cases} \tag{2.77}
\]

and

\[
b_i = \begin{cases} b_i + \gamma \beta f_i - \gamma \beta, & \text{for the closest vector} \\ b_i + \gamma \beta f_i, & \text{for all other vectors} \end{cases} \tag{2.78}
\]

The constants reported by Dekker are: \(\gamma = 1024, \beta = \frac{1}{1024}\), thus \(\gamma \beta = 1\). The result of this is that a relatively small cluster will be allocated a vector rather than lumped into a larger cluster. Dekker observes that the bias changes slowly and thus the updates are implemented to occur during the linear search for the closest vector.

Another algorithm based on the Kohonen neural network method is presented by Jun, Kim and Cha [39]. They modify the updating computation by considering the last correction during the current correction. They refer to this as momentum. They also use a dynamic weighting term based on time and employ Euclidean distance.
Adopting from the equations above their main computations are

\[
\langle R_i, G_i, B_i \rangle (t + 1) = \alpha (t + 1) \langle R, G, B \rangle (t) + (1 - \alpha (t + 1)) \langle R_i, G_i, B_i \rangle (t) \quad (2.79)
\]

\[
\alpha (t + 1) = F (\mu_i) + \frac{1}{f_i} \quad (2.80)
\]

where \( F \) is a normalization function and \( f_i \) is the frequency. They make allowances for neighborhood values from a 4 X 4 block to avoid blocking effects. If the neighbors of a value meet a threshold criteria they are weighted with the current value. A short paper by Galli, Mecocci and Cappellini [26] reports the successful merging of Kohonen neural network methods and the LBG algorithm. The details are not presented and the performance is given in signal to noise ratio terms.

Verevka and Buchanan proposed a local k-means (LKM) algorithm based on a set of observations and modifications of the Kohonen implementation above [64]. Their first modification is to remove the 'elastic' aspect of the network, only the closest vector is modified. A second speed modification is to use a different distance metric based on a \( L_\alpha \) norm with \( \alpha = \frac{1}{2} \) for a vector \( x \in \mathbb{R}^n \)

\[
||x||_\alpha = (1 - \alpha) \sum_{i=1}^{n} |x_i| + \alpha \max_i |x_i| \quad (2.81)
\]

During nearest neighbor searching, they report that the \( L_1 \) norm chooses the wrong neighbor 12% of the time and that the \( L_\alpha \) they use chooses the wrong neighbor 4% of the time based on \( L_2 \) norm choosing correctly at all times. Further search time improvements are: (i) short circuit of equation (2.81), if the calculation exceeds the current minimum the computation is terminated, (ii) sorted palette entries based on one coordinate, when the chosen coordinates between the next palette entry and the current datum exceed the current minimum the search is terminated, and (iii) next neighbor distance, when the current minimum is less than half the distance between the current palette entry and the next palette entry the search is terminated.
Another type of iterative algorithm comes from a feedback system. The color quantization is performed and the resulting image compared with the original. Some form of error metric is used to determine the quality and if insufficient results were produced a correction system is used and the color quantization performed again. Joy and Xiang propose a system such as this [73][38] where the embedded quantization algorithm is a modified version of the algorithm proposed in an earlier work [72]. The modification is to add importance factors to the data values. These factors are based on false contours produced in the resulting quantized image. An edge detection scheme of their design is used and importance factors are based on three criteria: (i) the highest edge element in a false contour, (ii) total color difference between the original image and the quantized image in the false contour area and (iii) the population of the false contour area. The importance assigned to the original values for the re-quantization phase is proportional to

\[(me)^2 \left(1 + \frac{tcd}{\max (tcd)}\right) \left(1 + \frac{pp}{\max (pp)}\right)\]  \hspace{1cm} (2.82)

where \(me\) is the maximum edge, \(tcd\) is the total difference, \(pp\) is the pixel population and the max functions operate on all values for \(tcd\) and \(pp\) in the current iteration. A color may occur in more than one area that receives importance weighting and is set to the highest importance obtained. They discuss problems and solutions with edge matching. Those details may be found in the reference.

An extremely exhaustive method is presented by Spaulding, Lawrence and Sullivan [60]. They designed an algorithm that takes an initial guess at the quantization and calculates a cost function

\[
Cost = \sum_{i=1}^{N_A-1} \sum_{i=1}^{N_B-1} \sum_{i=1}^{N_C-1} P A_{0,i} B_{0,j} C_{0,k} \Delta E (A_0, B_0, C_0, A_q, B_q, C_q)^2
\]

\hspace{1cm} (2.83)

where \(P\) is the probability for each input color and \(\Delta E\) is the error function. They use equal probability and Euclidean distance as an error. They then calculate the
change in the cost function of all cases where a pair of bins is reduced / enlarged by one unit. They process each component channel for a specified number of iterations or until the reduction / enlargement search fails to produce an improvement. They do no time analysis but report run times in the 10 hour range on a MIMD parallel processor with 64 nodes rated at 640 Mflops peak performance. They do not give the dimensions of the contrived test image.

A method of contouring detection and adjustment is presented by Shufelt [59]. Although this algorithm may be used as a single pass method, it may be employed iteratively for improved results and thus is included here. The first phase of the algorithm is a color map generation using the median-cut algorithm. The second phase is a contouring colors detection phase. A k-d tree is created during the first phase and modifications to the tree occur based on the suspected contouring colors. A co-occurrence matrix is created based on the color map. Given a color map of size \( k \), the matrix is \( k \times k \) in size. This method is common in textural analysis. The elements of the co-occurrence matrix \( A[i,j] \) record the number of times color \( i \) and color \( j \) occur in a specified spatial relationship. For this work the spatial relationship is adjacency. Statistical measures are generated based on the co-occurrence matrix that are used to determine contouring and tree restructuring. A co-occurrence frequency is computed

\[
C_j(i) = \frac{A[i,j]}{\sum_{k=0}^{n-1} A[i,k]}, \text{ if } i \text{ occurs, otherwise } 0
\]  

(2.84)

The concept of self co-occurrence is when a color appears next to itself and is given as

\[
S(i) = C_i(i)
\]  

(2.85)

Then the mean and standard deviations are computed with \( \mu_s \) and \( \sigma_s \) for \( S(i) \), \( i = 0 \cdots n - 1 \), and \( \mu_{C(j)} \) and \( \sigma_{C(j)} \) for \( C_j(i) \), \( i = 0 \cdots n - 1 \). Shufelt employs the Euclidean metric \( D(i,j) \) and contends that two colors are contouring iff

\[
S(i) > \mu_s + \sigma_s
\]
\[
S(j) > \mu_s + \sigma_s \\
C_i(j) > \mu_{C(i)} + \sigma_{C(i)} \\
C_j(i) > \mu_{C(j)} + \sigma_{C(j)} \\
D(i, j) \leq \tau
\] 

(2.86)

and determined that the threshold value \( \tau = 12 \) works best for the test images used.

In order to modify the k-d tree to improve on contouring and yet keep the tree size constant, a modification that splits a node must match a modification that merges two leaves. Thus the concept of compressible colors is used. They contend that two colors are compressible iff \( i \) and \( j \) are leaves of the same parent and

\[
A[i, j] = 0 \\
S(i) < \mu_s \\
S(j) < \mu_s
\]

(2.87)

Based on these measures, a list of contouring colors is created and sorted into a partial order based on the sum of the self occurrence frequencies and self co-occurrence frequency. Multiple occurrence of colors is possible and only the highest valued occurrence is retained. This list is then used with the compressible pairs list to modify the k-d tree. As long as a pair can be obtained from the lists, the tree is modified. When one of the lists is exhausted the algorithm stops. At this point the algorithm may iterate and recompute the statistics and lists and modify the tree again until one of the lists cannot be generated. Example results are presented for no adjustment, single pass and iterative adjustment but no mention of number of iterations is made.

Image Sequences (Video)

The methods discussed above pertain to still images and thus the time complexity may be treated lightly if quality is the determining priority. This does not hold
true when color quantization is applied to sequences of images such as those in video or animation. Although an algorithm may be applied to a sequence off-line and the result stored for later viewing, live feeds or streams of images need to be processed in real time. Even with a relaxation from real time, the algorithm should be as quick as possible. The algorithm above which consumes up to 10 hours on an advanced machine for a single image is completely out of the question. The high speed demand must implicitly grant a quality variance with today’s machine speeds. Thus algorithm design for image sequences will tend toward simple methods. Image sequences also carry extra burden that must be addressed with algorithm design. The third dimension is added to the existing spatial dimensions of the image. This is not simply a move to another spatial dimension but rather a change in spatial dimension over time. Thus individual images should not be processed in isolation. Consideration of the similarity between images should be incorporated.

The proposed algorithm of Gong et al., [30] acknowledges the inter-image dependencies and offers some solution. They present an algorithm that works in the $L^*u^*v^*$ space. The algorithm starts, or restarts for a scene change, by clearing a color lookup table and examining the first two images of the sequence. The initial image is color quantized in a still image manner with high frequency colors receiving priority. A neighborhood of the chosen colors is determined based on a radius $D$. A management table (MT) is maintained which records various statistics for the chosen colors. This table contains such items as the neighborhood radius size, frequency, tolerance based on nearest neighbor and an array of nearest neighbors along with the $L^*u^*v^*$ values of the color. The radius is the region of the color space that is represented with a quantization value and the tolerance is the minimum distance to a common chord between neighboring quantization values. These regions are allowed to overlap. The difference between the quantized image and the next in the sequence is computed. This difference is used to classify the pixels into 3 categories: (i) difference
less than tolerance, (ii) difference between tolerance and radius and (iii) difference
greater than radius. If the difference is (i), allocate the same color as the last frame.
If the difference is (ii), choose the closest color from the last image pixel and the array
of neighbors. If the difference is (iii), find the closest match in the entire MT. During
the third category operation the statistics are checked. If the color is not represented
in the MT a secondary table, SVLUT (name convention unspecified), is checked. If
the color is located there, move it to MT and remove the least frequent color of MT
while updating any pixels assigned that color. Place the removed color into SVLUT
and if SVLUT is full, overwrite the oldest stored color in SVLUT. The definition of
oldest is based on which frame was processed when the color was moved to the table.
SVLUT is used as a cache for colors that go out of use but that may reappear later
in the sequence. Finally update the MT to reflect the color changes. If too many of
these table operations are detected it is assumed that a scene change occurred and
the whole algorithm is reset. The only analysis is images from a quantized sequence.

Another approach is presented by Roytman and Gotsman [57] where they also
address the inter-image problem but refer to it as screen flicker when color maps
change between images. The algorithm is designed for speed and mixes crude uniform
quantization with more sophisticated processing. The algorithm take an image and
its color space an recursively divides the space into smaller cubes. The cube with the
highest number of values is divided in half on all three dimensions. This proceeds until
the desired number of quantization regions is reached. The color map is computed
from the centroids of the regions. The next image is processed similarly and then the
resulting color map is checked against the last map. Regions that match are arranged
so no change occurs between images. A small amount of the map is left open so
that colors from the next image may be loaded that do not affect the last image.
Quantization values in the second map that are not in the first are then assigned to
the map. These are colors not used in the first image so no flicker occurs. The last
filling is to match close colors between the two images that do not form same size regions. They use an unspecified algorithm of $O(cK)$ time complexity to approximate nearest neighbor. They claim the flicker is all but eliminated from their test sequence and that which remains is imperceptible to the eye.

Furiani, McMillan and Westover take the LBG algorithm one step further [21]. The algorithm starts with a uniform distribution of quantization values and iteratively migrates them to match the image data. The algorithm uses a nearest neighbor method and records how many image values are represented by the quantization values. The quantization values are then classified according to use. The color space is divided into 64 regions and underutilized values are removed from competition if the region they represent contains no image data. If they are utilized very little and they are the last value left in the region they are retained. Values that represent large populations of image data are moved to the average of the values they represent. Values that are not used in the current pass are used to help overtaxed values by placing them at the six sides of the value and removing the overused value from competition. The migration is controlled by a parameter $t_{\text{max}}$ and the underutilization threshold is determined stochastically from the image data. The extension to sequence data is to use a color map from one image and process the new image data against it. The migration limiting term here serves to reduce or eliminate flicker or flash between sequence images. The migration limit is allowed to be 256 for still images since no flicker is possible and they report values between 9 and 12 give good results in sequences. They also note that small movement of values between image frames is restricted since this introduces a jittering effect in the sequence.
CHAPTER 3

IMAGE DATA CHARACTERIZATION

A major area of concern with digital video is transmission and storage due to the amount of data necessary to produce quality motion pictures. Limits in bandwidth due to spectral limits of hardware and carrier frequency have set bounds on the amount of data that can be transmitted. Thus compression is a leading area of research and development at this time. A strong second area is motion compensation. This area is of strong interest both for the motion detection itself and for employment in supporting and enhancing compression.

Theory

To date there has been a great deal of research into motion compensation. Much of the effort follows similar methods. This dissertation brings a new method using differences between consecutive frames. The distribution of the differences was the starting point for this research. Advances along these lines are possible and greater accuracy is obtained by utilizing the autocorrelation of the differences.

In order to research and interpret video frame differences it is necessary to understand the data itself and what has produced it. The data is a recording of the effect of light on objects. In the context of video data it comes as a surprise to many that a video made with a stationary camera aimed at only stationary objects is not a sequence of frames with identical data. It is in fact rare to find two consecutive pixel values that are exactly the same. This randomness shall be referred to as background
noise or just noise. There are many causes of this property of digital data. The lowest level of these causes is the hardware itself that captures or creates the images. In the analog to digital conversion process, noise is caused by quantization errors. In direct digital there are various causes. Quantization errors may be caused or aggravated by power fluctuations within the hardware itself. Cell bleeding in charged coupled devices (CCD), the "film" of a digital camera, is a known cause of errors.

At a higher level, noise is caused by natural and man made means. The alternating current (AC) in common use for interior lighting by its very nature is not consistent. The eyes just don't register the variations. Natural lighting may be viewed as a more direct source but it also has fluctuations. Solar flares are an extreme example of the fluctuations in nature. Atmospheric conditions such as humidity, temperature, density (altitude), particulate matter (water vapor, smoke, dirt), and others all effect the light reaching the capture medium and thus the image. If we allow all of these as givens and rely on the eye's adaptation to compensate for them, they still make up a portion of the noise that constitutes the digital data. One area of research that allows some insight into the effects of light on objects is energy flow analysis.

Energy flow is a rich mathematical foundation which encompasses the physics of light. Included in this foundation are the metrics of light from both radiometry and photometry. The field of physics has metrics in flux, flux density and angular flux density. Radiometry and photometry have parallel metrics with physics and add a few more. Radiometry and photometry parallel each other with different terms for the same metric. Radiometry uses radiant energy, radiant power, radiance, irradiance, radiosity and radiant intensity. The parallel metrics from photometry are luminous energy, luminous power, luminance, illuminance, luminosity and luminous intensity respectfully. The second and third of each align with flux and angular flux density of physics while the forth and fifth fall into flux density of physics. A brief look at these metrics will allow insight into the complexity of light and lead to understanding the
interactive effects of objects on lighting. The objective is to illustrate how the energy of light is modeled and from that understand the interaction of light and objects.

The basic level to start from is the amount of light. The concept of light as particles is used (photons). Particle density, \( p(x) \), is the amount of particles per unit volume at \( x \). The number of particles is \( P(x) = p(x) dV \). If the particles are all moving in one direction, this is also the amount of particles that will cross a surface in a given time. Allowing for surface angle, \( \theta \), introducing a velocity vector, \( v \), and converting volume into area moving in time a more general form is \( P(x) = p(x) (v dt \cos \theta) dA \). Dropping a restriction that all the particles are moving in the same direction at the same speed we get particle density as a function of position, \( x \), and direction, \( \omega \),

\[
P(x, \omega) = p(x, \omega) \cos \theta d\omega dA.
\]

This formula is then modified by removing the quantum nature of light through a conversion to energy. The radiant energy in a unit of volume is the product of the number of particles and their energy, i.e., volume density times photon energy. Photon energy is Planck's constant, \( h \), times the speed of light, \( c \). Thus we have radiance [48]

\[
L(x, \omega) = \int p(x, \omega, \lambda) \frac{hc}{\lambda} d\lambda.
\]

The radiance distribution is a complete characterization of the light and all the other radiometric metrics can be derived from it. An example is differential flux for a beam with cross-section \( dA \) and differential solid angle \( d\omega \)

\[
d\Phi = L(x, \omega) \cos \theta d\omega dA.
\]

This is related to total energy per unit area incident onto a surface. Using an incoming radiance \( L_i \) and integrating over a hemisphere \( \Omega \)

\[
d\Phi = \int [L_i \cos \theta d\omega]\ dA.
\]

\[
d\Phi = \int_{\Omega} [L_i \cos \theta d\omega]\ dA.
\]
Dividing by total area to get energy per unit area we have irradiance or illuminance

\[ E = \int L_i \cos \theta d\omega. \]

By reversing the direction of the radiance or the concept from incoming to outgoing, we have the radiosity or luminosity

\[ B = \int L_o \cos \theta d\omega. \]

Although radiance is useful for characterizing the light between surfaces, it has a weakness in dealing with point light sources due to the singularity. For these the quantity radiant or luminous intensity is defined

\[ d\Phi \equiv I(\omega) d\omega \]

with \( I \) defined as a value of power per unit solid angle. For isotropic light

\[ I = \frac{\Phi}{4\pi}. \]

The previous equations form the basis of the characterization of the light itself. Along with these we have added complexity due to reflection of the light by the surfaces. The surface type adds considerable complexity but we will defer to the literature [51],[8],[16],[63],[33],[31],[32] and only include basics necessary to complete the radiosity formula development. Given a light incident on a surface with a differential angle \( \omega_i \), the reflected light in direction \( \omega_r \) is proportional to the incident irradiance

\[ dL_r(\omega_r) \propto dE(\omega_i) \]

This constant of proportion is contained in the Bidirectional Reflection Distribution Function (BRDF)

\[ f_r(\omega_i \rightarrow \omega_r) \equiv \frac{L_r(\omega_r)}{L_i(\omega_i) \cos \theta_i d\omega_i}. \]

Since reflection behaves linearly, this can be rearranged into a hemispherical integration to give the reflectance equation

\[ L_r(\omega_r) = \int f_r(\omega_i \rightarrow \omega_r) L_i(\omega_i) \cos \theta_i d\omega_i. \]
There are various augmentations and allowances to account for the surface properties which shall not illustrated here. However a simplifying step is to assume the proportion is constant through Lambertian diffuse reflection which assumes light is equally likely to be scattered in any direction independent of incident direction.

The remaining part of the puzzle is the illumination portion itself, the light source. This may be a local or direct light source which drops into the equations easily or a global or indirect light source which is more complex. In a global source model, the illumination of one surface is related to the reflected light distribution of another surface. A property of radiance is that it is invariant along a ray, barring scatter or absorption. We thus get a relation

\[ L_i(x_2, \omega_i) = L_o(x_1, \omega_o) V(x_1, x_2) \]

where the directions \( \omega_i = -\omega_o \), and \( V \) is a visibility function to account for occlusion (shadow). \( V(x_j, x_k) \) is 1 if surface \( x_j \) is visible to surface \( x_k \), and 0 otherwise. The integral of the reflectance equation is modified from the hemisphere integral to an area integral over all the other surfaces of the environment. This is done through the relation of the solid angle subtended by the source to the projected surface

\[ d\omega_i = \frac{\cos \theta_o dA}{|x_1 - x_2|^2} \]

which is dotted to get a projected solid angle

\[ d\omega_i \cos \theta_o dA = G(x_1, x_2) \ dA \]

where

\[ G(x_1, x_2) = G(x_2, x_1) = \frac{\cos \theta_i \cos \theta_o}{|x_1 - x_2|^2}. \]

This is placed into the reflectance equation

\[ L(x_2, \omega_i) = \int_S f_r(\omega_i \to \omega_r) L(x_1, \omega_o) G(x_1, x_2) V(x_1, x_2) \ dA. \]
This is named the rendering equation and has one final term. If all surfaces in an environment are opaque then the only other light source is the emission from the surface which is simply added in

\[ L(x_2, \omega_i) = L_e(x_2, \omega_i) + \int_S f_r(\omega_i \rightarrow \omega_r) L(x_1 \omega_o) G(x_1, x_2) V(x_1, x_2) dA. \]

The radiosity equation follows through simplification. The Lambertian diffuse reflectance assumption is made and thus the BRDF is independent of the directions and can be removed from under the integral

\[ L(x_2, \omega_i) = L_e(x_2, \omega_i) + f_r(\omega_i \rightarrow \omega_r) \int_S L(x_1 \omega_o) G(x_1, x_2) V(x_1, x_2) dA \]

\[ = L_e(x_2) + \frac{\rho(x_2)}{\pi} \int_S L(x_1 \omega_o) G(x_1, x_2) V(x_1, x_2) dA. \]

Since a Lambertian surface has outgoing radiance the same in all directions and is equal to the radiosity \( B \) divided by \( \pi \), this reduces to

\[ B(x_2) = E(x_2) + \rho(x_2) \int_S B(x_1) \frac{G(x_1, x_2) V(x_1, x_2)}{\pi} dA \]

where \( E = \frac{L_e}{\pi} \) is the energy per unit area emitted. This equation defines in part the amount of light at the surface of an object. The scattering, absorption and reflectiveness of the surface were all ignored or simplified to reach this stage. Not included are allowances for distribution of energy across the spectrum (color), interaction with the surface (fluorescence), energy loss through heat transfer and others. Note that the result is an integral equation which is piecewise smooth and continuous in all derivatives within regions bounded by discontinuities in value or derivatives. In order to implement this result for image rendering an approximate solution in a discrete domain is used. One approach is to use a Monte Carlo type method of ray tracing. This is a view dependent method and any change in view demands a complete recalculation. View independent methods allow for a one time calculation and storage so that the scene may be rendered from any angle.
In order to accomplish view independent computation, a set of linear equations is determined. These equations define the various regions of the scene which are partitioned into localized areas with a dedicated equation of local support taking responsibility for the entire area. This equation need not be linear. It may be barycentric and thus distribute the calculated value across the area in a more realistic manner. The areas themselves are called master elements and typical implementation is to use triangular or quadrilateral shaped elements. A parametric mapping is also used to render a surface from a master element. The system of equations is created based on an error metric which is established between the approximate radiosity and the actual. Since the actual is unknown, the approximate is used on both sides of the equation and a weighting scheme implemented on the residual. An important arrangement of the terms of the equation is the grouping of emission elements. This grouping resulting in a term called the form factor. This represents the fraction of energy that leaves an element and arrives at another element. This interaction of elements is a key to the lighting of an environment.

The form factor of radiosity is an excellent metric for lighting in an environment in that it is based solely on geometry of elements. The reflectivity, color, texture, absorption and other properties of the surfaces are not contained in the term. In the use of form factors for rendering an image, changes in any other property other than environment geometry do not impact the image in an appreciable degree. This implies motion plays a major role in environment lighting in that it causes changes to the environment. A look at form factor use illustrates the effect of motion.

The form factor equation is given as

$$ F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos \theta_i \cos \theta_j}{\pi r^2} \cdot V_{ij} dA_j dA_i $$

where $V_{ij}$ is the visibility term between the two areas $A_i$ and $A_j$. Note the $\cos \theta$ terms. This indicates that the effect of one area on another diminishes as the horizon is approached. The inner integral may be replaced by an equivalent integral over a
hemisphere around $dA$ through the use of solid angles

$$d\omega_j = \frac{\cos \theta_j}{r^2} dA_j$$

and a hemisphere of directions $\Omega$. This gives an area-hemisphere equation

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{\Omega} \frac{\cos \theta_i}{\pi} V_{ij} d\omega_j dA_i.$$ 

This equation is conceptualized by placing a hemisphere over the area of interest. All other areas of the environment that are visible from this area are the defining bounds of a solid angle through the enclosing hemisphere. Thus the hemisphere is a geometric illustration of the energy level of the light emitting from the surface to other surfaces. The size, orientation and distance of the other surfaces define how much of the emitted energy reaches to their surface. Due to the computational complexity of determining the solid angle intersection with the hemisphere surrounding the emitting surface, an efficiency implementation is to use a hemicube over the area. The hemicube is usually defined with height 1 and top dimensions of 2 by 2. The hemicube is partitioned into a grid pattern and each grid is given a computed value based on the geometry. These values are stored in a lookup table. When computation is performed for other areas, the intersection of the solid angle to the center of the emitting area and the surface through the hemicube is determined and the lookup table consulted for each grid contained in the intersection. The total is assigned to the surface. In this manner the interaction of surfaces is modeled.

The point to be taken is that light in an environment is complex with many factors working in tandem. The key point is that each surface causes light changes to every other surface and thus moving objects change the distribution of light energy in the environment. The orientation and energy contained in the moving object causes changes or disturbances to the environment. These are the founding thoughts for this work. There are low level noises in an environment caused by the lighting. There are low level noises created by capture devices. These are static noise. The energy flow
theory defines the interaction of energy and objects. Motion of objects will cause changes in the energy flow. These changes are dynamic noise. Large changes are motion. A method of determining noise from motion is now investigated.

Video Methods

In order to achieve data reduction in video sequences, it is possible to obtain large reduction if the redundant or static data may be differentiated from the dynamic data. Data that does not change appreciably from one frame to the next does not need to be stored or transmitted. The noise of a video sequence makes the determination of static and dynamic data difficult. The added complication is that of static and dynamic noise. A method of characterizing the data is necessary.

A new method of motion determination was developed based on the distribution of errors between consecutive frames of the video. An uncompressed sequence of frames with no motion was created to study the inherent noise of the camera. A second uncompressed video sequence was also created in the same environment with motion for the study. Three data domains were studied: the original RGB domain along with the YIQ and YUV domains obtained from standard transformation from RGB. To obtain motion detection, localized processing was implemented in the form of 32 by 32 pixel neighborhoods or blocks. This block size is arbitrary for illustration. The developed methods may be applied to any size block that is large enough to support the calculations. A full application of this method may include various size blocks in separate phases to achieve containment of the moving objects and to provide multiple resolution for control of quality.

Let \( X_{ij}(t-1) \) represent a block of data from a frame of video at time \( t-1 \). Let \( X_{ij}(t) \) represent an equal sized block from a frame at time \( t \). The location in space of both blocks is the same, \( ij, 0 < i, j < N \). The error or difference is vector or matrix subtraction, \( Y_{ij} = X_{ij}(t-1) - X_{ij}(t) \). A histogram is created for the members of \( Y_{ij} \).
This histogram is the distribution of the data and differences in distribution is used to determine motion. Non-motion blocks are used to investigate the noise distribution. Some typical noise distributions are shown in Figures 3.1, 3.2 and 3.3.

The noise distribution figures show the respective distributions of the individual band depending on the domain. These are from blocks where no motion occurred and represent typical noise levels from the camera. These serve as a base background distribution set from which motion detection methods must work. The distributions of blocks where motion occurs is investigated. A drastic change in the distribution is present. A set of distributions from a block where motion was occurring are shown in Figures 3.4, 3.5 and 3.6.
The change in distributions between a non-motion block and a motion block are extremely evident in the RGB domain while the major changes in the YIQ and YUV domain is concentrated in the Y-band. In order to characterize the distributions of the differences, the autocorrelation of the differences within a block are examined. Let $e_{ij}$ and $e_{ij+1}$ be spatially consecutive elements of the error or difference block $Y_i$. The autocorrelation is $E(y_{ij}) = E(e_{ij} \cdot e_{ij+1})$. The autocorrelation is placed in a histogram to produce the distribution. Sample distributions of the autocorrelation in non-motion blocks are shown in Figures 3.7 and 3.8. Sample distributions of the autocorrelation of motion blocks are shown in Figures 3.9 and 3.10.

The figures used here have high concentrations about zero and were chosen for

Figure 3.4: Motion differences from a RGB block

Figure 3.5: Motion differences from a YIQ block

Figure 3.6: Motion differences from a YUV block
Figure 3.7: Autocorrelation distribution from RGB non-motion

Figure 3.8: Autocorrelation distribution from YIQ non-motion

Figure 3.9: Autocorrelation distribution from RGB motion
Many motion blocks have more widely scattered distributions that would not illustrate well in the paper. Just from this change in distribution a simple algorithm is capable of remarkable accuracy in determining motion. For example, a shift in the mean of the distribution is able to determine most motion blocks. It was found that non-motion block differences exhibit a mean at zero. An algorithm that used a threshold value to detect a shift of the mean away from zero produced very good results. An example frame of the video is shown in Figure 3.11.

The sample frame illustrates where motion was detected between the shown frame and...
and the next in the sequence. Painted blocks are where motion was determined and
dark blocks are where no motion is determined. The subject is walking from left to
right and thus the empty blocks in front of the subject will be entered in the next
frame. This also means that the blocks containing the back side of the subject will
be vacated. Blocks within the area of the subject which are not painted indicate that
not enough change occurred in the mean to indicate motion. These blocks indicate
that a previous frame block is adequate enough for use in painting this frame. The
use of multiresolution, in the form of larger and smaller blocks, would exclude this
behavior. An increase in sensitivity, by narrowing the threshold also excludes it.

Empty blocks behind the subject are suspect since no gross motion occurred but
the indicator was found. These are either "residual" of "ghost" imaging caused by re-
stabilization of the CCD in the camera or changes in the lighting of the environment
caused by the moving object, "shadow" or "reflection". Similar results are obtained
from using the Y-band in YIQ and are shown in Figure 3.12. Since the Y-band
transformation is identical for YIQ and YUV the same result would follow from
the algorithm in the YUV space. Further study of the video frames indicates that
the method is actually identifying lighting changes. The motion of the actor causes
shadows and reflections to change the lighting in the environment where the method
has identified. It was found that this feature could be either included or excluded
from the motion identified blocks by changing the threshold of the mean shift. This
demonstrates the viability and utility of the method as well as demonstrating the
sensitivity. Hard shadow and reflection would be harder to remove from actual object
motion and are themselves motion objects.

The mean shift method of motion detection has merit in that it is a simple method
and therefore very quick. The use of a threshold value to detect the change in the mean
provides a variable parameter to the method that allows for easy adjustments. In the
test video it was found that the shadows and reflections caused by the moving objects
could be included or excluded from the motion identification. Since reflection and shadow are typically low intensity items in an environment, their motion is very close to the static noise of the video data. By lowering the threshold, they may be included in the motion identified blocks. This allows for use in sensitive low level motion detection. The successful identification of shadow and reflection motion illustrates the capability of the method to detect changing lighting conditions as outlined in the energy flow theory. The application of autocorrelation in this manner is extended to still images or single frames of a video where enough movement occurs to render the outlined method inefficient. If there is considerable movement between video frames such as a moving camera angle or scene change, then an individual frame method based on these methods may be used to reduce the data. The method may also be used to locate or identify edges or features of interest within the image.

Digital still images are composed of numerical data arranged in a specific order. These values represent intensity measures in respective domains. The specific order defines the location of the intensity value within the image. Constant values across consecutive regions indicate an area of the image where no change is occurring and
the only feature is "no change." This feature is easily represented and thus easily compressed. Various values across consecutive regions and within a specific region indicate some feature of the image. It is these areas that are more difficult to compress and the feature may be of interest. Thus a quick and simple method of identifying regions with little or no change from areas with significant change is desired.

Autocorrelation of data is a standard measurement used in various fields for many purposes. Typical use is to determine the amount of similarity between close neighbors in the data. This method indicates the very localized behavior from one point to the next. In a two dimension image a single datum usually contains eight neighbors, image edge data being the exception. While methods employing this resolution of processing are very useful, they are also more computationally intensive than necessary in featureless regions. A simple, fast method of identifying regions with important features is to use autocorrelation between the differences from the mean of the region and the individual data it contains. This autocorrelation is a distribution and distribution methods are used to characterize the region.

Still Methods

The digital image data is processed in a block by block manner. This allows for quick region identification where smaller block processing may be used or other more intensive methods employed. For this work the blocks were 32 by 32 in size. The block size may be set dependent on the types of images so that successful feature versus featureless containment will occur. Let \( m_{ij} \) be the mean of a block of data \( ij \). Let \( Y_{ij} = x_{k} - m_{ij} \) be the error or difference vector or matrix between the actual data and the mean. Let \( E(y_{ij}) = E(e_{ij} \ast e_{ij+1}) \) be the autocorrelation differences of the elements of \( Y \). An initial pass through the block data is performed to determine the mean and then a second pass to compute the difference from the mean. A third pass determines the auto-correlation between adjacent differences in the image horizontal
direction which is arranged in a histogram. Feature region identification is then as simple as range or variance checking. A threshold value to account for noise in the data should be implemented since smooth regions will contain some variance. The range check is a quick initial method to identify smooth regions with the variance check used secondly to help eliminate smooth regions with "rogue" values.

The image used here for illustration is the standard color Lenna. Various other images from standard databases along with camera and scanner images were used and produced similar results. The effect of color on the method adds a small consideration to the implementation. In the typical red-green-blue (RGB) domain results may vary considerably depending on which color band is used for a particular image. This may be normalized by processing the Y-band from a standard YIQ domain. Since the Y-band itself is sufficient, the complete YIQ transform is unnecessary. Two regions of the image are used to illustrate the results. They are shown in Figure 3.13 labeled as 'feature' and 'featureless' indicating the areas producing the results.
To demonstrate the results from different domains both RGB and YIQ samples are shown. The featureless region has results shown in Figures 3.14, 3.15 and 3.16. Although the results show differences, the characteristics of the distributions are similar. The distinctive mean and small variance is very typical of featureless regions. The blue band of the region has a minimum value of -9 and a maximum of 44 with a variance of 5.35. The green band has minimum -68, maximum 193 and variance 21.28. The red band has minimum -173, maximum 273 and variance 39.42. Exact values are not necessary so truncation and rounding is present to simplify calculations.

The feature region was drawn from an area adjacent to the featureless region. This allows for better comparison. The lighting of adjacent regions is similar and the basic area similar. The feature itself is the primary difference and its size does not dominate the region. Its presence, however, does constitute a major impact on the resulting distribution. Compare the characteristics of Figures 3.17, 3.18 and 3.19 with those of the featureless regions. The differences are obvious and of sufficient magnitude to be easily differentiated. The blue band has minimum -226, maximum 12807 and variance 2305.67. The green has minimum -153, maximum 5774 and variance 1071.24. The red has minimum -152, maximum 1338 and variance 266.98.
Figure 3.15: Green band results from featureless region

Figure 3.16: Red band results from featureless region

Figure 3.17: Blue band results from feature region

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The results are similar in the YIQ transformed domain as well. The three color bands can be processed in parallel if the cost of transformation is deemed too great. However, if the image data is transformed to another domain as part of another process, such as compression, that domain could be just as informative in this process. A sample from the featureless region is shown in Figure 3.20 and the feature region results are shown in Figure 3.21. Once again the differences are obvious and drastic. The featureless region has minimum -33, maximum 105 and variance 12.22 while the feature region has minimum -116, maximum 6749 and variance 1233.83. Given the numerical disparity of these two sample regions it is a simple matter to computationally determine which has a feature and which does not.

The example "Lenna" image was processed so that a range of difference for a block autocorrelation less than 512 indicated a smooth block. This is approximately
a deviation from the local mean of 16. The blocks deemed smooth by the method are blacked out and the result is shown in Figure 3.22.

Given the nature of the "Lenna" image and the background detail, many blocks remain. An increasingly smaller block size method is suggested to narrow the resolution and identify more data that can be reduced. To illustrate the performance of the still method a single frame of the video sequence was chosen. The same criteria was used for the computation as that of the "Lenna" image. The original video frame is shown in Figure 3.23. The results of the computation are shown in Figure 3.24.

Summary

The reduction of data necessary for video and still images relies on identifying redundant or unnecessary data. In order to use reduction for real time processing,
Figure 3.22: Still image feature results of Lenna

Figure 3.23: Sample frame from video sequence
this identification process must be fast. The nature of the images is explained and modeled by energy flow theory. With this philosophy considered, a new method of characterizing image data was developed. The use of the autocorrelation distribution of differences as a characterizing method is both efficient and successful in the video and still image domain. What remains is to perform data reduction in the areas that are characterized as non-redundant or necessary. This is addressed in the next chapter.
CHAPTER 4

VECTOR QUANTIZATION

Large volumes of data from a similar domain such as image or sound data may be processed for reduction in a grouped manner. In the audio domain these groups are linear in time and in the image domain they may be linear in either of two dimensions or block like incorporating both dimensions. The video domain adds a third dimension which could also be used linearly or in combination with the two dimensions of the individual images. An area where groups of data are processed for reduction is known as vector quantization (VQ). The arrangement of the data in the target domain can be determined arbitrarily. The process of VQ is simple in that the data is processed in groups herein called test vectors and these are mapped to representative vectors comprising a codebook. In general, some measure of closeness is used to do the mapping. This is the analysis phase. The index of the closest codebook vector is stored or transmitted and used to reconstruct the data. This is the synthesis phase. Unless the codebook and data have perfect correspondence, this method is lossy since perfect reconstruction is impossible. The objective is to have close enough reconstruction such that the loss or error is undetectable. VQ may be used for both off-line processing where all data to be reduced is known and for on-line processing where reduction is performed on data as it is presented. This distinction is important since a codebook may be optimized for known data but must be more tolerant for arbitrary data. The concern here is for on-line use. Thus the discussion and development is towards that end.
Background

VQ has been explored and developed for many years. Some of the key methods date to the early 1980s. This implies that the method was considered earlier and in fact some theory may be found much earlier. The concept itself is simple enough. A codebook of size $N$ vectors of length $k$ is used to represent a possibly infinite $k$-dimensional space. This is similar to the scalar quantization which occurs from the analog to digital conversion with the added complexity that the extra dimensions bring along. The major complexities involved with VQ methods are codebook design and test vector mapping. The codebook design is critical to get good reconstruction and the mapping is critical to implementation speed.

Codebook Design

The design of a good codebook includes the size of both the vector length and the size of the codebook. The color quantization methods outlined above typically target 256 three dimensional vectors as a codebook. An increase in vector length and codebook size increases the difficulty in mapping to the codebook. A larger codebook allows for better reconstruction but carries an increase in matching search. Larger codebooks also reduce the reduction ratio since there are more code indices possible and this requires more bits to represent. Longer vectors also add to the search time but allow for higher reduction of the necessary reconstruction data. Longer vectors also complicate the matching process since there are more fields that must align closely with the data. Balancing the size and length of the codebook and vectors can be itself a difficult task but known characteristics of the data to be reduced help determine these values. Since many design paradigms are general in nature, this issue is left open to implementation. Here we look at the construction of a codebook itself and later address the searching.
Most popular codebook designs are implemented in data dependent manners. An entire field of methods may be grouped as training or learning paradigms. The most common is the LBG [42] algorithm. In this method, a codebook of size $N$ and length $k$ is chosen in an arbitrary way, usually a random sampling of the target data is used to "seed" the algorithm. Test vectors are then matched to the closest codebook entries and grouped. The groups are processed to determine their centroids and these are the new codebook entries. The test vectors are then matched to the new entries and the process iterates until some stopping criteria is met. Usually a change in codebook vectors or error measurement less than a given $\varepsilon$ is used. The convergence depends on the quality of the initial arbitrary codebook. Training or learning is also implemented by neural network methods. The quality of these codebooks depends on the representative quality of the training test vectors. If good quality training test vectors are available another paradigm called a splitting method can be used.

A splitting method is basically a division of the vector data domain into partitions. In one dimension these partitions are ranges, in two they are areas, in three they are volumes and above three they become harder to envision but can be thought of as hyper-areas that are represented by a codebook entry. The methods take the known data vectors and recursively divide them into smaller groups until the codebook size is reached and the centroids of the partitions are the codebook entries. The division may or may not put equal amounts in each new region. Simple methods use one dimension of the vector as the determining factor for the split and more complex methods incorporate more than one dimension at a time. The splitting may be done with a single division creating two new groups or many divisions at the same time creating many new groups. This paradigm has many variations and application areas and thus there is a rich body of literature on the subject.

An emerging field of codebook design is based on relatively data independent methods. The codebook is designed as a fixed entity for all data within a given
domain. This is born of a need for quick processing of vast data on-line. A separate codebook for each arbitrary block of data is too costly to develop and thus a general purpose codebook is desired. A system of this design allows for faster storage and transmission since the codebook need not be stored or transmitted with the data and optimization of the processing may be implemented. The inherent nature of codebook use allows for some loss. The quality of all lossy methods becomes subjective and thus good enough becomes a goal. The fixed codebook design is not a new concept. Simple uniform quantization is one of these methods and has a long history. The emerging interest and research is in higher quality through better design and method.

Quantization Methods

The actual use of a codebook, once it is determined, involves mapping the test vectors to a codebook entry. The simplest implementation is a brute force search for the closest match based on the desired error measure. Various improvements on this design for VQ exist [28]. There are methods that perform some processing which may be called normalizing. Other methods address the undesired effects caused by processing the data in fixed lengths or blocks. Some methods use a multistage system while others use interpolation. More involved methods incorporate many methods into hybrid systems. A short description of the various methods is included here.

To address the searching problem, tree based implementations may be used. This is a greedy algorithm approach where the best branch is chosen at each step. Both balanced and unbalanced tree methods exist [55]. These methods result in local optimization but decrease the search time. Unbalanced methods may degrade into exhaustive search. Fast search methods remain an open area of research.

One type of multistage VQ is to map the test vector to a codebook entry and then compute an error vector from the choice. This error vector is then processed by VQ and both the original codebook index and the error codebook index is stored.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
or transmitted. The reconstruction phase adds the vectors together to produce the resultant vector. Obviously this can be carried to as many stages as desired. The difficulty is in designing multiple codebooks that are interrelated so one works well with the residual result of the other.

Some methods perform normalizing by subtracting the vector mean before VQ is performed. This is called separating mean VQ. An approach closer to true normalization subtracts the mean and then divides by the variance. This is called gain-shaped VQ. Another form of gain-shaped VQ involves minimizing the mean square error

\[ d(x, g, s) \]

where \( g \) is gain defined as

\[ g = \left[ \sum_{m=1}^{k} (x_m)^2 \right]^{\frac{1}{2}} \]

(4.2)

and \( s \) is a shape code vector with unit gain

\[ \left[ \sum_{m=1}^{k} (s_m)^2 \right]^{\frac{1}{2}} = 1 \]

(4.3)

The method works by maximizing \( xs^t \) and minimizing \( g - 2gx^t \). The indices of the gain and shape code vector are used for reconstruction.

Hierarchical VQ works with blocks of data. A threshold is determined and the variance of a data block computed. If the variance exceeds the threshold the block is subdivided into smaller blocks. The sub-block variances are computed and checked against the threshold. The process may repeat into ever smaller blocks. Separate codebooks exist for each block size and code bits used to indicate the vector lengths. The codebook vectors are typically of a smooth shape and high activity regions are divided further. Autocorrelation holds that small regions will be relatively smooth.

An interpolative VQ uses Pulse Code Modulation (PCM) where a single value from a block is retained intact and the difference between it and the rest of the block forms an error vector which then under goes VQ. The calculations are reversed for reconstruction of the data.
Entropy-constrained VQ uses variable length codewords for the codebook indices based on their probability distribution. This is a successful hybrid implementation but only for off-line processing and involves more than one pass through the data.

A lapped VQ has been developed [67] where the decoder vectors are longer than the encoder vectors. The overlapping reconstruction helps to eliminate the blocking effect known as Gibb's phenomena from the reconstructed data.

Predictive VQ has received much attention [45],[47],[15],[75],[46],[56],[17], [3] and has been used successfully in audio, image and video coding. Its implementation is similar to Digital Pulse Code Modulation (DPCM) with the substitution of vectors for scalars. A current test vector and a predicted vector, based on the last coded vector, are compared. The error vector is then sent through VQ and it is combined with the last predicted vector to create the new predicted vector and the process repeats. The reconstruction uses a reconstructed vector to predict the next vector. This is combined with the error vector from the codebook to create the next reconstructed vector and the process repeats. The system is "seeded" with the first test vector to start the process. The linear nature of audio data has determined a convention of method in that field. Many methods exist based on linear prediction (LP) and are grouped as Analysis-by-Synthesis (AbS) schemes.

In AbS there exists two general areas, long time prediction (LTP) and short time prediction (STP). These come from the nature of the data arrangement in time. There is a long time element which is periodic and short time variance which appears periodic. The schemes are Codebook Excitation (CELP), Self-Excitation (SELP), Regular Pulse Excitation (RPELPC) and Multi-Pulse Excitation (MPLPC).

The CELP uses a codebook of vectors in conjunction with associated scaling or gain vectors. The best match based on a predetermined error metric is chosen. SELP is an adaptive codebook method using a LTP system. Initially a starter sequence is randomly chosen and then searched for a sequence of correct length, L, that minimizes
the error. This vector is used to code the current data and then replaced into the system and the oldest L elements discarded. The system acts as a window into the sequence with the closest sub-sequence serving as the coding vector. The replacement phase creates the adaptability of the method. The difficulty is in determining the optimal sequence length to capture the characteristics of the data. In the MPLPC method a small set of pulses (relatively large amplitudes or values) in the vector is used as the critical match points. The encoding is based on the determination of the positions of the pulses and the gain scaling that minimizes the error. A typical ratio used is around one pulse per eight samples. Three major methods are employed in MPLPC. One matches a position and optimizes its amplitude in a sequential stepwise fashion. The second holds previously located positions constant but optimizes amplitudes for all determined positions at each step. The third refrains from optimizing the amplitudes until all positions are fixed. The second method is reported to work best but is compromised by the quantization phase and thus the third method is preferred due to the single global optimization. A drawback to this method is that large number of bits required to encode the positions determined. A large codebook CELP is effectively a MPLPC. A small codebook CELP is a MPLPC with hard restrictions on positions and amplitudes. If only the positions are restricted and the amplitudes allowed to vary we have the RPELPC method. In RPELPC the pulse positions are spaced at regular intervals within a codebook vector and subsequent vectors contain phase shifted versions of the same vector. Both the multi- and regular pulse methods contain a difficulty with the determination of pulses per sample length. High pitch audio requires more pulses per length than does low pitch and compromises are made.

Transform VQ is applied to coefficients resulting from a transform operation [41]. A transform such as the Discrete Cosine Transform (DCT) is used on separate blocks of data. The DC coefficients are grouped into vectors and spatially identical located bands of the transformed data is grouped into vectors. This implies many
codebooks of differing size. If adaptability in vector size is added then category identification bits must be incorporated. A simplification to this method is to only perform transformation in a row oriented manner within the block and then use vectors aligned with the columns of the block. An extension to the method is to group spatially corresponding coefficients from many blocks into vectors. Since the original transform occurs in the actual spatial domain, this is an application of VQ to the temporal domain.

A hybrid system combines transformation on the prediction errors of the predictive VQ with VQ. Combinations and adaptations resulting in hybrid methods seem endless. The only hard limits are memory size and computation time. With off-line processing research can continue to profit from faster machines. A counterpart to this is the current interest in on-line processing for transmission. The objective is to have a robust quality VQ in real time. This is an objective of the dissertation.

Problem

The searching of the codebook in VQ is a major bottleneck in the implementation. Long length vectors compound the problem by adding to the computational overhead. Large codebooks provide good coverage of the domain but lengthen the search time. A fast vector quantization codebook search is desired. To search for a solution, the nature of the behavior of the VQ process is needed. A formal analysis of the vector quantization is performed here providing for better codebook searching methods.

A normal distribution with mean 0 and variance 1 is contained in a $k$ dimensional space. For this dissertation $k = 40$ will be used as the running problem and solution example, but the problem and solution are not tied to this dimension size. Specific implementations later will use other sizes. Sample test vectors are processed from this distribution and the desire is to represent the test vectors with a vector drawn from a
small representative set of vectors from the space. This representative set is a code­
book of the space and the nearest code vector to the test vector is used to represent
the test vector. Such practice is common in compression algorithm methods. The
codebook is generated by a random number generator and all elements are considered
independent, i.e., no correlation exists between vector elements or between vectors.
Nearness is determined by a standard Euclidean metric (sum of square differences).
Since a vast number of test vectors is to be processed it is important to process them
in a fast manner. The exhaustive method of testing every test vector against every
code vector is an \( O(kN) \) operation where \( k = 40 \) and \( N = 1024 \) in this case. Since
there is quantization error inherent in the codebook paradigm, a heuristic method of
determining nearest neighbor is acceptable. The optimum parameters will be speed
versus accuracy. Speed is determined as reduction in operation when compared to
the exhaustive method. Accuracy is determined relative to the exhaustive method.
For example, if every pairing occurs in the heuristic that occurs in the exhaustive,
accuracy is 100% and if it takes one half as many operations to achieve this accuracy
then speed efficiency is improved.

Development

A general purpose VQ able to handle data in real time is the desired target of
this research. Audio and video data is the area of interest but a more general method
is of greater value. Since data vectors may be normalized easily and quickly, we shall
focus on data from a normal distribution with mean zero and variance one, \( n(0,1) \).

**Theorem 1**

Let \( X_{i,j}, j = 1, \ldots, k; i = 1, \ldots, N \) be \( N \) vectors with \( k \) components each. \( X_{i,j} \)
are independent identically distributed (i.i.d) \( \sim n(0,1) \). The \( N \) vectors constitute a
codebook. New vectors \( Y = [Y_1, Y_2, \ldots, Y_k] \), with \( Y_j \) being i.i.d \( \sim n(0,1) \), and also
\[ Y_j, j = 1, \ldots, k \text{ being independent of } X_{i,j}, j = 1, \ldots, k, i = 1, \ldots, N. \text{ Then} \]

\[
Z_i = \sum_{j=1}^{k} (X_{i,j} - Y_j)^2 / 2 \sim \chi^2(k)
\]

The Euclidean distance of the vector \( Y \) from a codebook vector is a random variable \( D \), with probability density function

\[
h(d) = \frac{d^{k-1}e^{-d/2}}{\Gamma\left(\frac{k}{2}\right)2^{k-1}}
\]

The expected Euclidean distance of the vector \( Y \) from a codebook vector is

\[
\frac{2\Gamma\left(\frac{k+1}{2}\right)}{\Gamma\left(\frac{k}{2}\right)}
\]

and the variance of this Euclidean distance is

\[
2k - \left(\frac{2\Gamma\left(\frac{k+1}{2}\right)}{\Gamma\left(\frac{k}{2}\right)}\right)^2
\]

Furthermore the probability density function of the random variable \( U = \min(Z_i, i = 1, \ldots, N) \) is \( h(u) = n(1 - G(u))^{n-1}g(u), 0 \leq u \leq +\infty \) where \( G(u) \) is the probability distribution function of \( Z_i \) and \( h(u) \) is the probability density function of \( U \).

**Proof**

Since \( X_{i,j} \) are i.i.d with mean zero and variance 1, and \( Y_j, j = 1, 2, \ldots, k \) are also i.i.d \( \sim n(0,1) \). Also since \( X_{i,j} \) and \( Y_j, j = 1, \ldots, k \) are independent, the \( X_{i,j} - Y_j \) are normally distributed. \( E(X_{i,j} - Y_j) = E(X_{i,j}) - E(Y_j) = 0 - 0 = 0 \). The variance of \( X_{i,j} - Y_j \) is

\[
\sigma^2_{X_{i,j}-Y_j} = E(X_{i,j} - Y_j)^2
\]

\[
= E(X_{i,j}^2) - 2E(X_{i,j}Y_j) + E(Y_j^2)
\]

\[
= 1 - 2E(X_{i,j})E(Y_j) + 1
\]

\[
= 2 - 2 \cdot 0 \cdot 0
\]

\[
= 2
\]
due to independence of $X_{i,j}$ and $Y_j$. Hence $X_{i,j} - Y_j \sim n(0,2)$. This implies that \( \frac{X_{i,j} - Y_j}{\sqrt{2}} \sim n(0,1) \) and $\frac{(X_{i,j} - Y_j)^2}{2} \sim \chi^2(1)$ is a chi-square with one degree of freedom.

From this last result we conclude that

$$Z_i = \frac{\sum_{j=1}^{k} (X_{i,j} - Y_j)^2}{2} \sim \chi^2(k), i = 1, \ldots, N$$

The probability density function of $Z_i$ is

$$f(z_i) = \frac{z_i^{\frac{k}{2} - 1} e^{-\frac{z_i}{2}}}{2^{\frac{k}{2}} \Gamma(\frac{k}{2})}, z_i \geq 0$$

The mean of the above distribution is $k$ and the variance is $2k$. Which implies that the average squared Euclidean distance of a newly generated vector of $k$ i.i.d $\sim n(0,1)$ components from an arbitrary vector in the codebook, is $2k$ and the variance is $4 \cdot 2k = 8k$. If $k = 40$ then the average distance square is 80 with variance equal to 320 or the standard deviation equal to $8\sqrt{5}$. This implies the distances are expected to have a relatively large fluctuation. The Euclidean distance of the vector $Y$ from a codebook vector is a random variable, denoted here by $D$. The expected Euclidean distance of the vector $Y$ from a codebook vector is:

$$E(D) = \int_0^\infty \frac{\sqrt{2} z_i^{\frac{k}{2} - 1} e^{-\frac{z_i}{2}}}{2^{\frac{k}{2}} \Gamma(\frac{k}{2})}, z_i \geq 0$$

The value of the above integral is:

$$E(D) = \frac{2\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})}$$

The variance of the Euclidean distance is:

$$\sigma_D^2 = E(D^2) - E^2(D)$$

$$E(D^2) = \int_0^\infty \frac{2z_i^{\frac{k}{2}} e^{-\frac{z_i}{2}}}{2^{\frac{k}{2}} \Gamma(\frac{k}{2})}, z_i \geq 0$$

or

$$E(D^2) = 2k$$
Hence

$$\sigma_D^2 = 2k - \left(2 \frac{\Gamma\left(\frac{k+1}{2}\right)}{\Gamma\left(\frac{k}{2}\right)}\right)^2$$

Note that $\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$. For $k=40$, the expected Euclidean distance of the vector $Y$ from a codebook vector is:

$$E(D) = \frac{\sqrt{\pi} \cdot 39 \cdot 37 \cdot \ldots \cdot 1}{38 \cdot 36 \cdot 34 \cdot \ldots \cdot 2} = 8.888550$$

and the variance of $D$ is:

$$\sigma_D^2 = 80 - \left(\frac{\sqrt{\pi} \cdot 39 \cdot 37 \cdot \ldots \cdot 1}{38 \cdot 36 \cdot 34 \cdot \ldots \cdot 2}\right)^2 = 0.993673$$

Let $H(d)$ denote the probability distribution function of the random variable $D$. Then

$$H(d) = P(D \leq d) = P(\sqrt{2Z} \leq d) = P(2X \leq d^2) = P(X \leq \frac{d^2}{2}) = F\left(\frac{d^2}{2}\right)$$

Where $F$ is the probability distribution function of a Chi-square with $k$ degrees of freedom.

From the above we have that the probability density function $h(d)$ is

$$h(d) = df(d^2) = \frac{d^{k-1}e^{-\frac{d^2}{4}}}{\Gamma\left(\frac{k}{2}\right)2^{k-1}}$$

Now let $U$ be a random variable defined as follows:

$$U = \min(Z_1, Z_2, \ldots, Z_N),$$

where $Z_1, Z_2, \ldots, Z_N$ are random variables denoting the squared Euclidean distance of a newly generated vector $Y = Y_1, Y_2, \ldots, Y_k$, from the 1st, 2nd, ..., $N$th, quantization vectors of the codebook, respectively.

Then

$$H(u) = P(U \leq u)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[ = 1 - P(U > u) \]
\[ = 1 - P(\min(Z_1, Z_2, \ldots, Z_N) > u) \]
\[ = 1 - P(Z_1 > u, Z_2 > u, \ldots, Z_N > u) \]
\[ = 1 - P(Z_1 > u)P(Z_2 > u) \cdots P(Z_N > u) \]
\[ = 1 - (1 - P(Z_1 \leq u)) \cdots (1 - P(Z_N \leq u)) \]

Let \( G(u) \) be the probability distribution of \( Z_i \), in other words, \( G(u) = P(Z_i \leq u) \), \( i = 1, 2, \ldots, N \). Then

\[ H(u) = 1 - (1 - G(u))^N, 0 \leq u < +\infty \]

In order to obtain the probability distribution function from the above, we have to take the first derivative with respect to \( u \). Thus

\[ h(u) = -N(1 - G(u))^{N-1}(-g(u)), 0 \leq u < +\infty \]

hence

\[ h(u) = Ng(u)(1 - G(u))^{N-1}, 0 \leq u < +\infty \]

where \( g(u) \) and \( G(u) \) are the probability density and probability distribution functions of \( Z_i, i = 1, 2, \ldots, N \), which are \( \chi^2(k) \).

The above result was simulated in the computer. 1024 vectors of size 40 each were simulated. The components of each vector were random variables, \( i.i.d \sim n(0, 1) \). Then 100 random vectors were simulated, each vector having 40 components of independently distributed random numbers having normal distribution with mean 0 and variance 1. The histogram of the square Euclidean distances has mean 80 and standard deviation \( 8\sqrt{5} \), as shown in Figure 4.1. The histogram of Euclidean distances is shown in Figure 4.2. The mean is indicated by a partition line at 8.896078. For this figure the variance was 0.957703. The histogram of the minimum distances has also been simulated and shown in Figure 4.3.
Figure 4.1: Histogram of square Euclid distances

Figure 4.2: Histogram of Euclid distances

Figure 4.3: Histogram of minimum distances
Definition

Consider a random vector $U = (U_1, U_2, ..., U_k)$, where $U_i$, i=1,2,...,k, are continuous random variables over the real line, i.i.d, and let $V_1, V_2, ..., V_N$ be N random vectors, each having k components. Assume further that the measure of the $P(U_i = 0) = 0$, then $U_i$, and $V_j$, have a sign agreement if they are both positive or both negative, otherwise have a sign disagreement.

Theorem 2

Let $V_1, V_2, ..., V_N$ be N random vectors, each having k-components, which constitute a random codebook. The components of each of the N random vectors are continuous random variables which are i.i.d over the real line, with the measure of the $P(U_i = 0) = 0$, and $P(U_i > 0) = P(U_i < 0) = 0.5$. Let $U = (U_1, U_2, ..., U_k)$, be a random vector, where $U_i$, i=1,2,...,k, are continuous random variables over the real line, i.i.d. Then the number of sign agreements between the vector U and an arbitrary vector $V_i$ from the codebook is a random variable $X_i$ with a binomial probability function $f(x_i) = P(X_i = x_i) = \binom{k}{x_i} \left(\frac{1}{2}\right)^k$. Now let $Y = max[X_1, X_2, ..., X_N]$ then the probability function $h(y)$ of the random variable Y is $h(y) = f(y)(F^{N-1}(y) + F^{N-2}(y)F(y-1) + ... + F(y)F^{N-2}(y) + F^{N-1}(y))$

Proof

Consider a codebook of N vectors, each vector having k random numbers. The probability of a random number being positive is equal to the probability of being negative. Now let’s consider a random vector of k random numbers. The question is “what is the maximum expected number of orientation matches between the codebook and a random vector?”

If in position m of the vector, the entry is negative, we can denote that with a zero. Similarly, if it is positive, we can represent it with a one. Thus the random vector of k-components can be represented as a binary number of length k. There are therefore $2^k$ possible configurations. Now consider an arbitrary random vector
with negatives at positions $i_1, i_2, ..., i_t$ and positives in all other positions. How many matches is this vector expected to have with the codebook?

Before we answer this question, we will consider two random vectors and the number of expected matches between them. The problem is similar to having two sets of $k$ coins which are numbered from 1 to $k$ and you toss them independently. There is a match in the $i^{th}$ position if the $i^{th}$ coin of the first set and the $i^{th}$ coin of the second set show the same outcome. The probability for that to happen is $\frac{1}{2}$ since the outcomes of the two coins are HH, TT, HT, and TH (Heads, Tails). So out of the $k$ positions of the two vectors $\frac{k}{2}$ will match and $\frac{k}{2}$ will not match.

The probability for $x$ matches out of the $k$ possible $P(X = x) = \binom{k}{x} \left(\frac{1}{2}\right)^k$. $X_1, X_2, ..., X_n$ are random variables denoting the number of matches between the random vector and each one of the $N$ codebook vectors. Then

$$P(X_i = x_i) = \binom{k}{x_i} \left(\frac{1}{2}\right)^k$$

Let

$$Y = \max \{X_1, X_2, ..., X_N\}$$

$$H(y) = P(Y \leq y)$$

$$= P(X_1 \leq y, X_2 \leq y, ..., X_N \leq y)$$

$$= [P(X \leq y)]^N$$

$$= [F(y)]^N$$

$$h(y) = H(y) - H(y - 1)$$

$$= [F(y)]^N - [F(y - 1)]^N$$

$$= [F(y) - F(y - 1)] \cdot$$

$$\left[ F^{N-1}(y) + F^{N-2}(y) F(y - 1) + \cdots \right]$$

From the above we obtain:

$$[Nf(y) F^{N-1}(y - 1) \leq h(y) \leq Nf(y) F^{N-1}(y)]$$
How many matches does the minimum Euclidean distance have? What is the distance of the maximum matches compared to the minimum Euclidean distance? A computer simulation was performed to compare the maximum sign matched vectors and the minimum Euclidean distance match. Figure 4.4 shows a histogram of the maximum sign match average found for 1000 test vectors against the codebook. The sign matching is also shown in Figures 4.5 and 4.6 where a single test vector is used and the number of sign matches plotted in a histogram. This indicates the average behavior of sign matching on a single vector.
Theorem 3

Let $X = [X_1, X_2, \ldots, X_k]$, be a random vector, of i.i.d random components which have the standardized normal distribution. Let $U = [U_1, U_2, \ldots, U_k]$, be also a random vector of i.i.d random components which have the standardized normal distribution. Then the mean square error of two components having the same sign orientation is:

$$E[(X_i - U_j)^2 | X_i > 0, U_i > 0] = 2 - \frac{4}{\pi}$$

and the mean square error of two components having different sign orientation is:

$$E[(X_i - U_j)^2 | X_i > 0, U_i < 0] = 2 + \frac{4}{\pi}.$$ 

Thus if $a$ is the number of components with the same sign orientation then $k - a$ is the number with different sign orientation and the total mean square error is:

$$(2 - \frac{4}{\pi})a + (2 + \frac{4}{\pi})(k - a)$$
Proof

\[
\begin{align*}
    f(x|X > 0) &= \frac{f(x)}{P(X > 0)} \\
    &= \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \\
    &= \sqrt{\frac{2}{\pi}} e^{-\frac{x^2}{2}}, x > 0
\end{align*}
\]

\[
E(X|X > 0) = \sqrt{\frac{2}{\pi}} \int_0^{+\infty} xe^{-\frac{x^2}{2}} dx = \sqrt{\frac{2}{\pi}}
\]

\[
E((X_i - U_i)^2 | X_i > 0, U_i > 0) \\
= \frac{2}{\pi} \int_0^{+\infty} \int_0^{+\infty} (x_i - u_i)^2 e^{-\frac{x_i^2}{2}} e^{-\frac{u_i^2}{2}} dx_i du_i \\
= \frac{2}{\pi} \int_0^{+\infty} \int_0^{+\infty} (x_i^2 + u_i^2 - 2x_iu_i) e^{-\frac{x_i^2}{2}} e^{-\frac{u_i^2}{2}} dx_i du_i \\
= \frac{2}{\pi} (2) \int_0^{+\infty} x_i^2 e^{-\frac{x_i^2}{2}} dx_i \int_0^{+\infty} e^{-\frac{u_i^2}{2}} du_i \\
- \frac{4}{\pi} \int_0^{+\infty} x_i e^{-\frac{x_i^2}{2}} dx_i \int_0^{+\infty} u_i e^{-\frac{u_i^2}{2}} du_i \\
= \frac{4}{\pi} \left( \frac{\pi}{2} \right) - \frac{4}{\pi} \\
= 2 - \frac{4}{\pi}
\]

Similarly:

\[
E((X_i - U_i)^2 | X_i > 0, U_i < 0) = 2 + \frac{4}{\pi}
\]

This shows that the error committed when the test vector and the codebook vector are in sign agreement on an element is much smaller than when they disagree. The agreement error is less than 1 and the disagreement error is greater than 3. Since the data is normalized, it will be un-normalized in implementation. The un-normalizing operation includes a multiplication by the standard deviation. A multiplication by a
value less than 1 will cause much less error than the multiplication by a value greater than 3. This shows the components that constitute the mean square error calculation are vastly different but undistinguished in the MSE. We prefer small errors to large ones and thus desire to control the process.

If \( k = 40, a = 20, k - a = 20 \) then the square error is \( (2 - \frac{4}{4}) \cdot 20 + (2 + \frac{4}{4}) \cdot 20 = 80 \) which is the same as the \( \chi^2 \).

If \( k = 40, a = 30, k - a = 10 \) then the square error is \( (2 - \frac{4}{4}) \cdot 30 + (2 + \frac{4}{4}) \cdot 10 = 80 - \frac{80}{\pi} \).

Let the probability that the maximum match vector will miss a certain orientation be \( P \). For a codebook of 1024 members and vectors of 40 elements of independent identically distributed normal random numbers, simulation results have shown that the maximum matches are between 28 and 34. The probability of a miss is about \( \frac{1}{4} \), so there is a probability \( P \) for the maximum matching codebook vector to miss the orientation of largest value vector. Let \( P_i \) be the probability for an entry in the random vector to have absolute value greater than 1. In the case of the normal distribution

\[
P(|x| > 1) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-1} e^{-\frac{x^2}{2}} \, dx + \frac{1}{\sqrt{2\pi}} \int_{+\infty}^{+\infty} e^{-\frac{x^2}{2}} \, dx
\]

\[
= \frac{2}{\sqrt{2\pi}} \int_{1}^{+\infty} e^{-\frac{x^2}{2}} \, dx
\]

\[
= p_1
\]

If the random vector has \( k \) components then \( kp_1 \) are expected to have values with absolute value greater than 1. Any disagreement in orientation between the random vector and the codebook would increase the mean square error drastically, so it is important to find the vector that matches as many of these vectors as possible. Let \( i_1, i_2, ..., i_m \) be \( m \) positions with values \( |x_1| > |x_2| > \cdots > |x_m| > 1 \). A random vector and any vector from the codebook are expected to match the orientation of \( |\frac{m}{2} \) of these...
components. The probability for two arbitrary vectors to match all \( m \) components is 
\[
p_2 = \left( \frac{1}{2} \right)^m
\]
and the probability not to match them is 
\[
q_2 = 1 - \left( \frac{1}{2} \right)^m.
\]
The probability that none of the \( N \)-vectors matches all \( m \)-components is 
\[
(q_2)^N = (1 - \left( \frac{1}{2} \right)^m)^N.
\]
Thus \( p_3 = \left( 1 - (1 - \left( \frac{1}{2} \right)^m)^N \right) \).

For \( m = 10 \), \( p_3 = \left( 1 - (1 - \left( \frac{1}{1024} \right)^{1024} \right) = \left( 1 - \left( \frac{1023}{1024} \right)^{1024} \right) \approx 1 - \frac{1}{e} \approx 0.63
\]
For \( m = 9 \), \( p_3 = \left( 1 - (1 - \left( \frac{1}{512} \right)^{1024} \right) = \left( 1 - \left( \frac{511}{512} \right)^{1024} \right) \approx 0.86
\]
For \( m = 8 \), \( p_3 = \left( 1 - (1 - \left( \frac{1}{256} \right)^{1024} \right) = \left( 1 - \left( \frac{255}{256} \right)^{1024} \right) \approx 0.98
\]
For \( m = 7 \), \( p_3 = \left( 1 - (1 - \left( \frac{1}{128} \right)^{1024} \right) = \left( 1 - \left( \frac{127}{128} \right)^{1024} \right) \approx 0.99
\]

Let \( x_1, x_2, \ldots, x_k \sim n(0,1) \) and \( y_1 > y_2 > \cdots > y_k \) be an ascending order of the same \( k \) random variables, then
\[
h(y_1, y_2, \ldots, y_k) = k! f(y_1) f(y_2) \cdots f(y_k).
\]

Simulation was performed with 4000 random test vectors against a codebook. The occurrence of \( m \) values greater than 1 in absolute value is shown in Figure 4.7. These test vectors were used to check the codebook for sign orientation matches. Figure 4.8 indicates the success of searching for sign matches given the type of occurrence shown in Figure 4.7. Figure 4.9 is the percentage of success in finding a sign matched codebook vector with the test vector. The success is recorded when a codebook vector has the same sign in the positions which match the positions in the test vector where a value greater than 1 in absolute value occurs.

From the results it appears that there is a near perfect chance of matching sign orientation for test vector with 8 values greater than 1 in absolute value. At 9 the probability starts to fall until the matching totally fails at 17 positions. This agrees with the results of the computations.

Characterization

Speed efficiency in a heuristic method must involve reducing the number of codebook vectors that are to be checked for nearness. A pre-processing step may be employed to prepare the codebook for later on-line processing of the test vectors. This
Figure 4.7: Histogram of occurrences $|x|>1$

pre-processing computation does not affect the efficiency of the use of the algorithm. Pre-processing is done once and the results used throughout the test vector processing with no added overhead. The goal in the pre-processing is to prepare the codebook so that fast access and subset candidate selection is possible. The goal of subset selection is to choose as few candidates as possible while including the nearest neighbor from the entire codebook. The ideal is a single selection that is correct. Due to the dimensionality involved, this ideal is elusive. An attractive solution is to use sign-matching solely to determine good candidates for final searching. However, it was determined that the nearest vector with the given Euclidean metric has a $\frac{1}{2}$ chance of sign match-
Figure 4.9: Histogram of probability of finding k amount of $|x| > 1$

ing in any given location. Thus by rejecting all codebook vectors as candidates for searching based on the signs in the location, the chances of successfully retaining the closest vector is cut in half.

Solution

The problem of determining nearest neighbor in 40 dimensions has a heuristic solution by using the nearest element in each dimension philosophy. A "vote" philos-
ophy was studied where the largest magnitude elements are used to indicate which codebook vectors would be preferable to search. The codebook is then searched in either a full width mode or short circuit mode. In full width, all the vector elements are used to determine nearness. In the short circuit method, only the elements in the positions of the voting elements are used.

A "reject" philosophy influenced the solution, and thus the algorithm presented here is to start with the total codebook and eliminate vectors to get the candidate set. A set intersection method is employed. Each dimension is searched for a subset of nearest elements. These dimension subsets are then intersected to produce a candidate subset for further processing. The following solution explanation includes example values however these are for illustration only. The target final subset size was set at 10% of the codebook. This was used to determine the amount of reduction in each dimension. Since there are 40 dimensions each considered independent of each other, the scheme used was to retain a certain percent in each dimension so that the intersection produced the target size set. Due to the independence this gives us:

\[(x)^{40} = 10\%\].

This solves to approximately 95% of each dimension being retained or equivalently, 5% being rejected. To use this method, a fast search in each dimension is necessary. The nearest 95% in each dimension must be determined. This can be done quickly in a sorted list with a binary search.

The codebook is pre-processed so that each dimension is sorted. This is done with a parallel table of indices of the vectors. The vectors themselves remain unchanged during the sorting. The indices are moved in the parallel table to indicate the correct sort arrangement. Thus a positional search is an indirect lookup and comparison. The intersection is determined with a flag array where all elements are set to on, indicating inclusion, and at each step of the positional search the necessary elements are set to off, indicating rejection. The resulting condition of the flag array indicates
which codebook vectors have "survived" and are candidates for exhaustive search.

An enhancement to this method is to weight the test vector elements according to their magnitude. This is done by a similar sorting method. A parallel array of indices is used and the test vector sorted through the indices. The rejection process can then be done with varying strength in individual locations of the dimension space. Large magnitude values are given more strength in rejecting candidates from the codebook. For example, if the value is relatively large and positive, then the relatively large negative values in that position of the codebook vectors are not good candidates and those vectors can be rejected. As the values become relatively small they may be close to vector values on either side of zero and thus they should only reject the largest magnitude values. Large magnitude test vector values will carry large errors if not matched well. Small magnitude values will not cause as large an error and thus it makes sense to allow the large magnitude values to perform stronger rejection than the small magnitude values. A variable system where the largest magnitude values reject 15-20% of the candidates and the smallest reject 2-4% is suggested and empirical tests show good results.

A further enhancement is suggested for speed concerns. Based on the large error cost of large magnitude values, the method can be short circuited to only search on the largest magnitude elements. Empirical tests show that the results are comparable with some loss for the quicker method.

Analysis

The original exhaustive method was noted to be \( O(kN) \). A strict "by the book" analysis will show the same cost since a fraction of the original method remains. Thus an operations analysis must be used. The original method for the example case would use 40 subtractions, 40 multiplications and 39 additions a total of 1024 times. The overhead of 1023 comparisons to determine the minimum will be ignored as well.
as the number of cycles necessary to perform the mathematical operations. It will suffice to note that they are more than simple. Thus a base of 121856 operations is determined. The solution contains a pre-processing phase whose cost is discounted. The online phase includes 40 binary searches of cost 10 (\(\log_2 1024 = 10\)). The flag array maintenance is allowed 102 operations (\(~90\%\) of 1024) for each of the 40 dimensions. The vector sort is allowed 6 operations. The final search will be allowed 20% of the original. Thus the solution has approximately 28857 operations or about 4 fold improvement. The allowances are biased towards worst case and best performance. The dominant factor is the fraction of exhaustive search remaining. The remaining exhaustive search time is lessened through the short circuit method outlined above.

Results

The implementation of the solution has provided respectable results. Accuracy results improve with increased candidate retention as expected. Approximate values are 80% accuracy with 10% retention, 85% accuracy with 15% retention, 90% accuracy with 20% retention, 95% accuracy with 25% retention and 99-100% accuracy with 50% retention. The element weighting was empirically optimized for 80% and some improvements are expected for other accuracy levels. In order to quantify these successes two different algorithms were used to emulate compression on images. The MSE was computed for the resulting images. The standard “Lenna” image was used. In order to test vector length versus image detail, the standard image was used to create two smaller versions. The original image is 512 by 512 in size. The smaller versions created were 256 by 256 and 128 by 128. The RGB versions were used, however they were partially transformed to YIQ and the Y band was used for the testing. The images were processed and the codebooks searched in the various means. A brute force search simply determines the closest vector in the codebook exhaustively. A search based on vector elements greater than 1 in magnitude was implemented in two
different manners. One does a full vector length distance computation when searching the candidates and the other is a short circuit which only does distance calculations on the elements where the magnitude is greater than 1. The last method is a rejection method which votes against the vectors whose elements are farthest away in the position under consideration.

A quad interpolation vector quantization (QIVQ) compression scheme where the image is processed in block manner was used. The image is processed in 16 by 16 pixel blocks. The corners of the block are used to interpolate the remaining block. The difference between the approximate block and the actual block is determined and the range checked. If the range is small enough, under 40 in testing, the errors of the block are vector quantized as 16 element vectors. If the range is outside that bound, the block is subdivided into four 8 by 8 blocks. Again the interpolation is performed and errors calculated and checked. If the range is acceptable, the errors are vector quantized by the same 16 element codebook. If the range is still too large, the block is again subdivided and each 4 by 4 sub-block interpolated and checked. Passing block errors are vector quantized and failing blocks are marked for raw storage or transmission. The compression results will vary. If a standard codebook known on both ends is used, only the corner elements and three bitmaps are necessary along with any failing 16 element sub-sub-blocks and the vector encoding array. The results are tabulated in Table 4.1 where MSEfull is the mean square error from the full length search and MSEshort is from the short circuit method.

<table>
<thead>
<tr>
<th>Image</th>
<th>128X128</th>
<th>256X256</th>
<th>512X512</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.471280</td>
<td>0.240010</td>
<td>0.123195</td>
</tr>
<tr>
<td>Brute</td>
<td>0.489016</td>
<td>0.248186</td>
<td>0.126517</td>
</tr>
<tr>
<td>Full</td>
<td>0.498555</td>
<td>0.252284</td>
<td>0.127425</td>
</tr>
<tr>
<td>Short</td>
<td>0.505030</td>
<td>0.254566</td>
<td>0.127864</td>
</tr>
<tr>
<td>Reject</td>
<td>0.489016</td>
<td>0.248160</td>
<td>0.126509</td>
</tr>
</tbody>
</table>

Table 4.1: Quad Interpolation Error Vector Quantization MSE Results
Table 4.2: Wavelet Tramfrom Error Vector Quatization MSE Results

<table>
<thead>
<tr>
<th>Image</th>
<th>512X512</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.026980</td>
</tr>
<tr>
<td>Brute</td>
<td>0.022015</td>
</tr>
<tr>
<td>Full</td>
<td>0.022318</td>
</tr>
<tr>
<td>Short</td>
<td>0.022857</td>
</tr>
<tr>
<td>Reject</td>
<td>0.022064</td>
</tr>
</tbody>
</table>

images are very comparable. Thus the short circuit method is preferred where large
data must be processed efficiently. A second test was performed on images resulting
from wavelet compression. A simple wavelet compression method was used with three
levels of transformation taking place and only the HH band of the last transform being
retained. This results in over 60 to 1 compression, and possibly higher depending on
the quantization phase. For this work a simple test result image was desired so the
quantization phase was not performed. The Y band of the YIQ transform of the
original 512 by 512 image was used for testing. A 16 tap Daubechies filter was used
for the transform. The results are tabulated and shown in Table 4.2.

Note that the result MSE for the images without any VQ is the highest. The
errors in the form of Mach bands and blocking artifacts are well outside the bounds
of acceptable quality. All the VQ enhanced methods improve the image quality with
the best visual performance on the wavelet method with the rejection algorithm.
The original result from the wavelet compression is shown in Figure 4.10. The brute
force search VQ application result is shown in Figure 4.11. The full search and short
circuit search based on magnitude above one matching is shown in Figures 4.12 and
4.13 respectively. The result of the rejection method is shown in Figure 4.14.

Summary

Vector quantization is a widely accepted and practiced method for obtaining
data reduction in digital images and audio. A synopsis of the methods used was
Figure 4.10: Wavelet compression results of "Lenna"

Figure 4.11: Brute force search VQ results
Figure 4.12: Full search magnitude over 1 VQ results

Figure 4.13: Short circuit search magnitude over 1 VQ results
Figure 4.14: Rejection method VQ results

presented as a background for general development of a new analysis of the method. The analysis of the vector codebook and test data interaction provides a new basis for the field at large. The interaction performance of the data is mathematically proven and empirical testing has verified that the results are valid and applicable. The understanding of the performance has provided for the design of new methods of codebook searching with improved search time. The application of random codebook vectors to real world images has shown feasibility of general purpose use of the random vectors thus providing for a general purpose data reduction scheme.
CHAPTER 5

CONCLUSION

This work was performed to investigate and advance the knowledge of data reduction in the digital audio and video domain. The reduction of necessary data used to display or present the images or sounds is a desired process to alleviate the demands that such data places on systems. In order to reduce the data it is necessary to characterize the data into necessary and unnecessary data. The unnecessary data or redundant information may be easily removed or recoded to reduce the data load. The necessary data may also be recoded to reduce the load.

The removal of grossly redundant data was outlined in the form of a motion detection method for video and a feature detection method for still images. Audio reduction is facilitated in a manner similar to the feature detection method. The characterization of data is enhanced by the clear understanding of the nature of the data. An understanding of the imagery and the light allows for successful development of application methods for reducing the data.

The most successful data reduction method and the one with the highest potential for data reduction is vector quantization. The analysis of vector quantization codebook and data matching, facilitates codebook design and use. Increasing vector length is necessary to increase data reduction. The increase in length complicates the use of vector quantization by increasing the search time necessary to match the data. The understanding of the codebook and test vector properties allows for intelligent codebook design and use.
Accomplishments

This work has successfully applied existing lighting model theory in the form of radiosity to understanding the lighting of images from the real world. This understanding led to the development of an efficient and accurate motion detection method where unnecessary data is easily identified. The implementation of the method represents a new algorithm for motion detection as well as data reduction. The method was also successfully extended to the still image domain in the form of a feature identification method. The data reduction method was outlined and the implementation of the method represents a new algorithm for feature identification.

The identification and characterization of light in an environment will help to better understand the problem of video and still image coding. Current high demands for high data reduction and compression in the video domain will be served by the findings and results of this work. The near future will include a high use of small digital cameras in increasing numbers of applications. The processing of the data for storage and transmission will be enabled by the work contained here.

The analysis of vector quantization codebook and data matching represents a new theory and foundation for the design and implementation of vector quantization. The results of the analysis provide insightful metrics into the performance of vector quantization codebooks. This is a new theoretical advancement in vector quantization methods. The findings of the research provided for new algorithm development which was shown to improve the performance of vector quantization. The fast codebook searching methods outlined represent new algorithms for vector codebook search.

Every type of digital audio and image data stored or transmitted will be processed by some sort of data reduction or compression scheme. With lossless compression it makes no sense to store or transmit raw data. Only research purposes may deal with data in a raw format simply to reduce overhead from extensive, repetitive processing. As more data is created digitally and stored or transmitted, the demand for higher
performance will increase. These demands will be served by higher dimensional vector quantization. The theory and application here will facilitate the advances necessary to meet the demands.

Future Research

The application of multi-resolution processing in the video algorithm is an obvious developmental step in the motion detection method. Linking the video and still images methods outlined into a functioning digital video codec is a practical exercise. A real time high data reduction method is currently desired by security oriented industry. Future work will include combining the motion detection and feature identification into a system for live video use.

The theoretical properties for the vector analysis will be used to determine the proper connection between the codebook length $N$ and the vector width $k$. The results indicate there exist distributions and expected values which will determine the best balance between $N$ and $k$.

The theoretical results will also be used to research adaptive length codebooks. The length $N$ may be tied to the actual use of the codebook.

The design of codebooks with compression method consideration will also be researched. The results of this dissertation coupled with knowledge of the target compression method will allow optimal codebook design to be performed.

The application of the fast searching algorithms to data dependent codebooks is the next phase of the algorithm development. Using the theory developed to design and create better codebooks is the appropriate action with the results. The use of random vectors as general purpose error quantization vectors is an obvious choice for handling any and all data in a codec. Reduction and compression methods designed to produce or at least account for this type of error are recommended.
APPENDIX

CODE LISTINGS

Video Code

This section contains code segments from the actual research implementation. The research implementation was performed on a personal computer running the Windows 95 operating system. The intention here is to provide the core routines performing the new algorithms described in the text. These are not complete program listings. These are not production quality codings. It is intended that they provide enough information so that a competent programmer or researcher may use or build on the algorithms.

This section contains a program global file listing as well as the local file globals listing. The program works by allocating a memory buffer into which the graphical drawing is performed and then the buffer is flashed to the screen. The dynamic memory allocation and release is contained at a global level so the routines containing that code is included so the action of the core routine may be understood. Some of the Microsoft Foundation Class (MFC) routines are used to access BMP and AVI files. Some of the MFC routines that handle screen painting and cursor control are also included. The Windows system code typically contains mixed case lettering. The developed code is all lower case with the exception of defined constants or variables which are treated as constant once set. The images used for the research were typically 640 by 480 and so some magic numbers appear in the code.
The focus of any research into this code should be on the actual data manipulation itself. The image data is accessed in a raw format. BMP and AVI data is 8 bit (unsigned char) in three interlaced bands (BGR). The research was conducted uncompressed and thus once the actual position of the data is determined, processing is performed in a linear addressing manner.

The OnInitDialog is a Windows procedure for initializing a dialog based window. It contains global allocation code for the program. The OnOK procedure contains the memory clean up calls.

The video motion detection is found in the OnSubblock procedure. The listing shows code set to blank areas that are determined not to contain motion. The code processes 158 frames of an AVI file and writes the painted frames into BMP files.

The still image edge or feature detection is found in the OnStill procedure. The procedure is set up to work with RGB, YIQ and YUV formats. The YIQ format contains the coding to blank the blocks determined to not contain an edge.
// globals.h, global to entire program
// These are mainly used to control screen drawing
#define SIZE 256

extern COLORREF the_color; // 24 bit value representing RGB value
extern int XSCR[641]; // x dimension mapping array for drawing
extern int YSCR[481]; // y dimension mapping array for drawing
extern int MSX; // global screen x width
extern int MSY; // global screen y height
extern int COLORCT; // global count of colors
extern int count;
extern int bytesize;
extern int INDEX;
extern int *meplay; // global memory pointer for drawing
extern CClientDC* dcplay; // global screen pointer for drawing
extern CBitmap* bits; // global memory for drawing

// file globals, global only to the containing file

PAVISTREAM ppavi;
unsigned char *buff1;
unsigned char *buff2;
int *rx[300];
int *gx[300];
int *bx[300];
int flags[300];
float *error;
int *counts;
int erplace;
int lastx;
int lasty;
CDib m_dibFile3;
int CCX;
int CCY;
BOOL avi::OnInitDialog()
{
    int j;
    int startx,starty;
    int wndxsz,wndysz;
    COLORREF back;
    
    CDialo g::OnInitDialog();
    lastx = 0;
    lasty = 0;
    CCX = 640;
    CCY = 480;
    for(j=0;j<300;j++){
        rx[j] = new int[1000];
        gx[j] = new int[1000];
        bx[j] = new int[1000];
    }
    error = new float[921600];
    counts = new int[408960];
    startx = 0;
    starty = 0;
    wndxsz = GetSystemMetrics(SM_CXSCREEN);
    wndysz = GetSystemMetrics(SM_CYSCREEN);
    SetWindowPos(&wndTop,startx,starty,wndxsz,wndysz,SWP_SHOWWINDOW);
    bits = new CBitmap;
    meplay = new CDC;
    back = BLACK;
    CBrush backbrush(back);
    if(meplay->GetSafeHdc() == NULL) {
        dcplay = new CClientDC(this);
        wndxsz = GetSystemMetrics(SM_CXSCREEN);
        wndysz = GetSystemMetrics(SM_CYSCREEN);
        MSX = wndxsz;
        MSY = wndysz;
        meplay->CreateCompatibleDC(dcplay);
        bits->CreateCompatibleBitmap(dcplay,MSX+10,MSY+10);
        meplay->SelectObject(bits);
        CBrush* oldbrush = meplay->SelectObject(&backbrush);
        meplay->PatBlt(0,0,MSX,MSY,PATCOPY);
        meplay->SelectObject(oldbrush);
    }
    buffi = NULL;
    buff2 = NULL;
    AVIFileInit();
    CScrollBar* pSB = (CScrollBar*)GetDlgItem(IDC_SCRPIC);
    pSB->SetScrollRange(nmin,nmax);
    pSB = (CScrollBar*)GetDlgItem(IDC_RANGE);
    pSB->SetScrollRange(fmin,fmax);
    return TRUE;
}
void avl::OnOK()
{
    int i;
    for(i=0;i<300;i++){
        delete [] rx[i];
        delete [] gx[i];
        delete [] bx[i];
        delete [] error;
        delete [] counts;
        delete meplay;
        delete dcplay;
        delete bits;
        m_bStart = FALSE;
        delete [] buff1;
        delete [] buff2;
        CDialoK::OnOK()
    }
}
void avi::OnSubblock()
{
    unsigned char *bytes;
    char filename[100];
    int i,j,k;
    int xdim,ydim;
    int xblock,yblock;
    int xstart,xstop,ystart,ystop;
    int istop,jstop;
    int blockcount;
    int next;
    long k;
    int rx[512],gx[512],bx[512];
    CDib *ndib;
    CSize ssize;
    CFile file;
    float mr,mg,mb;
    long size;
    long count;
    unsigned char *tmp;
    ::SetCursor(::LoadCursor(NULL,IDC_WAIT));
    ssize.cx = 640;
    ssize.cy = 480;
    ndib = new CDib(ssize,24);
    ndib->m_lpImage = new unsigned char[3 * 640 * 480];
    bytes = ndib->m_lpImage;
    UpdateData(TRUE);
    xdim = 640 * 3;
    ydim = 480;
    xblock = 32;
    yblock = 32 * 3;
    xstop = ydim - xblock + 1;
    ystop = xdim - yblock + 1;
    for(j=0; j<512; j++){
        rx[j] = 0;
        gx[j] = 0;
        bx[j] = 0;
    }
    for(j=0; j<300; j++){
        flags[j] = 0;
    }
    if (! m_bStart){
        if(!buff1){
            buff1 = new unsigned char[921600];
            buff2 = new unsigned char[921600];
        }
        AVIStreamOpenFromFile(ppavi,"D:\images\both2. avi",
            streamtypeVIDEO,0,OF_READ,NULL);
        AVIStreamRead(ppavi,1,1,buff1,921600,&size,&count);
        next = 0;
        for(l=1; l<159; l++){
            blockcount = 0;
            AVIStreamRead(ppavi,2+next,1,buff2,921600,&size,&count);
            for(xstart=0;xstart<xstop;xstart+=xblock){
                for(ystart=0;ystart<ystop;ystart+=yblock){
                    istop = xstart + xblock;
                    jstop = ystart + yblock;
                    for(i=xstart; i<istop; i++){
                        for(j=ystart; j<jstop; j+=3){
                            // Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
                        }
                    }
                }
            }
        }
    }
}
```c
rx[buf2[i*xdim+j]] - buff1[i*xdim+j] + 255]]++;
gx[buf2[i*xdim+j+1] - buff1[i*xdim+j+1] + 255]]++;
bx[buf2[i*xdim+j+2] - buff1[i*xdim+j+2] + 255]]++;
}
}
mr = (float) 0.0;
mg = (float) 0.0;
mb = (float) 0.0;
for(j=0; j<512; j++){
    mr += (rx[j] * (j - 255));
    mg += (gx[j] * (j - 255));
    mb += (bx[j] * (j - 255));
}
mr /= 1024;
mg /= 1024;
mb /= 1024;
for(j=0; j<512; j++){
    rx[j] = 0;
    gx[j] = 0;
    bx[j] = 0;
}
if((fabs(mr) > m.thresh) ||
    (fabs(mg) > m.thresh) ||
    (fabs(mb) > m.thresh)){
    flags[blockcount] = 1;
}
blockcount++;
}
if(1){
    blockcount = 0;
    for(xstart=0; xstart<xstop; xstart+=xblock){
        for(ystart=0; ystart<ystop; ystart+=yblock){
            istop = xstart + xblock;
            jstop = ystart + yblock;
            if(!flags[blockcount]){  
                for(i=xstart; i<istop; i++){
                    for(j=ystart; j<jstop; j+=3){
                        buff1[i*xdim+j] = 0;
                        buff1[i*xdim+j+1] = 0;
                        buff1[i*xdim+j+2] = 0;
                    }
                }
            }
            blockcount++;
        }
    }
    blockcount++;
}
k = 0;
for(i=0; i<480; i++){  
    for(j=0; j<640; j++){
        gputpixel(j, 480-i, RGB(buff1[k+2], buff1[k+1], buff1[k]));
        bytes[k+2] = buff1[k+2];
        bytes[k+1] = buff1[k+1];
        bytes[k] = buff1[k];
        k += 3;
    }
}
setcolor(WHITE);
for(i=0; i<=480; i+=32){
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
    gline(0,i,640,i);
}
for(j=0;j<=640;j+=32){
    gline(j,0,j,480);
}
dcplay->BitBlt(0,0,641,481,meplay,0,0,SRCCOPY);
sprintf(filename,"D:\\image8\\short%d.bmp",1);
file.Close();
}
for(i=0;i<300;i++){
    flags[i] = 0;
}
tmp = buff1;
buff1 = buff2;
buff2 = tmp;
next++;
}
 AVIStreamRelease(ppavi);
AVIFileExit();
delete ndib->m_lpImage;
::SetCursor(::LoadCursor(NULL,IDC_ARROW));
}
void avi::OnStill()
{
    int i,j,k,l,m,n;
    int off;
    int xdim,ydim;
    int type;
    int xblock,yblock;
    int xstart,xstop,ystart,ystop;
    int istop,jstop;
    int index;
    char filename[20];
    char output[40];
    CFile file;
    unsigned char *bytes;
    float *ycl;
    float mr,mg,mb;
    float diffs[1024];
    float range[1024];
    float min,max;
    COLORREF back;
    CSize sizeFileDib;

    UpdateData(TRUE);
    back = BLACK;
    CBrush backbrush(back);
    CBrush* oldbrush = meplay->SelectObject(&backbrush);
    meplay->PatBlt(0,0,MSX,MSY,PATCOPY);
    meplay->SelectObject(oldbrush);
    for(i=0;i<300;i++){
        for(j=0;j<1000;j++){
            rx[i][j] = 0;
            gx[i][j] = 0;
            bx[i][j] = 0;
        }
    }
    for(i=0;i<1024;i++){
        range[i] = (float) 0.0;
    }
    index = 0;
    if(m_type == "RGB"){
        type = 0;
    }
    else{
        if(m_type == "YIQ"){
            type = 1;
        }else{
            type = 2;
        }
    }
    CFileDialog FDialog(TRUE,"bmp","*.bmp",
        OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT|OFN_NOCHANGEDIR|OFN_EXPLORER,
        "Supported Files (*.bmp) (*.bmp) | ");
    if(FDialog.DoModal() != IDOK){
        return;
    }
    CString FileName = FDialog.GetPathName();
    file.Open(FileName, CFile::modeRead);
if(m_dibFile3.Read(&file) == FALSE){
    sprintf(output, "Filename is %s", filename);
    AfxMessageBox(output);
}
else {
    ::SetCursor(::LoadCursor(NULL, IDC_WAIT));
    sizeFileDib = m_dibFile3.GetDimensions();
    m_dibFile3.Draw(meplay, CPoint(0, 0), sizeFileDib);
    bytes = (unsigned char *) m_dibFile3.m_lpImage;
    setcolor(WHITE);
    CCX = sizeFileDib.cx;
    CCY = sizeFileDib.cy;
    for(i=0;i<=CCY;i+=32){
        gline(0,i,CCX,i);
    }
    for(j=0;j<=CCX;j+=32){
        gline(j,0,j,CCY);
    }
    dcplay->BitBlt(0,0,640,512,meplay,0,0,SRCCOPY);
    ycl = new float[921600];
    if(type){
        off = 0;
        for(j=0;j<CCY;j+=) {
            for(k=0;k<CCX;k++) {
                switch(type) {
                    case 1: // yiq
                        ycl[off] = (float) (0.299 * bytes[off+2] +
                                         0.587 * bytes[off+1] +
                                         0.114 * bytes[off]) ;
                        ycl[off+1] = (float) (0.596 * bytes[off+2] +
                                             -0.274 * bytes[off+1] +
                                             -0.322 * bytes[off]) ;
                        ycl[off+2] = (float) (0.211 * bytes[off+2] +
                                             -0.523 * bytes[off+1] +
                                             0.312 * bytes[off]) ;
                        break;
                    case 2: // yuv
                        ycl[off] = (float) (0.299 * bytes[off+2] +
                                            0.587 * bytes[off+1] +
                                            0.114 * bytes[off]) ;
                        ycl[off+1] = (float) (0.701 * bytes[off+2] +
                                         -0.587 * bytes[off+1] +
                                         -0.114 * bytes[off]) ;
                        ycl[off+2] = (float) (-0.299 * bytes[off+2] +
                                             -0.587 * bytes[off+1] +
                                             0.886 * bytes[off]) ;
                        break;
                }
                off += 3;
            }
        }
        xdim = sizeFileDib.cx * 3;
        ydim = sizeFileDib.cy;
        xblock = 32;
        yblock = 32 * 3;
        xstop = ydim - xblock + 1;
        ystop = xdim - yblock + 1;
off = 0;
for(xstart=0;xstart<xstop;xstart+=xblock){
    for(ystart=0;ystart<ystop;ystart+=yblock){
        mr = (float) 0.0;
        mg = (float) 0.0;
        mb = (float) 0.0;
        istop = xstart + xblock;
        jstop = ystart + yblock;
        for(i=xstart;i<istop;i++){
            for(j=ystart;j<jstop;j+=3){
                switch(type){
                    case 0: // RGB
                        mb += bytes[i*xdim+j];
                        mg += bytes[i*xdim+j+1];
                        mr += bytes[i*xdim+j+2];
                        break;
                    case 1: // yiq
                    case 2: // yuv
                        mb += ycl[i*xdim+j];
                        mg += ycl[i*xdim+j+1];
                        mr += ycl[i*xdim+j+2];
                        break;
                }
            }
            mr /= (float) 1024.0;
            mg /= (float) 1024.0;
            mb /= (float) 1024.0;
            l = 0;
            for(i=X8tart;i<istop;i++){
                for(j=ystart;j<jstop;j+=3){
                    switch(type){
                        case 0: // rgb
                            rx[index][bytes[i*xdim+j] + 255]++;
                            gx[index][bytes[i*xdim+j+1] + 255]++;
                            bx[index][bytes[i*xdim+j+2] + 255]++;
                            error[off+2] = (float) (bytes[i*xdim+j] - mb);
                            error[off+1] = (float) (bytes[i*xdim+j+1] - mg);
                            error[off] = (float) (bytes[i*xdim+j+2] - mr);
                            break;
                        case 1: // yiq
                            rx[index][(int) (ycl[i*xdim+j]) + 255]++;
                            gx[index][(int) (ycl[i*xdim+j+1]) + 302]++;
                            bx[index][(int) (ycl[i*xdim+j+2]) + 297]++;
                            range[1] = (float) (ycl[i*xdim+j] - mb);
                            l++;
                            break;
                        case 2: // yuv
                            rx[index][(int) (ycl[i*xdim+j]) + 255]++;
                            gx[index][(int) (ycl[i*xdim+j+1]) + 357]++;
                            bx[index][(int) (ycl[i*xdim+j+2]) + 451]++;
                            break;
                    }
                }
                off += 3;
            }
        }
        if(type == 1){
            m = 0;
        }
    }
}
for(l=0;l<32;l++){
    for(n=0;n<31;n++){
        diffs[m] = range[1*32+n] * range[1*32+n+1];
        m++;
    }
}

min = diffs[0];
max = diffs[0];
for(l=1;l<992;l++){
    if(min > diffs[l]){
        min = diffs[l];
    }
    if(max < diffs[l]){
        max = diffs[l];
    }
}

if((max - min) < 512){
    for(i=xstart;i<istop;i++){
        for(j=ystart;j<ystop;j+=3){
            bytes[i*xdim+j] = 0;
            bytes[i*xdim+j+1] = 0;
            bytes[i*xdim+j+2] = 0;
        }
        index++;
    }
    delete [] yc1;
}
if(type == 1){
    m_dibFile3.Draw(meplay,CPoint(0,0),sizeFileDib);
    for(i=0;i<=CCY;i+=32){
        gline(0,i,CCX,i);
    }
    for(j=0;j<=CCX;j+=32){
        gline(j,0,j,CCY);
    }
    dcplay->BitBlt(0,0,640,512,meplay,0,0,SRCCOPY);
}
file.Close();
::SetCursor(::LoadCursor(NULL,IDC_ARROW));
Vector code

This section contains the core code segments from the vector quantization research program. Once again, these are not complete program listings. The variables from the normal distribution with mean 0 variance 1 are generated by the bmrard procedure. The sorting is done by the quickmerge procedure.

The OnInitDialog is again a Windows initialization procedure that contain global allocation code. An OnOK procedure has the memory clean up code similar to the above code and is not included. The OnReadFile procedure opens and reads the image into memory.

The code to perform the block processing data reduction through interpolation is contained in the OnVector4 procedure. This code analyzes a block of size 16 by 16 and determines if it is smooth enough to interpolate from the corner data and corrected through vector quantization. If the block is not smooth enough, it is subdivided into smaller blocks until 4 by 4 blocks are reached. The default behavior is to pass the 4 by 4 data through untouched if it cannot be interpolated and error corrected. The supporting procedures are deblock, deblock and interpolate.

The wavelet truncation method is contained in the OnVector5 procedure. This creates a 4 level pyramid wavelet transform and then discards the lower three levels before reconstructing the data. The vector quantization then corrects for the lost data. Supporting routines include the wavelet coefficients as well as the x and y traversal subsampling.
### Global.h

```c
extern COLORREF the_color; // 24 bit value representing RGB value
extern int XSCR[641];       // x dimension mapping array for drawing
extern int YSCR[481];       // y dimension mapping array for drawing
extern int MSX;             // global screen x width
extern int MSY;             // global screen y height
extern int CCX;             // global for image x size
extern int CCY;             // global for image y size
extern int COLORCT;         // global count of colors
extern int count;
extern int INDEX;
extern int bytesize;
extern CDC* memplay;        // global memory pointer for drawing
extern CClientDC* dc;       // global screen pointer for drawing
extern CBitmap* bitmap;     // global drawing memory
extern unsigned char *image;
```

```c
#define VECT 1024
#define LEN 16

double bmrand(void)
{
    double denum;
    double ul, u2;
    double x;
    double pi = 3.1415926535;

    denum = (double) RAND_MAX;
    ul = (double) rand() / denum;
    u2 = (double) rand() / denum;
    while(!(ul))
    {
        ul = (double) rand() / denum;
    }
    x = sqrt(-2.0 * log(ul)) * cos(2.0 * pi * u2);
    return x;
}
```
void quickmerge(int *index, double *sorter, int start, int stop)
{
    int i, j, k, itmp, *iray;
    double dtmp, *dray;
    if((stop - start) > 1){
        quickmerge(index, sorter, start, (start + stop) / 2);
        quickmerge(index, sorter, (start + stop) / 2 + 1, stop);
        i = start;
        j = (start + stop) / 2 + 1;
        k = 0;
        iray = new int[VECT];
        dray = new double[VECT];
        while((i <= ((start + stop) / 2) + 1) && (j <= stop)){
            if(sorter[i] < sorter[j]){
                dray[k] = sorter[i];
                iray[k] = index[i++];
            } else{
                dray[k] = sorter[j];
                iray[k] = index[j++];
            }
            k++;
        }
        if(j > stop){
            while(i < ((start + stop) / 2 + 1)){
                dray[k] = sorter[i];
                iray[k] = index[i++];
                k++;
            }
        } else{
            while(j <= stop){
                dray[k] = sorter[j];
                iray[k] = index[j++];
                k++;
            }
        }
        k = 0;
        for(i = start; i <= stop; i++) {
            sorter[i] = dray[k];
            index[i] = iray[k++];
        }
        delete [] iray;
        delete [] dray;
    } else{
        if(sorter[start] > sorter[stop]) {
            itmp = index[start];
            index[start] = index[stop];
            index[stop] = itmp;
            dtmp = sorter[start];
            sorter[start] = sorter[stop];
            sorter[stop] = dtmp;
        }
    }
}
BOOL CViewerDlg::OnInitDialog()
{
    int startx;
    int starty;
    int wndxsz;
    int wndysz;
    COLORREF back;
    CDialog::OnInitDialog();
    startx = 0;
    starty = 0;
    wndxsz = GetSystemMetrics(SM_CXSCREEN);
    wndysz = GetSystemMetrics(SM_CYSCREEN);
    SetWindowPos(hWndTop, startx, starty, wndxsz, wndysz, SWP_SHOWWINDOW);
    bitmap = new CBitmap;
    memplay = new CDC;
    back = BLACK;
    CBrush backbrush(back);
    if (memplay->GetSafeHdc() == NULL)
    {
        dc = new CClientDC(this);
        wndxsz = GetSystemMetrics(SM_CXSCREEN);
        wndysz = GetSystemMetrics(SM_CYSCREEN);
        MSX = wndxsz;
        MSY = wndysz;
        memplay->CreateCompatibleDC(dc);
        bitmap->CreateCompatibleBitmap(dc, MSX+10, MSY+10);
        memplay->SelectObject(bitmap);
        CBrush* oldbrush = memplay->SelectObject(&backbrush);
        memplay->PatBlt(0, 0, MSX, MSY, PATCOPY);
        memplay->SelectObject(oldbrush);
        computepos();
    }
    SetIcon(m_hIcon, TRUE);
    SetIcon(m_hIcon, FALSE);
    return TRUE;
}
void CVeiwerDlg::OnReadfile()
{
    char *buff;
    unsigned char *bytes;
    int x, y;
    int howmuch;
    LONG xdim, ydim;
    DWORD offset;
    CFile file;
    COLORREF back;
    FILE *fp;

    back = BLACK;
    CBrush backbrush(back);
    CBrush* oldbrush = memplay->SelectObject(&backbrush);
    memplay->PatBlt(0,0,MSX,MSY,PATCOPY);
    memplay->SelectObject(oldbrush);
    dc->BitBlt(0,50,MSX,MSY-50, memplay,0,50,SRCCOPY);
    CFileDialog FDialog(TRUE,"bmp","*.bmp",
                        OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT|
                        OFN_NOCHANGEDIR|OFN_EXPLORER,
                        "Supported Files (*.bmp)\0*.bmp\0")
    CString FileName = FDialog.GetPathNameO;
    if(image){
        delete [] image;
    }
    fp = fopen(FileName,"rb");
    buff = new char[100];
    howmuch = 3 * sizeof(WORD) + sizeof(DWORD);
    fread(buff,howmuch,1,fp);
    fread(&offset,sizeof(DWORD),1,fp);
    fread(buff,sizeof(DWORD),1,fp);
    fread(&xdim,sizeof(LONG),1,fp);
    fread(&ydim,sizeof(LONG),1,fp);
    howmuch = offset-(3*sizeof(WORD)+3*sizeof(DWORD)+2*sizeof(LONG));
    buff = new char[howmuch];
    fread(buff,howmuch,1,fp);
    image = new unsigned char[howmuch];
    fread(image,howmuch,1,fp);
    ::SetCursor(::LoadCursor(NULL,IDC_WAIT));
    CCX = xdim;
    CCY = ydim;
    bytes = image;
    for(y=CCY+50;y>50;y--){
        for(x=0;x<CCX;x++){
            memplay->SetPixel(x,y,RGB(bytes[2],bytes[1],bytes[0]));
            bytes += 3;
        }
    }
    dc->BitBlt(0,50,CCX,CCY+50, memplay,0,50,SRCCOPY);
    fclose(fp);
    delete [] buff;
    ::SetCursor(::LoadCursor(NULL,IDC_ARROW));
}
void interpolate(double *sc, double *vc, int *flag, int xdim, int ydim) {
    int i, k;
    double j;
    double len;
    double min, max, tmp;

    // top row
    len = xdim - 1;
    vc[0] = sc[0];
    vc[xdim - 1] = sc[xdim - 1];
    // bottom row
    vc[(ydim - 1) * xdim] = sc[(ydim - 1) * xdim];
    vc[ydim * xdim - 1] = sc[ydim * xdim - 1];
    // left side
    j = 1.0;
    for (i = xdim; i <= ((ydim - 1) * xdim); i += xdim) {
        vc[i] = vc[0] * (1.0 - j / len) + vc[(ydim - 1) * xdim] * (j / len);
        j += 1.0;
    }
    // right side
    j = 1.0;
    for (i = (2 * xdim - 1); i <= (xdim * ydim - 1); i += xdim) {
        vc[i] = vc[xdim - 1] * (1.0 - j / len) + vc[ydim * xdim - 1] * (j / len);
    }
    // the center region
    for (k = 0; k < ydim; k++) {
        j = 1.0;
        for (i = (k * xdim + 1); i <= ((k + 1) * xdim - 1); i++) {
            vc[i] = vc[k * xdim] * (1.0 - j / len) + vc[(k + 1) * xdim - 1] * (j / len);
            j += 1.0;
        }
    }
    min = 0.0;
    max = 0.0;
    for (i = 0; i < xdim; i++) {
        for (k = 0; k < ydim; k++) {
            tmp = sc[i * xdim + k] - vc[i * xdim + k];
            if (tmp < min) {
                tmp = min;
            }
            if (tmp > max) {
                max = tmp;
            }
        }
    }
    if ((max - min) < 40.0) {
        flag[0] = 0;
    } else {
        flag[0] = 1;
    }
}
void deblock(double *sc, double *yc, int ccx, int ccy, int len)
{
    int i, j, k;
    int xdim, ydim, xblock, yblock;
    int xstart, xstop, ystart, ystop, istop, jstop;

    xdim = ccx;
    ydim = ccy;
    xblock = len;
    yblock = len;
    xstop = ydim - xblock + 1;
    ystop = xdim - yblock + 1;
    k = 0;
    for(xstart=0; xstart<xstop; xstart+=xblock){
        for(ystart=0; ystart<ystop; ystart+=yblock){
            istop = xstart + xblock;
            jstop = ystart + yblock;
            for(i=xstart; i<istop; i++){
                for(j=ystart; j<jstop; j++){
                    sc[k] = yc[i*xdim+j];
                    k++;
                }
            }
        }
    }
}

void deblock(double *sc, double *yc, int ccx, int ccy, int len)
{
    int i, j, k;
    int xdim, ydim, xblock, yblock;
    int xstart, xstop, ystart, ystop, istop, jstop;

    xdim = ccx;
    ydim = ccy;
    xblock = len;
    yblock = len;
    xstop = ydim - xblock + 1;
    ystop = xdim - yblock + 1;
    k = 0;
    for(xstart=0; xstart<xstop; xstart+=xblock){
        for(ystart=0; ystart<ystop; ystart+=yblock){
            istop = xstart + xblock;
            jstop = ystart + yblock;
            for(i=xstart; i<istop; i++){
                for(j=ystart; j<jstop; j++){
                    sc[i*xdim+j] = yc[k];
                    k++;
                }
            }
        }
    }
}
void CVeiwerDlg::OnVector4()
{
    int i, j, k, l;
    int pt16, pt08, pt04;
    int DIM;
    int off;
    int here, pick;
    int left, right;
    int prob[512];
    int flag[VECT];
    int tmpdex[VECT];
    int index[VECT][LEN];
    int *block16, *block08, *block04;
    char filename[20];
    char output[40];
    CFile file;
    CDib dibFile;
    unsigned char *bytes;
    double min;
    double check[LEN], csort[LEN];
    double dist[VECT], sorter[VECT], *base[VECT];
    double *runner;
    double mean, var;
    COLORREF back;

    ::SetCursor(::LoadCursor(NULL, IDC_WAIT));
    UpdateData(TRUE);
    back = BLACK;
    CBrush backbrush(back);
    CBrush* oldbrush = memplay->SelectObject(&backbrush);
    memplay->PatBlt(0, 50, MSX, MSY - 50, PATCOPY);
    memplay->SelectObject(oldbrush);

    // create an index table
    for(i=0; i<LEN; i++) {
        for(j=0; j<VECT; j++) {
            index[i][j] = j;
        }
    }

    // create vector table
    for(i=0; i<VECT; i++) {
        base[i] = new double[LEN];
    }

    srand((unsigned)time(NULL));
    srand(1999);

    // create vector set
    for(i=0; i<VECT; i++) {
        for(j=0; j<LEN; j++) {
            base[i][j] = bmrand();
        }
    }

    // pre-process sort the column values
    for(i=0; i<LEN; i++) {
        for(j=0; j<VECT; j++) {
            sorter[j] = base[j][i];
            tmpdex[j] = j;
        }
    }
    quickmerge(tmpdex, sorter, 0, VECT - 1);
    for(j=0; j<VECT; j++) {
        // code continues...
    }
}
index[j][i] = tmpdex[j];
}

// index[] now has column sorted information
::SetCursor(::LoadCursor(NULL, IDC_ARROW));
CFileDialog FDialog(TRUE, "bmp", ".bmp",
     OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT|
     OFN_NOCANCEL|OFN_EXPLORER,
     "Supported Files (*.bmp);*.bmp") ;
CString FileName = FDialog.GetPathNameO;
file.Open(FileName, CFile::modeRead) ;
if(dibFile.Read(&file) == FALSE) {
    sprintf(output,"Filename is %s",filename);
    AfxMessageBox(output);
}
else{
    ::SetCursor(::LoadCursor(NULL, IDC_WAIT));
    CSize sizeFileDib = dibFile.GetDimensions();
    dibFile.Draw(memplay,CPoint(0,50),sizeFileDib);
    bytes = (unsigned char *) dibFile.m_lplImage;
    setColor(WHITE);
    CCX = sizeFileDib.cx;
    CCY = sizeFileDib.cy;
    dc->BitBlt(0,50,CCX,CCY+50,memplay,0,50,SRCCOPY);
    sc = new double[CCX * CCY];
    yc = new double[CCX * CCY];
    vc = new double[CCX * CCY];
    rc = new double[CCX * CCY];
    ac = new double[CCX * CCY];
    bc = new double[CCX * CCY];
    tmvec = new double[LEN * LEN];
    DIM = (CCX * CCY) / (LEN * LEN);
    block16 = new int[DIM];
    block08 = new int[4 * DIM];
    block04 = new int[16 * DIM];
    i = 0;
    off = 0;
    for(j=0;j<CCY;j++){
        for(k=0;k<CCX;k++){
            // partial yiq transform
            yc[i] = (0.299 * bytes[off+2] +
                      0.587 * bytes[off+1] +
                      0.114 * bytes[off]);
            i++;
            off += 3;
        }
    }
    // yc now has the Y band of the image
    // block processing [16 X 16] blocks, linearize the data
doblock(sc,yc,CCX,CCY,LEN);
    pt16 = 0;
    pt08 = 0;
    pt04 = 0;
    for(i=0;i<CCX*CCY;i+=(LEN*LEN)){
        interpolate(&sc[i],&vc[i],&block16[pt16],LEN,LEN);
        if(block16[pt16]){  
            for(k=0;k<LEN*LEN;k++){
                tmvec[k] = sc[i+k];
            }
        }
    }
doblock(&sc[i], tmvec, LEN, LEN, LEN/2);
for(k=0; k<LEN*LEN; k+= (LEN*LEN/4)) {
  interpolate(&sc[i+k]), &vc[i+k],
  &block08[pt08]), LEN/2, LEN/2;
if(block08[pt08]) {
  for(l=0; l<(LEN*LEN/4); l++){
    tmvec[l] = sc[i+k+l];
  }
  doblock(&sc[i+k], tmvec, LEN/2, LEN/2, LEN/4);
  for(l=0; l<(LEN*LEN/4); l+= (LEN*LEN/16) ) {
    interpolate(&sc[i+k+l]), &vc[i+k+l],
    &block04[pt04]), LEN/4, LEN/4);
    pt04++;
  }
  pt08++;
}
pt16++;
// sc[0] has original, vc[0] has approximate, compute residual
for(i=0; i<CCX*CCY; i++){
  rc[i] = (sc[i] - vc[i]);
}
// compute stats for normalization
for(i=0; i<652; i++){
  prob[i] = 0;
}
for(i=0; i<CCX * CCY; i++){
  prob[(int) rc[i]+255]++;
}
mean = 0;
for(i=0; i<512; i++){
  mean += (i-255) * (double) prob[i] / (CCX * CCY);
}
var = 0;
for(i=0; i<512; i++){
  var += ((i-255) - mean)*((i-255) - mean)*
  (double) prob[i]/(CCX * CCY);
}
var = sqrt(var);
for(i=0; i<CCX*CCY; i++){
  rc[i] -= mean;
  rc[i] /= var;
}
// rc[0] and beyond have the residuals.
i = 0;
runner = rc;
while(((i+2*LEN) < (CCX*CCY))){
  for(j=0; j<LEN; j++){
    check[j] = rc[i+j];
    csort[j] = check[j];
    tmpdex[j] = j;
  }
  // sort the temp test vector
  quickmerge(tmpdex, csort, 0, LEN-1);
  left = 0;
  while(csort[left] <= -1.0){
    left++;
  }
  right = LEN-1;
while(csort[right] >= 1.0){
    right--;  
}
right++;
// determine with test method
for(j=0;j<VECT;j++){  
    flag[j] = 1;
}  
pick = 950;
for(j=0;j<left;j++){  
    for(k=pick;k<VECT;k++){
        flag[index[k][tmpdex[j]]] = 0;
    }
}
for(j=right;j<LEN;j++){
    for(k=0;k<VECT-pick;k++){
        flag[index[k][tmpdex[j]]] = 0;
    }
}
// compute distance to all vectors
for(j=0;j<VECT;j++){  
    dist[j] = (double) RAND_MAX;
    //dist[j] = 0.0;
    if(flag[j]){
        dist[j] = 0.0;
        for(k=0;k<LEN;k++){
            dist[j] += (csort[k]-base[j][tmpdex[k]])*
                       (csort[k]-base[j][tmpdex[k]]);
        }
    }
}
// find closest vector
min = dist[0];
here = 0;
for(j=1;j<VECT;j++){  
    if(dist[j] < min){
        min = dist[j];
        here = j;
    }
}
// replace residual with vector
for(j=0;j<LEN;j++){
    runner[LEN] = base[here][j];
}  
runner += LEN;
i += LEN;
}
// rc[0] and beyond now have the vector replacements.
// de-linearize the data
pt16 = 0;
pt08 = 0;
pt04 = 0;
for(i=0;i<CCX*CCY; i+==(LEN*LEN)){  
    if(block16[pt16]){  
        for(k=0;k<LEN*LEN;k+=(LEN*LEN)){
            if(block08[pt08]){  
                for(l=0;l<(LEN*LEN/4); l+=(LEN/LEN/16)){  
                    if(block04[pt04]){  
                        // code continues here
                    }
                }
            }
        }
    }
}
for(j=0;j<(LEN*LEN/16);j++){
    rc[i+k+l+j] = 0.0;
    vc[i+k+l+j] = sc[i+k+l+j];
}
    pt04++;
}
for(l=0;l<(LEN*LEN/4);l++){
    tmvec[l] = sc[i+k+l];
}
deblock(&sc[i+k],tmvec,LEN/2,LEN/2,LEN/4);
for(l=0;l<(LEN*LEN/4);l++){
    tmvec[l] = vc[i+k+l];
}
deblock(&vc[i+k],tmvec,LEN/2,LEN/2,LEN/4);
for(l=0;l<(LEN*LEN/4);l++){
    tmvec[l] = rc[i+k+l];
}
deblock(&rc[i+k],tmvec,LEN/2,LEN/2,LEN/4);
}
    pt08++;
}
for(k=0;k<LEN*LEN;k++){
    tmvec[k] = sc[i+k];
}
deblock((&sc[i]),tmvec,LEN,LEN,LEN/2);
for(k=0;k<LEN*LEN;k++){
    tmvec[k] = vc[i+k];
}
deblock((&vc[i]),tmvec,LEN,LEN,LEN/2);
for(k=0;k<LEN*LEN;k++){
    tmvec[k] = rc[i+k];
}
deblock((&rc[i]),tmvec,LEN,LEN,LEN/2);
}
    pt16++;
}
deblock(ac,rc,CCX,CCY,LEN);
deblock(bc,vc,CCX,CCY,LEN);
off = 0;
for(i=0;i<CCX * CCY;i++){
    yc[i] = ac[i] * var + mean + bc[i];
    bytes[off+2] = (unsigned char) yc[i];
    bytes[off+1] = (unsigned char) yc[i];
    bytes[off] = (unsigned char) yc[i];
    off += 3;
}
dibFile.Draw(memplay,CPoint(0,50),sizeFileDib);
dc->BitBlt(0,50,CCX,CCY+50,memplay,0,50,SRCCOPY);
mean = 0.0;
for(i=0;i<CCX*CCY;i++){
    mean += (yc[i] - sc[i]) * (yc[i] - sc[i]);
}
mean /= sqrt(mean);
mean /= (double) (CCX*CCY);
delete rc;
delete vc;
delete yc;
delete sc;
delete ac;
delete bc;
delete block16;
delete block08;
delete block04;
delete tmvec;
}
file.Close();
for(i=0;i<VECT;i++){
    delete base[i];
}
::SetCursor(::LoadCursor(NULL,IDC_ARROW));
}
void filterx(double *image, double *result,
              int x, int y, double *hi, double *lo, int wide)
{
    int i;
    int xpos;
    int ypos;
    int stop;
    int respos;
    int filtpos;
    int samppos;
    int halsize;

    halsize = x * y / 2;
    stop = x - wide;
    respos = 0;
    /* clean the result area */
    for(i=0; i<x*y; i++) {
        result[i] = 0.0;
    }
    /* into the image */
    for(ypos=0; ypos<y*x; ypos+=x) {
        for(xpos=ypos; xpos<ypos+stop; xpos+=2) {
            samppos = xpos;
            for(filtpos=0; filtpos<wide; filtpos++) {
                result[respos] += lo[filtpos] * image[samppos];
                result[respos+halsize] += hi[filtpos] * image[samppos];
                samppos += 2;
            }
            respos++;
        }
        for(xpos=ypos+stop+2; xpos<ypos+x; xpos+=2) {
            samppos = xpos;
            for(filtpos=0; filtpos<wide; filtpos++) {
                result[respos] += lo[filtpos] * image[samppos];
                result[respos+halsize] += hi[filtpos] * image[samppos];
                /* loop on rows */
                samppos = (++samppos) % x + ypos;
            }
            respos++;
        }
    }
}
void filtery(double *image, double *result,
    int x, int y, double *hi, double *lo, int wide)
{
    int i;
    int xpos;
    int ypos;
    int stop;
    int step;
    int bott;
    int respos;
    int filtpos;
    int samppos;
    int halfsize;

    halfsize = x * y / 2;
    stop = (y - wide) * x;
    bott = (y - 1) * x;
    step = 2 * x;
    /* clean the result area */
    for(i=0; i<x*y; i++) {
        result[i] = 0.0;
    }
    /* into the image */
    for(ypos=0; ypos<x; ypos++) {
        respos = ypos;
        for(xpos=ypos; xpos<=ypos+stop; xpos+=step) {
            samppos = xpos;
            for(filtpos=0; filtpos<wide; filtpos++) {
                result[respos] += lo[filtpos] * image[samppos];
                result[respos+halfsize] += hi[filtpos] * image[samppos];
            }
        }
        respos+=x;
    }
    for(xpos=ypos+stop+step; xpos<ypos+bott; xpos+=step) {
        samppos = xpos;
        for(filtpos=0; filtpos<wide; filtpos++) {
            result[respos] += lo[filtpos] * image[samppos];
            result[respos+halfsize] += hi[filtpos] * image[samppos];
        }
        /* loop on columns */
        samppos = (samppos + x) % (bott + x);
    }
    respos+=x;
}
}
void synthx(double *image, double *result, 
double *hi, double *lo, int wide, int x, int y)
{
    int i;
    int xpos;
    int ypos;
    int stop;
    int respos;
    int filtpos;
    int samppos;
    int count;
    int halfsize;
    int halfstop;

    halfsize = x * y / 2;
    halfstop = x / 2;
    stop = (x - wide) / 2;
    respos = 0;
    count = 0;
    /* clean the result area */
    for(i=0;i<x*y;i++)
    {
        result[i] = 0.0;
    }
    /* into the image */
    for(ypos=0;ypos<halfsize;ypos+=halfstop)
    {
        for(xpos=ypos;xpos<=ypos+stop;xpos++)
        {
            samppos = xpos;
            respos = count;
            for(filtpos=0;filtpos<wide;filtpos++)
            {
                result[respos] += lo[filtpos] * image[samppos];
                result[respos] += hi[filtpos] * image[samppos+halfsize];
                respos++;
            }
            count += 2;
        }
        for(xpos=ypos+stop+1;xpos<ypos+halfstop;xpos++)
        {
            samppos = xpos;
            respos = count;
            for(filtpos=0;filtpos<wide;filtpos++)
            {
                result[respos] += lo[filtpos] * image[samppos];
                result[respos] += hi[filtpos] * image[samppos+halfsize];
                /* loop on rows */
                respos = (++respos) % x + 2 * ypos;
            }
            count += 2;
        }
    }
}
void synthy(double *image,double *result,
    double *hi,double *lo,int wide,int x,int y)
{
    int i;
    int xpos;
    int ypos;
    int stop;
    int step;
    int bott;
    int respos;
    int filtpos;
    int samppos;
    int count;
    int halfsize;
    int halfstop;

    halfsize = x * y / 2;
    halfstop = x / 2;
    step = 2 * x;
    stop = ((y - wide) * x) / 2;
    bott = (y - 1) * x;
    respos = 0; /* clean the result area */
    for(i=0;i<x*y;i++) {
        result[i] = 0.0;
    } /* into the image */
    for(ypos=0;ypos<x;ypos++) {
        samppos = ypos;
        count = ypos;
        for(xpos=ypos;xpos<=ypos+stop;xpos+=x) {
            respos = count;
            for(filtpos=0;filtpos<wide;filtpos++) {
                result[respos] += lo[filtpos] * image[samppos];
                result[respos] += hi[filtpos] * image[samppos+halfsize];
                respos+=x;
            }
            samppos+=x;
            count += step;
        }
        for(xpos=ypos+stop+x;xpos<ypos+bott/2;xpos+=x) {
            respos = count;
            for(filtpos=0;filtpos<wide;filtpos++) {
                result[respos] += lo[filtpos] * image[samppos];
                result[respos] += hi[filtpos] * image[samppos+halfsize];
                /* loop on columns */
                respos = (respos + x) % (bott + x);
            }
            samppos+=x;
            count += step;
        }
    }
}
void buildlevel(double *image, int x, int y,
                   double *hi, double *lo, int wide)
{
    int totsize, offset;
    double *temp;

totsize = x * y;
offset = totsize / 2;
temp = new double[totsize];
filterx(image, temp, x, y, hi, lo, wide);
x /= 2;
filtery(temp, image, x, y, hi, lo, wide);
filtery(temp + offset, image + offset, x, y, hi, lo, wide);
delete[] temp;
}

void buildpyr(double *image, int x, int y, int level,
               double *hi, double *lo, int wide)
{
    int i, offset;

    offset = x * y / 4;
    for(i=0; i<level; i++) {
        buildlevel(image, x, y, hi, lo, wide);
        offset /= 4;
x /= 2;
y /= 2;
    }
}

void ubuildlevel(double *code, double *result, int x, int y,
                  double *hi, double *lo, int wide)
{
    int twice, offset;

twice = x * 2;
offset = twice * y;
synthe(code, result, hi, lo, wide, x, twice);
synthe(code + offset, result + offset, hi, lo, wide, x, twice);
synthx(result, code, hi, lo, wide, twice, twice);
}

void ubuildpyr(double *code, double *result, int x, int y,
                int level, double *hi, double *lo, int w)
{
    int i, xs, ys;
xsize = x >> level;
ysize = y >> level;
    for(i=0; i<level; i++) {
        ubuildlevel(code, result, xs, ys, hi, lo, w);
xsize += xs;
ysize += ys;
    }
}
// wavelet transform method
void CVeiwerDlg::OnVector5()
{
    int i,j,k;
    int off;
    int ofst;
    int here,pick;
    int left,right;
    int level;
    int filtwidth;
    int flag[VECT];
    int tmpdex[VECT];
    int index[VECT][LEN];
    int prob[512];
    char filename[20];
    char output[40];
    CFile file;
    CDib dibFile;
    unsigned char *bytes;
    double *sc,*yc,*vc,*rc,*ac,*bc;
    double min;
    double che;
    double check[LEN],csort[LEN];
    double dist[VECT],sorter[VECT],*base[VECT];
    double *runner;
    double *lo_filter,*hi_filter;
    COLORREF back;
    double mean,var;
    SetCursor(:=LoadCursor(NULL,IDC_WAIT));
    UpdateData(TRUE);
    back = BLACK;
    CBrush backbrush(back);
    CBrush* oldbrush = memplay->SelectObject(&backbrush);
    memplay->PatBlt(0,50,MSX,MSY-50,PATCOPY);
    memplay->SelectObject(oldbrush);

    // create an index table
    for(i=0;i<LEN;i++){
        for(j=0;j<VECT;j++){
            index[i][j] = j;
        }
    }

    // create vector table
    for(i=0;i<VECT;i++){
        base[i] = new double[LEN];
    }

    srand((unsigned)time(NULL));
    srand(1999);
    // create vector set
    for(i=0;i<VECT;i++){
        for(j=0;j<LEN;j++){
            base[i][j] = bmrnd();
        }
    }

    // pre-process, sort the column values
    for(i=0;i<LEN;i++){
        for(j=0;j<VECT;j++){
            sorter[j] = base[j][i];
            tmpdex[j] = j;
        }
    }
}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
quickmerge(tmpdex, sorter, 0, VECT – 1);
for(j=0;j<VECT;j++){
    index[j][i] = tmpdex[j];
}

// index[] now has column sorted information
::SetCursor(::LoadCursor(NULL, IDC_ARROW));
CFileDialog FDialog(TRUE, "bmp", ".*_.bmp",
    OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT |
    OFN_NOCHANGEDIR | OFN_EXPLORER,
    "Supported Files (*.bmp);;*.bmp")
CString FileName = FDialog.GetPathName();
file.Open(FileName, CFile::modeRead);
if(dibFile.Read(&file) == FALSE){
    sprintf(output,"Filename is %s",filename);
    AfxMessageBox(output);
} else{
    ::SetCursor(::LoadCursor(NULL, IDC_WAIT));
    CSize sizeFileDib = dibFile.GetDimensions();
dibFile.Draw(memplay, CPoint(0,50), sizeFileDib);
    bytes = (unsigned char *) dibFile.m_lplmage;
    setcolor(WHITE);
    CCX = sizeFileDib.cx;
    CCY = sizeFileDib.cy;
dc->BitBlt(0,50,CCX,CCY+50,memplay,0,50,SRCCOPY);
    sc = new double[CCX * CCY];
    yc = new double[CCX * CCY];
    vc = new double[CCX * CCY];
    rc = new double[CCX * CCY];
    ac = new double[CCX * CCY];
    bc = new double[CCX * CCY];
    i = 0;
    off = 0;
    for(j=0;j<CCY;j++){
        for(k=0;k<CCX;k++){
            // partial yiq transform
            yc[i] = (0.299 * bytes[off+2] +
                     0.587 * bytes[off+1] +
                     0.114 * bytes[off]);
            i++;
            off += 3;
        }
    }

    //("Usage pyr <filename> x-size y-size levels filter-width\n";
    //("Where <filename> is the target image\n";
    //(" x-size is the x dimension\n";
    //(" y-size is the y dimension\n";
    //("Sorry square power of two only yet\n";
    //(" levels is number of pyramid\n";
    //(" filter-width is self explanatory\n";
    //("Current filters:\n";
    //("Filter width ENTER number Whose\n";
    //("4.6.8,10,12,14,16,18,20 4.6.8,10,12,14,16,18,20 Daubechies\n";
    //("5.7.9,11,13 5.7.9,11,13 Simoncelli\n";
    //("8A,8B,12A,12B 22,24,28,30 Simoncelli\n";
    //("8J,12J 26,32 Johnston\n")
}
filtwidth = 16;
filterset(&hi_filter,&lo_filter,&filtwidth);
for(i=0;i<CCX*CCY;i++){
    ac[i] = yc[i];
    sc[i] = yc[i];
}
level = 3;
buildpyr(ac,CCX,CCY,level,hi_filter,lo_filter,filtwidth);
// remove lower levels to create errors
ofst = CCX * CCY / 32;
for(i=ofst;i<CCX*CCY;i++){
    ac[i] = 0.0;
}
ubuildpyr(ac,vc,CCX,CCY,level,hi_filter,lo_filter,filtwidth);
for(i=0;i<CCX*CCY;i++){
    che = ac[i] + 0.5;
    if(che < 0.0){
        che = 0.0;
    } else{
        if(che > 255.0){
            che = 255.0;
        }
    }
    vc[i] = che;
} // sc[0] has original, vc[0] has approximate, compute residual
for(i=0;i<CCX*CCY;i++){
    rc[i] = (sc[i] - vc[i]);
} // compute stats for normalization
for(i=0;i<512;i++){
    prob[i] = 0;
}
for(i=0;i<CCX*CCY;i++){
    prob[(int) rc[i]+255]++;
}
mean = 0;
for(i=0;i<512;i++){
    mean += (i-255) * (double) prob[i] / (CCX * CCY);
}
var = 0;
for(i=0;i<512;i++){
    var += (((i-255)-mean)*((i-255)-mean)*
    (double) prob[i]/(CCX * CCY));
}
var = sqrt(var);
for(i=0;i<CCX*CCY;i++){
    rc[i] -= mean;
    rc[i] /= var;
} // rc[0] and beyond have the residuals.
i = 0;
runner = rc;
while((i+2*LEN) < (CCX*CCY)){
    for(j=0;j<LEN;j++){
        check[j] = rc[i+j];
        csort[j] = check[j];
        tmpdex[j] = j;
// sort the temp test vector
quickmerge(tmpdex, csort, 0, LEN-1);
left = 0;
while((csort[left] <= 0.0) && (left < LEN)){
  left++;
}
left--;
right = LEN-1;
while((csort[right] >= 0.0) && (right < LEN)){
  right--;
}
right++;
// determine with test method
for(j=0; j<VECT; j++){
  flag[j] = 1;
}
pick = 950;
for(j=0; j<left; j++){
  for(k=pick; k<VECT; k++){
    flag[index[k][tmpdex[j]]] = 0;
  }
}
for(j=right; j<LEN; j++){
  for(k=0; k<VECT-pick; k++){
    flag[index[k][tmpdex[j]]] = 0;
  }
}
// compute distance to all vectors
for(j=0; j<VECT; j++){
  dist[j] = (double) RAND_MAX;
  if(flag[j]){  
    dist[j] = 0.0;
    for(k=0; k<LEN; k++)
      dist[j] += (csort[k]-base[j][tmpdex[k]])* (csort[k]-base[j][tmpdex[k]]);
  }
}
// find closest vector
min = dist[0];
here = 0;
for(j=1; j<VECT; j++){
  if(dist[j] < min){
    min = dist[j];
    here = j;
  }
}
// replace residual with vector
for(j=0; j<LEN; j++){
  runner[j] = base[here][j];
}
runner += LEN;
i += LEN;
// rc[0] and beyond now have the vector replacements.
off = 0;
for(i=0; i<CCX * CCY; i++){
yc[i] = rc[i] * var + mean + vc[i];
bytes[off+2] = (unsigned char) yc[i];
bytes[off+1] = (unsigned char) yc[i];
bytes[off] = (unsigned char) yc[i];
off += 3;
}
dibFile.Draw(memplay,CPoint(0,50),sizeFileDib);
dc->BitBlt(0,50,CCX,CCY+50,memplay,0,50,SRCCOPY);
mean = 0.0;
for(i=0;i<CCX*CCY;i++){
    mean += (yc[i] - sc Li]) ♦ (yc[i] - sc)[i]);
}
mean = sqrt(mean);
mean /= (double) (CCX*CCY);
delete [] rc;
delete [] vc;
delete [] yc;
delete [] sc;
delete [] ac;
delete [] bc;
}
file.Close();
for(i=0;i<VECT;i++)
delete [] base[i];
}:
::SetCursor(::LoadCursor(NULL,IDC_ARROW));
}
void filterset(double **hi, double **lo, int *n)
{
    int k;
    float sig;
    static double c4[4] = { 0.4829629131445341,
                            0.836516037378079,
                            0.2241436680420134,
                            -0.1294095225512604};
    static double c5[6] = {-0.0761025,
                            0.3535534,
                            0.8593118,
                            0.3535534,
                            0.0761025,
                            0.0};
    static double c6[6] = { 0.3326705529500825,
                            0.8068915093110924,
                            0.4598775021184914,
                            -0.1350110200102546,
                            -0.0854412738820267,
                            0.0352262918857095};
    static double c7[8] = { -0.0074972,
                            -0.0731952,
                            0.3610506,
                            0.8534972,
                            0.3610506,
                            -0.0731952,
                            -0.0074972,
                            0.0};
    static double c8[8] = { 0.2303778133088964,
                            0.7148465705529154,
                            0.6308807679398587,
                            0.0279837694168599,
                            -0.1870348117190931,
                            0.0308413818355607,
                            0.0328630116668852,
                            -0.0105974017850690};
    static double c9[10] = { 0.0282204,
                              -0.0603941,
                              -0.0738819,
                              0.4139475,
                              0.7984298,
                              0.4139475,
                              -0.0738819,
                              -0.0603941,
                              0.0282204,
                              0.0};
    static double c10[10] = { 0.1601023979741929,
                              0.6038292697971895,
                              0.7243085284377726,
                              0.1384281459013203,
                              -0.2422948870663823,
                              -0.0322248695846381,
                              -0.0062414902127983,
                              -0.0125807519990820,
                              0.0033357252854738,
                              0.005612};
    static double c11[12] = { 0.0005612,
                              0.0244078,
                              -0.0558173,
                              -0.0732233,
                              0.4088095,
                              0.8047379,
                              0.4088095,
                              0.4088095,
}
static double cl2[12] = { 0.11540743350, 0.494623890398, 0.751133908021, 0.315250351709, -0.22624693965, -0.12976867567, 0.096501605687, -0.031582039318, 0.000553842201, 0.004777257511, -0.001077301085};

static double cl3[14] = {-0.0145152, 0.0211069, 0.0406707, -0.0990339, -0.0587709, 0.4314804, 0.7723375, 0.4314804, -0.0587709, -0.0990339, 0.0406707, 0.0211069, -0.0145152, 0.0};

static double cl4[14] = { 0.0778520540850037, 0.3975393194813912, 0.7291320908461957, -0.0158291052563823, -0.2840155429615824, 0.004724845739124, 0.1287474266204893, -0.0173693010018090, -0.044082539307971, 0.0139810279174001, 0.0087460940474065, -0.0048703529934520, -0.0003917403733770, -0.0006755494064506, -0.0001174767841248};

static double cl6[16] = { 0.0544158422431072, 0.3128715909143166, 0.6756307362973195, 0.5853546836542159, -0.0158291052563823, -0.2840155429615824, 0.004724845739124, 0.1287474266204893, -0.0173693010018090, -0.044082539307971, 0.0139810279174001, 0.0087460940474065, -0.0048703529934520, -0.0003917403733770, -0.0006755494064506, -0.0001174767841248};

static double cl8[18] = { 0.0380779473638778, 0.2438346746125858, 0.6048212363690095, 0.6573880780512736,
\[0.1331973858249883, -0.2932737832791663, -0.0968407832229492, 0.1485407493381256, 0.0307256814793385, -0.0676328290613279, 0.0002509471148340, 0.0223616621236798, -0.0042815036824635, 0.001476468830563, 0.002303857635232, -0.0002519631889427, 0.0000393473203163];

static double c20[20] = {
    0.026670057901,
    0.188176800078,
    0.527201188932,
    0.68459039454,
    0.281172343661,
    -0.249846424327,
    -0.195946274377,
    -0.127369340336,
    0.093057364604,
    -0.071394147166,
    0.029457536822,
    0.03212674059,
    0.003606553667,
    -0.01073175483,
    0.00395361747,
    0.001972405285,
    -0.00658565695,
    -0.000116466855,
    0.000093588670,
    -0.000013264203};

static double c22[8] = {
    0.0042330,
    -0.0545462,
    0.0545462,
    0.7028738,
    0.7028738,
    0.0545462,
    -0.0545462,
    0.0042330};

static double c24[8] = {
    0.0138932,
    -0.0981376,
    0.0981376,
    0.6932135,
    0.6932135,
    0.0981376,
    -0.0981376,
    0.0138932};

static double c26[8] = {
    0.0132759,
    -0.0999205,
    0.0981901,
    0.6929634,
    0.6929634,
    0.0981901,
    -0.0999205,
    0.0132759};

static double c28[12] = {
    -0.0024175,
    0.0155117,
    0.0019685,
    -0.1117252,
    0.1141427,
    0.6886266,
static double c30[12] = {-0.0056647, 0.0266007, -0.0048733, -0.1185671, 0.1242317, 0.6853794, 0.1242317, -0.1185671, -0.0048733, 0.0266007, -0.0056647};

static double c32[12] = {-0.0053876, 0.0266667, -0.0038329, -0.1197755, 0.1251126, 0.6850152, 0.6850152, 0.1251126, -0.1197755, -0.0038329, 0.0266667, -0.0053876};

static double c35[16] = {-0.0012475221, -0.0024950907, 0.0087309530, 0.0199579580, -0.0505290000, -0.1205509700, 0.2930455800, 0.7061761600, 0.2930455800, -0.1205509700, -0.0505290000, 0.0199579580, 0.0087309530, -0.0024950907, -0.0012475221, 0.0};

static double c37[4] = {-1.0, 2.0, -1.0, 0.0};
static double c39[4] = {1.0, 2.0, 1.0, 0.0};

static double c4r[4], c6r[6], c8r[8], c10r[10], c12r[12], c14r[14], c16r[16], c18r[18], c20r[20];

sig = -1.0;
switch(n) {
  case 4:
    *lo=c4;
    *hi=c4r;
    break;
  case 5:
    *lo=c5;
*hi=c6r;
(*n)++;
break;
case 6:
  *lo=c6;
  *hi=c6r;
  break;
case 7:
  *lo=c7;
  *hi=c8r;
  (*n)++;
  break;
case 8:
  *lo=c8;
  *hi=c8r;
  break;
case 9:
  *lo=c9;
  *hi=c10r;
  (*n)++;
  break;
case 10:
  *lo=c10;
  *hi=c10r;
  break;
case 11:
  *lo=c11;
  *hi=c12r;
  (*n)++;
  break;
case 12:
  *lo=c12;
  *hi=c12r;
  break;
case 13:
  *lo=c13;
  *hi=c14r;
  (*n)++;
  break;
case 14:
  *lo=c14;
  *hi=c14r;
  break;
case 16:
  *lo=c16;
  *hi=c16r;
  break;
case 18:
  *lo=c18;
  *hi=c18r;
  break;
case 20:
  *lo=c20;
  *hi=c20r;
  break;
case 22:
  *lo=c22;
  *hi=c8r;
  *n = 8;
  break;
case 24:
  *lo=c24;
  *hi=c8r;
  *n = 8;
break;
case 26:
  *lo=c26;
  *hi=c8r;
  *n = 8;
  break;
case 28:
  *lo=c28;
  *hi=c8r;
  *n = 8;
  break;
case 30:
  *lo=c30;
  *hi=c12r;
  *n = 12;
  break;
case 32:
  *lo=c32;
  *hi=c12r;
  *n = 12;
  break;
case 35:
  *lo=c35;
  *hi=c16r;
  *n = 16;
  break;
case 37:
  *lo=c37;
  *hi=c4r;
  *n = 4;
  break;
case 39:
  *lo=c39;
  *hi=c4r;
  *n = 4;
  break;
default:
  exit();
break;
}
for (k=0;k<*n;k++){
  *(hi + (((n)-k-1))) = sig * (*(lo + k));
  sig = -sig;
}

 Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
BIBLIOGRAPHY


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.


VITA

Graduate College
University of Nevada, Las Vegas

Blaine Lee Hagstrom

Local Address:
4722 Ashbrook Place
Las Vegas, Nevada 89147

Home Address:
1950 266th Avenue North East
Wilton, North Dakota 58579

Degrees:
Associate of Science, 1991
Bismarck State College

Bachelor of Arts, Mathematics, 1993
University of North Dakota

Bachelor of Science, Computer Science, 1993
University of North Dakota

Master of Science, Computer Science, 1995
University of Nevada, Las Vegas

Special Honors and Awards:
Barrick Fellow, University of Nevada, Las Vegas, 1994-1995
Phi Beta Kappa, National Honor Society.
Upsilon Pi Epsilon, National Honor Society of the Computing Sciences.
Outstanding Student in Mathematics and Computer Science,
Bismarck State College, 1991

Publications:
Symbolic Solution of Linear PBPK Models Using Graph Theory: Risk Assessment

Comparison of Symbolic, Numerical Area Under the Concentration Curves of Small
Linear Physiologically Based Pharmacokinetic Models. ACS Symposium Series 643,
1996 American Chemical Society, Washington D.C.


Dissertation Title: Dimension Reduction of Image and Audio Space

Dissertation Examination Committee:
Chairperson, Dr. Evangelos Yfantis, Ph.D.
Committee Member, Dr. Lawrence Larmore, Ph.D.
Committee Member, Dr. Clinton Jeffery, Ph.D.
Committee Member, Dr. Wolfgang Bein, Ph.D.
Graduate Faculty Representative, Dr. Peter Stubberud, Ph.D.