Adaptive Discrete Cosine Transform Image Compression Applied to Visual Flight Simulators

1986

Nancy A. Burrell

University of Central Florida

Find similar works at: https://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation

https://stars.library.ucf.edu/rtd/4934

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
ADAPTIVE DISCRETE COSINE TRANSFORM
IMAGE COMPRESSION APPLIED
TO VISUAL FLIGHT SIMULATORS

BY
NANCY ANN BURRELL
B.S.E., University of Central Florida, 1982

RESEARCH REPORT
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in the Graduate Studies Program of the
College of Engineering
University of Central Florida
Orlando, Florida

Fall Term
1986
ABSTRACT

A visual flight simulator requires a huge amount of image data to be stored in the database. To make a photo-based system feasible an image compression scheme must be devised to compress the data.

An adaptive discrete cosine transform (DCT) technique is used to compress 24 bit color images to an average of 3 bits per pixel. The bits for the image are distributed based on the relative activity in different parts of the scene. A software implementation of this technique is applied to some sample database images. Results and error analysis are presented.
TABLE OF CONTENTS

NOMENCLATURE ........................................ iv
INTRODUCTION ........................................... 1

Chapter
I. REQUIREMENTS ........................................ 3
II. ADAPTIVE DCT METHOD ............................... 5
III. IMPLEMENTATION ..................................... 7
    Discrete Cosine Transform ........................... 7
    Adaptive Assignment Code ............................ 9
    Standard Deviation Matrices ....................... 12
    Bit Assignment Matrices ............................ 12
    Quantization ....................................... 13
    Decompression .................................... 15
IV. RESULTS ............................................. 18
    Subjective Error Analysis ......................... 18
    Objective Error Analysis .......................... 21
V. CONCLUSIONS ....................................... 22
APPENDIX ................................................. 25
REFERENCES ............................................. 48
<table>
<thead>
<tr>
<th>NOMENCLATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
</tr>
<tr>
<td>BPP</td>
</tr>
<tr>
<td>DCT</td>
</tr>
<tr>
<td>AAC</td>
</tr>
<tr>
<td>BAM</td>
</tr>
<tr>
<td>SDM</td>
</tr>
<tr>
<td>QLUT</td>
</tr>
</tbody>
</table>
INTRODUCTION

A database for a visual flight simulator can be made up of aerial photographs. These color photographs of rural and urban areas are high resolution (1 to 4 foot) and are stored as 24 bit per pixel (BPP) red (R), green (G) and blue (B) color quantization.

Because of the enormous amount of data needed for this database, it must be stored in a compressed form and decompressed as necessary. The decompression should be fairly simple and quick. The data needs to be compressed from 24 to 3 BPP average without degrading the image below acceptable levels. Since the application is for a visual flight simulator, the subjective image degradation is very important both in terms of absolute and relative error. The discrete cosine transform (DCT) technique is generally accepted as yielding a high compression ratio with a fairly low amount of operations required for decompression. An adaptive technique takes into account the variations of activity within a scene. Less bits are used to code areas of relative low activity, such as a desert, than would be used for areas of high activity, such as a city.
An optimal block size is determined, as well as the number of transform coefficients to be retained for acceptable results. Error measurement between the original and reconstructed images, both subjective and objective, is investigated.
CHAPTER I

REQUIREMENTS

The images to be compressed are high resolution color photographs to be used as database for a visual flight simulator. Donovan [1] has defined a requirement to store 50 billion pixels on disk for a high resolution flight simulator database. The data needed for the realtime image is retrieved and stored in memory boards in the simulator hardware. To keep the amount of memory needed to a reasonable amount, it is necessary to compress the data to an average of 3 bits per pixel. Since the 24 BPP color image is actually 8 BPP red, 8 BPP green and 8 BPP blue, each color is processed separately and is compressed to an average of 1 BPP. The decompression technique must require little hardware and minimal processing time.

A frequency domain transform technique, such as the discrete cosine transform, has advantage over a spatial domain one for compression because the transform contains
information about the entire image, in varying degrees, in each coefficient. Therefore, the coefficients having a lesser effect on the image can be eliminated, resulting in a data compression.

Also, any error term is spread throughout the image, perhaps making its effects less important when a coefficient is discarded. The adaptive DCT has a fairly simple decompression algorithm. Habbi [2] states that the cosine transform has been shown to have a better mean square error performance than the Fourier or Hadamard transforms, and is easier to implement than the Karhunen-Loeve.

Error analysis between the original and reconstructed images consist of both objective and subjective measurements. The objective error is calculated using a mean square error method. The subjective analysis will consider absolute and relative errors. Relative errors are color shifts between transform block, representing a change in error between pixels. Absolute error is an incorrect color.
CHAPTER II

ADAPTIVE DCT METHOD

The DCT technique is chosen because it is a fast algorithm to implement and has excellent compression ratios as Chen [3] states. The adaptive DCT breaks the image into transform blocks of $4 \times 4$, $8 \times 8$ or $16 \times 16$ pixels. The smaller transform blocks give greater adaptivity but require more processing.

The adaptivity is in distributing the bits over the image. The transform blocks are compared and assigned an average number of bits based on the activity within the block. These are termed the adaptive assignment codes (AAC).

The database for use with the adaptive DCT can be generated by processing a large group of images and calculating standard deviation matrices (SDM), bit assignment matrices (BAM) and quantization lookup tables (QLUTs). These then become part of the database and any images being compressed can access them. The DCT is
performed and the coefficients are normalized by the corresponding SDM. The corresponding BAM value points to the QLUT table to use and the normalized coefficient is the address to the table.

The output is the quantization code used to represent a particular coefficient. The adaptive assignment code is overhead information carried along with each block to be used for image reconstruction. The AAC is used to access the proper BAM and SDM files. The BAM value points to the proper inverse quantization table (IQLUT). The code stored as the compressed image is the address to the table. The output is the normalized DCT coefficient. This is multiplied by the corresponding SDM and the inverse transform is performed. The output of this is the reconstructed image.
CHAPTER III
IMPLEMENTATION

A block diagram of the adaptive coding scheme is shown in Figure 1. Because of the limited images available, there was no database generated of bit assignment matrices, standard deviation matrices and quantization lookup tables. These are calculated for the image being processed only. A copy of the code used in the implementation is included in the appendix.

Discrete Cosine Transform

The transform matrix $C$ for the discrete cosine transform can be expressed for a $N \times N$ transform block as:

$$ C = \frac{1}{2} \begin{bmatrix} C_{jm} \end{bmatrix}, $$

where

$$ C_{jm} = \begin{cases} 1 & m = 0 \\
2 \cos(2j + 1) \times m \times \pi / 2N, & j = 0, N-1; \ m = 1, N-1.
\end{cases} $$
Figure 1. Adaptive DCT compression.
The two-dimensional transform is then

\[ T(m,n) = C * I(j,k) * C', \]

where

- \( I(j,k) \) = original pixel intensity at \( j,k \)

and

- \( T(m,n) \) = transformed coefficients in position \( m,n \).

The transform converts the image data to a set of coefficients representing the energy distribution. Each coefficient contains information about the whole image. The dc term represents the average image brightness. The other coefficients are increasing frequency terms containing image edge information. The number of coefficients retained affects the resolution of the image.
Adaptive Assignment Code (AAC)

The image is divided into transform blocks of size $N \times N$. The energy content of each transform block is measured by the variance between its pixels. These energies are compared and the AAC is assigned as described in Pacelli [4] by:

$$
AAC_i = \begin{cases} 
\text{TRUNC}[X_i], & 1 < x < 9 \\
1, & x < 1 \\
8, & x > 9, 
\end{cases}
$$

where

- TRUNC = real to integer truncation,
- $i$ = transform block,
- $X_i = \log_2(\text{SIG}_{i}^2/2) - D/N$,
- $N$ = number of pixels per block,
- $\text{SIG}_{i}^2$ = variance of the $i$th transform block,
- $D$ = distortion

and

$$
\sum_{i=1}^{N} AAC_i = N \times (AAC \text{ for desired bit rate}).
$$

From Pacelli [4], the AACs for 1 BPP compression are defined as shown in Table 1.
<table>
<thead>
<tr>
<th>AAC</th>
<th>AVG BPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.375</td>
</tr>
<tr>
<td>2</td>
<td>0.375</td>
</tr>
<tr>
<td>3</td>
<td>0.6875</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
</tr>
<tr>
<td>5</td>
<td>1.3125</td>
</tr>
<tr>
<td>6</td>
<td>1.625</td>
</tr>
<tr>
<td>7</td>
<td>1.9375</td>
</tr>
<tr>
<td>8</td>
<td>2.25</td>
</tr>
</tbody>
</table>
Standard Deviation Matrices (SDM)

The standard deviations are calculated between each coefficient in a transform block and the corresponding coefficients in other blocks assigned the same AAC. These eight resulting matrices are the SDMs. The SDMs are used to determine the bit assignment matrices (BAMs), or number of bits assigned to each coefficient for a particular AAC class. The coefficients of the DCT are normalized by the corresponding standard deviation.

Bit Assignment Matrices (BAM)

The BAMs allocate the bits per transform block between the coefficients in the block. The BAMs are calculated as in Pacelli [4] by:

\[ N_{ij} = \text{Trunc}[(\log_2 \text{SIG}_{ij}) - D], \]

where

- \( N_{ij} \) = (i,j)th element in the BAM,
- \( \text{TRUNC} \) = real to integer truncation function,
- \( \text{SIG}_{ij} \) = Standard deviation truncation from the (i,j)th position of the SDM

and

- \( D \) = distortion term, incremented on successive iterations.

The number of bits assigned to each coefficient in a block cannot exceed the total number of bits allocated to that block by the AAC. Therefore, iterations are done so
that

\[
\sum_{i=1}^{\text{Row}} \sum_{j=1}^{\text{Col}} N_{ij} = N_{\text{tot}},
\]

where

Row = the number of coefficients per row of a block,
Col = the number of coefficients per column of a block,
N_{\text{tot}} = the number of bits corresponding to the AAC,
multiplied by the number of coefficients in the block.

Quantization

The normalized coefficients are grouped according to their corresponding BAM value. All coefficients using the same BAM are grouped together into a bin and normalized to be in the range of 0 to \(2^{**N-1}\), where \(N\) is the number of bits assigned by the BAM. For example, coefficients assigned a BAM of 3 BPP would be normalized to range from 0 to 7. These are the output levels. The output levels can be optimized by using statistical methods to distribute the coefficients throughout the bin. Each level within the bin would then ideally contain coefficients which are close enough in value to be adequately represented by an average.
One such optimization method is Max's algorithm, Max [5]. This is defined by

\[ X_i = \frac{(Y_i + Y_{i-1})}{2} \quad i, = 2, \ldots, N \]

and

\[ \sum_{X_i} (X - Y_i) P(x) dx = 0, \quad i = 1, 2, \ldots, N - 1 \]

where

- \( N \) = the number of quantization levels,
- \( X_i \) = end points of the \( N \) levels,
- \( Y_i \) = output level corresponding to each input range,

and

\[ P(x) = \text{input amplitude probability density as defined by the histogram.} \]

This algorithm is solved by iterative calculations, changing the choice of \( Y_1 \) until a solution is found. These are the output levels to be stored in the QLUTs. All coefficients are processed to create the QLUTs. To then access the correct one, the normalized coefficient is fit into one of the levels and the appropriate QLUT is addressed. This output is the value of the compressed image at that location. When this is done for all of the coefficients in all of the transform blocks, the compressed image is now complete and can be stored with an average of 1 BPP.
Decompression

The decompression requires the database information containing the SDMs, BAMs and the inverse quantization lookup tables (IQLUTs), which are formed when the QLUTs are being addressed. For each element in the compressed image there is an IQLUT value, which corresponds to the coefficient average value assigned to that output level during quantization.

In addition the decompression requires the AAC assignment for each block in the image. The flow of decompression is shown in Figure 2.

The AAC is extracted for each location in the compressed image. From this, BAM can be accessed. The level of the location is the value in the compressed image. Knowing the BAM value and this level, the proper IQLUT table can be accessed. The normalized DCT value is the value in the IQLUT. Inverse normalization can be done, using the SDMs, yielding an average DCT value.
Figure 2. Adaptive DCT decompression.
The DCT values can now be run through an inverse transform:

\[ I'(j,k) = C' \ast T(m,n) \ast C \]

where

- \( I'(j,k) \) = the pixel intensity,
- \( T(m,n) \) = the transformed coefficients in position M,N

and

- \( C \) is defined previously.

The result of the inverse DCT is the reconstructed image.
CHAPTER IV

RESULTS

The resultant images obtained from the 24 to 3 bits per pixel compression and reconstruction are shown in Figure 3. The upper left image is the original image. Upper right is the image with 16 X 16 transform block sizes, retaining 8 X 8 coefficients per block. The lower left image was obtained with a block size of 8 X 8 pixels, 8 X 8 coefficients per block retained, and the lower right used a block size of 4 X 4 and retained 2 X 2 coefficients per block. Better results are obtained when this technique is applied over the entire 512 by 512 image, rather than just 256 by 256 as in Figure 3. Time constraints made it difficult to do that large an image here.
Subjective Error Analysis

The upper right image in Figure 3 has the largest block size and the shortest processing time. The image blocks are noticeable. There are some color shifts between the blocks and some slight incorrect color.

The image with the 8 X 8 block size, which is lower left in Figure 3, is of poor quality due to the relatively few bits per coefficient. With 8 coefficients retained there is no reduction in data over the spatial domain and the bits assigned must be spread over the whole transformed image. The color is very good in this case and the picture streaks could perhaps be filtered out. When the 8 X 8 block size is used with 4 coefficients retained, the blocks are evident due to too few coefficients being retained.

The lower right image of Figure 3 has the smallest block size, but only retains 4 coefficients per block. The color shifts are very obvious and incorrect color is very evident. The reduced number of coefficients causes unacceptable image degradation.
Figure 3. Results obtained from the adaptive DCT.
Objective Error Analysis

The error between the original image and the reconstructed image can be calculated by the mean-square error technique. The error at each pixel in the image is the absolute difference between the pixel intensities measured in red, green and blue. The absolute mean-square error over the image is calculated by:

$$\text{ABS } E = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} (R(i,j) - R'(i,j))^2 + (G(i,j) - G'(i,j))^2 + (B(i,j) - B'(i,j))^2 / N$$

The relative error between pixels is calculated by:

$$\text{REL } E = \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} (\text{ES}(i,j) - \text{ES}(i-1,j))^2 + (\text{ES}(i,j) - \text{ES}(i,j-1))^2 / N$$

where $N$ is the number of pixels.

The results obtained from the compression of the 16 x 16 blocks are a mean-squared error of 507 and a relative error of 304. For the 8 x 8 the mean-squared error is 652 and the relative error is 449. For the 4 x 4 in Figure 3 the mean-squared error is 7474 and the relative error is 1526.
The results obtained indicate that compressing to 3 BPP is too much compression for a high quality visual database. More bits per pixel are necessary for smooth, clear images. When too many coefficients are retained there are fewer bits per coefficient and the results are very blurred edges and streaks through the image. When too few are retained there are prominent color shifts between blocks and incorrect color within blocks. The blocks become very evident. The results obtained in the 16 X 16 with 256 coefficients per block would be sufficient perhaps for background in a simulator system with a highly detailed cut-out where the pilot is directly looking. This area of high resolution would have to be compressed to greater than 3 BPP.

Ideally, a large collection of images would be used when generating the database for this technique. The SDM,
QLUT and BAM assignment criteria would then be based on transform block comparisons between a larger variety of image information per adaptive assignment code. This could lead to a more accurate statistical data pool for the use of future images. New image transform blocks would be assigned an AAC and would then access the proper BAMs, QLUTS and SDMs already available. The process would be much quicker and the results should be better.

The adaptive technique could be applied over a whole database of image sequences. The same technique of comparing relative activity of blocks within an image can be applied to comparing activity between image frames in the database. Images that are relatively inactive may not need the overall average of 3 BPP, while very busy images may fare better with a higher average of bits per pixel. A classification system could be set up for classifying images by comparing the activity of images in the database. A coding scheme similar to the AAC can be used to assign a maximum number of bits per pixel to each image, with the overall database average being 3 BPP.

By redistributing the available bits over the entire database and then redistributing within each image, more accurate results can be obtained without affecting the ultimate goal of minimal storage.

The quantization is an essential part of the adaptivity. The data must be quantized as efficiently as
possible without degrading the results too severely. Due to
time limitations, an optimal quantization scheme was not
achieved, but one such as in Max [5] should be very
effective. The method used uniformly groups values within a
bin into the levels of the bin. Each level ends up with
approximately the same number of entries. Max's method
takes into account the values being grouped together, as
well as the probability of certain ranges of values
occurring for a particular bin. With this knowledge the
distribution of coefficients between the levels of a bin
could be optimized.

There is room for improving the adaptivity. For a
visual flight simulator, desiring high resolution for
tactical missions training the image must be very high
quality. The adaptive DCT method requires large amounts of
computer time and space to implement when setting up the
database. After a database is generated however the process
should be considerably simplified.
APPENDIX

PROGRAM COMPRESS

C THIS PROGRAM USES AN ADAPTIVE DCT METHOD TO COMPRESS AN IMAGE

C IMPLICIT NONE
INCLUDE 'COLOR.INC'

INTEGER I,J,M,K,N,FUNIT,N_COLS,CONSTANT,L
INTEGER R,C,PIXEL,IOS,MINR,MAXR,MINB,MAXB,MING,MAXG
INTEGER IMAGE_SIZE
BYTE BYTE
EQUIVALENCE (BYTE,PIXEL)

C READ THE IMAGE INTO THE RED, GREEN AND BLUE COLOR ARRAYS

C INITIAL FLG=0
DBGFLG=0
PRINT*,"DO YOU WANT DIAGNOSTICS TURNED ON l=YES 0=NO'
READ*,DBGFLG
DISTORT_AAC=1024
DISTORT_BAM=1

C OPEN QUANTIZE ARRAY

OPEN(UNIT=12,NAME='IQLUT.DAT',TYPE='UNKNOWN'
& FORM='FORMATTED')
OPEN(UNIT=14,NAME='CSCENE1.DAT',TYPE='UNKNOWN',
& FORM='UNFORMATTED')
OPEN(UNIT=15,NAME='BAMFILE.DAT',TYPE='UNKNOWN',
& FORM='FORMATTED')
OPEN(UNIT=16,NAME='SMPFILE.DAT',TYPE='UNKNOWN',
& FORM='FORMATTED')
OPEN(UNIT=17,NAME='AACFILE.DAT',TYPE='UNKNOWN',
& FORM='FORMATTED')
PRINT*,'IMAGE SIZE=
READ*,IMAGE_SIZE

C READ IN 512X512 ARRAY

OPEN(UNIT=10,
& NAME='BIGSCSN.DAT',
& TYPE='OLD',FORM='UNFORMATTED',ERR=100)
DO I=0,IMAGE_SIZE-1
READ(10,ERR=101)(RED_IMAGE(I,J),J=0,IMAGE_SIZE-1)
READ(10,ERR=102)(GRE_IMAGE(I,J),J=0,IMAGE_SIZE-1)
READ(10,ERR=103)(BLU_IMAGE(I,J),J=0,IMAGE_SIZE-1)
ENDDO
CLOSE(UNIT=10)
C SET UP VARIABLE PARAMETERS TO BE PASSED IN COMMON
IF(INITIAL_FLAG.EQ.0) THEN
  50 PRINT*, 'BLOCK SIZE = '
  PRINT*, '4X4, 8X8, 16X16, ENTER 4, 8 OR 16'
  READ(*,*,ERR=50) BLK_SIZE
  N_ROWS=BLK_SIZE
  N_COLS=BLK_SIZE
  NUM_BLKS=IMAGE_SIZE/BLK_SIZE
ENDIF
C RED, GREEN, BLUE LOOP
C
DO K=1,3
  IF(K.EQ.1) COLOR=1
  IF(K.EQ.2) COLOR=2
  IF(K.EQ.3) COLOR=3
  DO J=0,IMAGE_SIZE-1
    DO I=0,IMAGE_SIZE-1
      IF(K.EQ.1) IMAGE(I,J)=RED_IMAGE(I,J)
      IF(K.EQ.2) IMAGE(I,J)=GRE_IMAGE(I,J)
      IF(K.EQ.3) IMAGE(I,J)=BLU_IMAGE(I,J)
    ENDDO
  ENDDO
C CALCULATE THE DCT
CALL CALC_DCT
C CALCULATE THE ENERGY OF EACH BLOCK
CALL CALC_ENERGY
C CALCULATE THE ADAPTIVE ASSIGNMENT CODE
CALL CALC_AAC
C CALCULATE THE STANDARD DEVIATION MATRICES
CALL CALC_SDM
C CALCULATE THE BIT ASSIGNMENT MATRICES
CALL CALC_BAM
C NORMALIZE THE COEFFICIENT
CALL NORMALIZE
C CALL QUANTIZE
CALL QUANTIZE
C
DO NEXT COLOR IMAGE END OF K LOOP, RESET INITIAL FLAG
INITIAL_FLAG=1
C WRITE COMPRESSED IMAGE TO FILE
N=NUM_BLKS*N_COEF_ROW
DO I=0,N-1
  WRITE(14)(IMAGE(J,I),J=0,N-1)
ENDDO
C ENDDO END OF K LOOP
C PRINT*, 'OPENED AND WROTE C-IMAGE'
CLOSE(UNIT=FUNIT)
CALL LIB$FREE_LUN(FUNIT)
CLOSE(12)
CLOSE(14)
CLOSE(15)
CLOSE(16)
CLOSE(17)
C STOP
C 100 PRINT*, 'ERROR IN OPENING IMAGE FILE IOSTAT = ', IOSTAT
   STOP
101 PRINT*, 'ERROR READING RED FILE IOSTAT = ', IOSTAT
   STOP
102 PRINT*, 'ERROR READING GREEN FILE IOSTAT = ', IOSTAT
   STOP
103 PRINT*, 'ERROR READING BLUE FILE IOSTAT = ', IOSTAT
C END ! END PROGRAM
C DO NEXT COLOR IMAGE- END OF K LOOP, RESET INITIAL FLAG
C WRITE COMPRESSED IMAGE TO FILE
C INITIAL_FLG=1
C WRITE COMPRESSED IMAGE TO FILE
N=NUM_BLKS*N_COEF_ROW
DO I=0,N-1
   WRITE(14)(IMAGE(J,I),J=0,N-1)
ENDDO
C ENDDO !END OF K LOOP
C
C PRINT*, ' OPENED AND WROTE C-IMAGE '
C CLOSE(UNIT=FUNIT)
C CALL LIB$FREE_LUN(FUNIT)
C CLOSE(12)
C CLOSE(14)
C CLOSE(15)
C CLOSE(16)
C CLOSE(17)
C
C STOP
C 100 PRINT*, ' ERROR IN OPENING IMAGE FILE IOSTAT =',I0S
C STOP
C 101 PRINT*, ' ERROR READING RED FILE IOSTAT =',I0S
C STOP
C 102 PRINT*, ' ERROR READING GREEN FILE IOSTAT =',I0S
C STOP
C 103 PRINT*, ' ERROR READING BLUE FILE IOSTAT =',I0S
C END
! END PROGRAM
SUBROUTINE CALC_ENERGY

C THIS ROUTINE CALCULATES THE ENERGY PER TRANSFORM BLOCK WITHIN
C AN IMAGE MEASURED AS THE VARIANCE BETWEEN PIXELS

IMPLICIT NONE
INCLUDE 'COLOR.INC'

INTEGER C,R,R_BLK_NUM,C_BLK_NUM,X,Y,J,K,I
REAL TEMP_AVG,DEV(16,16),TEMP_SD,AVG,MIN,MAX

CALC ENERGY PER TRANSFORM BLOCK

MIN=99999
MAX=-99999
DO I=1,NUM_BLKS
  DO J=1,NUM_BLKS
    ENERGY(I,J)=0
  ENDDO
ENDDO

DO R_BLK_NUM = 1,NUM_BLKS
  DO C_BLK_NUM = 1,NUM_BLKS
    X = (R_BLK_NUM-1)*BLK_SIZE
    Y = (C_BLK_NUM-1)*BLK_SIZE
    CALC THE AVG
    TEMP_SD=0.
    TEMP_AVG=0.
    DO J = X,X+BLK_SIZE-1
      DO K = Y,Y+BLK_SIZE-1
        TEMP_AVG = FLOAT(IMAGE(K,J)) + TEMP_AVG
      ENDDO
    ENDDO
    AVG = TEMP_AVG/FLOAT(BLK_SIZE*BLK_SIZE)
    FIND EACH DEVIATION
    DO J = X,X+BLK_SIZE-1
      DO K = Y,Y+BLK_SIZE-1
        DEV(K-Y+1,J-X+1) = FLOAT(IMAGE(K,J)) - AVG
      ENDDO
    ENDDO
    TEMP_SD = DEV(K-Y+1,J-X+1)*DEV(K-Y+1,J-X+1) + TEMP_SD
    CALC VAR
    ENERGY(C_BLK_NUM,R_BLK_NUM)=TEMP_SD/(BLK_SIZE*BLK_SIZE)
    IF(ENERGY(C_BLK_NUM,R_BLK_NUM).LT.5) THEN
      ENERGY(C_BLK_NUM,R_BLK_NUM)=5.
    ENDIF
    IF(DBGFLG.EQ.3) THEN
      PRINT*,AVG,TEMP_SD,ENERGY(C_BLK_NUM,R_BLK_NUM)
    ENDIF
  ENDDO
ENDDO

FIND THE AVERAGE OF ALL THE BLOCKS
AVG = 0.0
DO R_BLK_NUM = 1, NUM_BLKS
  DO C_BLK_NUM = 1, NUM_BLKS
    AVG = AVG + ENERGY(C_BLK_NUM, R_BLK_NUM)
  ENDDO
ENDDO
AVG = AVG / FLOAT(NUM_BLKS * NUM_BLKS)

C FIND THE STANDARD DEVIATION OF ALL THE BLOCKS
C
TEMP_SD = 0
DO R_BLK_NUM = 1, NUM_BLKS
  DO C_BLK_NUM = 1, NUM_BLKS
    TEMP_SD = TEMP_SD + (ENERGY(C_BLK_NUM, R_BLK_NUM) - AVG)**2
  ENDDO
ENDDO
TEMP_SD = TEMP_SD / FLOAT(NUM_BLKS * NUM_BLKS)
RETURN
END

******************************************************************************
SUBROUTINE CALC_AAC
IMPLICIT NONE
INCLUDE 'COLOR.INC'
INTEGER R_NUM,C_NUM,CNT,FUNIT,X,Y
REAL I,PREVI,DELTA,DISTORT_NOW
REAL F_BLK_SIZE,AAC_SUM,TEMP,RAAC(l28,128),PAAC_SUM,SAVE

F_BLK_SIZE = FLOAT(BLK_SIZE*BLK_SIZE)
CNT = 0
AAC_SUM=0
PAAC_SUM=0
PREVI=11.0
I=10.0
DELTA=1.0
SAVE=DISTORT_AAC
DISTORT_NOW=DISTORT_AAC

50 CONTINUE
CNT = CNT + 1
AAC_SUM=AAC_SUM / NUM_BLKS**2
IF(CNT.EQ.1) PAAC_SUM=AAC_SUM
IF(CNT.LT.12) THEN
IF(AAC_SUM.GT.4.) THEN
TEMP=ENERGY(C_NUM,R_NUM)/2.0
IF(TEMP.LT.0.0001) THEN
RAAC(C_NUM,R_NUM) = 1
ELSE
RAAC(C_NUM,R_NUM) = (ALOG(TEMP) / ALOG(2.0))**DISTORT_NOW/(F_BLK_SIZE)
ENDIF
IF(DBGFLG.EQ.1)PRINT*,' RAAC ',RAAC(C_NUM,R_NUM)
AAC_SUM = AAC_SUM + RAAC(C_NUM,R_NUM)
ENDIF
ENDIF
GOTO 50
ELSEIF(AAC_SUM.LT.4) THEN
IF(AAC_SUM.GT.PAAC_SUM) THEN
PAAC_SUM=AAC_SUM
SAVE=DISTORT_NOW
ENDIF
TEMP = I
I=I-DELTA
IF(.NOT.(I.LT.PREVI.AND.TEMP.LT.PREVI)) THEN
DELTA=DELTA/2.
ENDIF
IF(DBGFLG.EQ.1)PRINT*,AAC_SUM,I,PREVI,DISTORT_NOW
GOTO 50
ENDIF
ENDDO
C SUM THE BPP OVER THE ENTIRE IMAGE
AAC_SUM=AAC_SUM/NUM_BLKS**2
IF(CNT.EQ.1) PAAC_SUM=AAC_SUM
IF(CNT.LT.12) THEN
IF(AAC_SUM.GE.4.) THEN
TEMP = I
I=I+DELTA
IF(.NOT.(I.GT.PREVI.AND.TEMP.GT.PREVI)) THEN
DELTA=DELTA/2.
I=PREVI+DELTA
ELSE
PREVI=TEMP
ENDIF
IF(I.GT.12.) I=12.
DISTORT_NOW = 2.**I
IF(DBGFLG.EQ.1)PRINT*,AAC_SUM,I,PREVI,DISTORT_NOW
GOTO 50
ELSEIF(AAC_SUM.LT.4) THEN
IF(AAC_SUM.GT.PAAC_SUM) THEN
PAAC_SUM=AAC_SUM
SAVE=DISTORT_NOW
ENDIF
TEMP = I
I=I-DELTA
IF(.NOT.(I.LT.PREVI.AND.TEMP.LT.PREVI)) THEN
DELTA=DELTA/2.
I = PREVI - DELTA
ELSE
ENDIF
IF (I.LT.1.) I = 1.
DISTORT NOW = 2. ** I
IF (DBGFLG.EQ.1) PRINT*, AAC, SUM, I, PREVI, DISTORT NOW
GOTO 50
ELSE
C THE AVG OF THE AVG BPP ASSIGNED BY THE AAC
C IS 1 BPP OVER THE ENTIRE IMAGE
GOTO 101
ENDIF
ENDIF
IF (DBGFLG.EQ.20) THEN
PRINT*, ' AAC EQUALITY FAILED TO CONVERGE SAVE = ', SAVE
ENDIF
DISTORT NOW = SAVE
DO R_NUM = 1, NUM_BKKS
DO C_NUM = 1, NUM_BKKS
TEMP = ENERGY(C_NUM, R_NUM) / 2.0
IF (TEMP.LT.0.0001) THEN
RAAC(C_NUM, R_NUM) = 0
ELSE
RAAC(C_NUM, R_NUM) = (ALOG(TEMP) / ALOG(2.0))
ENDIF
END IF
ENDDO
ENDDO
C IF IT CONVERGES
101 CONTINUE
C
DO R_NUM = 1, NUM_BKKS
DO C_NUM = 1, NUM_BKKS
IF (RAAC(C_NUM, R_NUM).LT.1.0) THEN
AAC(C_NUM, R_NUM) = 1
ELSEIF (RAAC(C_NUM, R_NUM).GE.9.0) THEN
AAC(C_NUM, R_NUM) = S
ELSE
AAC(C_NUM, R_NUM) = INT(RAAC(C_NUM, R_NUM))
ENDIF
ENDIF
ENDDO
ENDDO
C WRITE THE AAC COMPARE DATA TO FILE IF RGB DONE
DO C_NUM = 1, NUM_BKKS
WRITE(17, 25) (AAC(C_NUM, X), X = 1, NUM_BKKS)
ENDDO
C
TABLE OF AVG BITS PER WORD FOR AACS
SET(1) = .375
SET(2) = .375
SET(3) = .6875
SET(4) = 1.0
SET(5) = 1.3125
SET(6) = 1.625
SET(7) = 1.9375
SET(8) = 2.25
RETURN
END
SUBROUTINE CALC_SDM
IMPLICIT NONE
INCLUDE 'COLOR.INC'
INTEGER X,Y,R_NUM,C_NUM,M,N,X_LOC,Y_LOC,COORD,START
REAL TEMP(16,128),AVG,SQDEV
REAL TEMP_SUM(16,128),SD
INTEGER FUNIT,CNT(8)
DO CODE=1,8
  CNT(CODE)=0
ENDDO
DO X=1,16
  DO Y=1,128
    TEMP(X,Y)=0
    TEMP_SUM(X,Y)=0
  ENDDO
ENDDO
DO CODE=1,8
  START = (COORD-1) * N_COEF_ROW
DO FOR EACH XFORM BLOCK
  DO R_NUM = 1,NUM_BLKS
    DO C_NUM = 1,NUM_BLKS
      X = (R_NUM-1) * N_COEF_ROW
      Y = (C_NUM-1) * N_COEF_ROW
      USE COEFFICIENTS OF BLOCKS FOR SAME AAC TO CALCULATE SDM
      IF(AAC(C_NUM,R_NUM).EQ.COORD) THEN
        CNT(CODE)=CNT(CODE)+1
        IF(DBGFLG.EQ.1) PRINT*,'
        X LOC = 1
      DO FOR EACH COEFF LOCATION
        DO N = X,N_COEF_ROW+(X-1)
          Y_LOC = 1
          DO M = Y,N_COEF_ROW+(Y-1)
            TEMP(Y.LOC,X_LOC+START)=DCT(M,N)+
            TEMP(Y_LOC,X_LOC+START)
            Y_LOC = Y_LOC + 1
          ENDDO
          X LOC = X LOC + 1
        ENDDO
      ENDDO
    ENDDO
  ENDDO
  ALL OF CODE X ARE DONE AND ADDED TO TEMP
DO FOR EACH BLOCK
  DO R_NUM = 1,NUM_BLKS
    DO C_NUM = 1,NUM_BLKS
      X = (R_NUM-1) * N_COEF_ROW
    ENDDO
  ENDDO
\[ Y = (C_{\text{NUM}}-1) \cdot N_{\text{COEF ROW}} \]

C EACH COEFFICIENT

IF(AAC(C_{\text{NUM}}, R_{\text{NUM}}).EQ.CODE) THEN

\[ X_{\text{LOC}} = 1 \]

DO N = X, N_{\text{COEF ROW}} + (X-1)

\[ Y_{\text{LOC}} = 1 \]

DO M = Y, N_{\text{COEF ROW}} + (Y-1)

IF(DBGFLG.EQ.1) PRINT*, ' TEMP ', TEMP(Y_{\text{LOC}}, X_{\text{LOC}}+START)

\[ \text{AVG} = \text{TEMP}(Y_{\text{LOC}}, X_{\text{LOC}}+\text{START}) / \text{FLOAT}(\text{CNT}(\text{CODE})) \]

C TAKE CARE OF THE CASE OF ALL ZERO COEFFICIENTS

IF(ABS(AVG).LT.0.000001) THEN

ZERO_ARRAY(Y_{\text{LOC}}, X_{\text{LOC}}+\text{START}) = 0

ELSE

ZERO_ARRAY(Y_{\text{LOC}}, X_{\text{LOC}}+\text{START}) = 99

ENDIF

C

\[ \text{SQDEV} = \text{DCT}(M,N) - \text{AVG} \]

\[ \text{TEMP SUM}(Y_{\text{LOC}}, X_{\text{LOC}}+\text{START}) = \text{TEMP SUM}(Y_{\text{LOC}}, X_{\text{LOC}}+\text{START}) + \text{SQDEV} \]

\[ Y_{\text{LOC}} = Y_{\text{LOC}} + 1 \]

ENDDO

\[ X_{\text{LOC}} = X_{\text{LOC}} + 1 \]

ENDDO

ENDIF

C ENDDO ENDDO

C TAKE RMS OF DEV

IF(CNT(CODE).NE.0) THEN

DO X_{\text{LOC}} = 1, N_{\text{COEF ROW}}

\[ SD = \text{SQRT}(\text{TEMP SUM}(Y_{\text{LOC}}, X_{\text{LOC}}+\text{START})/\text{FLOAT}(\text{CNT}(\text{CODE}))) \]

IF(SD.GT.0.1) THEN

SDM(Y_{\text{LOC}}, START+X_{\text{LOC}}) = SD

ELSE

SDM(Y_{\text{LOC}}, START+X_{\text{LOC}}) = 0

ENDIF

ENDDO

ENDDO

ELSE

DO X_{\text{LOC}} = 1, N_{\text{COEF ROW}}

\[ SDM(Y_{\text{LOC}}, START+X_{\text{LOC}}) = 0 \]

ENDDO

ENDDO

ENDIF

C END OF THAT AAC CODE

C ENDDO

C WRITE TO DATAFILE IF RGB DONE

PRINT*, ' COLOR ', COLOR

C

DO CODE=1,8

DO Y=1, N_{\text{COEF ROW}}

\[ X_{\text{LOC}} = (\text{CODE}-1) \cdot N_{\text{COEF ROW}} + 1 \]

WRITE(16,25)(SDM(Y,X), X=X_{\text{LOC}}, X_{\text{LOC}}+N_{\text{COEF ROW}}-1)
SUBROUTINE CALC_BAM

IMPLICIT NONE
INCLUDE 'COLOR.INC'

INTEGER CODE, X_LOC, Y_LOC, XX, CNT, FUNIT, X, Y
INTEGER INT, YY, NUM_DIS, END_X_LOC, END_Y_LOC
REAL BAM_SUM, TEMP, BAM_AVG, SAVE, BEST
REAL INITIAL, NBPBLK, DISTORT_NOW
LOGICAL POSSIBLE, SOLN

INITIAL = DISTORT_BAM

DO X_LOC = 1, 128
  DO Y_LOC = 1, 16
    BAM(Y_LOC, X_LOC) = 0
  ENDDO
ENDDO

DO CODE = 1, 8
  DISTORT_NOW = INITIAL
  SOLN = .FALSE.
  NBPBLK = SET(CODE) * N_COEF_ROW * N_COEF_ROW
  BEST = 0
  CNT = 0
  SAVE = 0
  XX = (CODE - 1) * N_COEF_ROW + 1
  CONTINUE
  BAM_SUM = 0
  BAM_AVG = 0
  POSSIBLE = .FALSE.
  CNT = CNT + 1
  DO X_LOC = XX, XX + N_COEF_ROW - 1
    DO Y_LOC = 1, N_COEF_ROW
      IF(SDM(Y_LOC, X_LOC) .NE. 0) THEN
        TEMP = ALOG(SDM(Y_LOC, X_LOC)) / ALOG(2.0)
        BAM(Y_LOC, X_LOC) = INT(TEMP - DISTORT_NOW)
        IF(BAM(Y_LOC, X_LOC) .GT. 8) THEN
          BAM(Y_LOC, X_LOC) = 8
        ENDIF
        IF(BAM(Y_LOC, X_LOC) .LT. 0) THEN
          BAM(Y_LOC, X_LOC) = 0
        ENDIF
      ELSE
        IF(ZERO_ARRAY(Y_LOC, X_LOC) .EQ. 0) THEN
          BAM(Y_LOC, X_LOC) = 0
        ELSE
          BAM(Y_LOC, X_LOC) = 1
        ENDIF
      ENDIF
    ENDDO
  ENDDO
ENDIF
BAM_SUM = BAM(Y_LOC,X_LOC) + BAM_SUM
ENDDO
ENDDO
IF(.NOT..POSSIBLE) goto 789
C
BAM_AVG = BAM_SUM/FLOAT(N_COEF_ROW**2)
C
IF(CNT.LT.50) THEN
  IF(BAM_SUM.LT.NBPBLK) THEN
    IF((NBPBLK-BAM_SUM).LT.1.0) GOTO 30
    IF(DISTORT_NOW.EQ.0) THEN
      CNT = 49
    ELSE
      DISTORT_NOW = DISTORT_NOW - .02
    ENDIF
  ELSEIF(BAM_SUM.GT.NBPBLK) THEN
    IF(DISTORT_NOW.EQ.10.0) THEN
      CNT = 49
    ELSE
      DISTORT_NOW = DISTORT_NOW + .02
    ENDIF
  ENDIF
ELSE
  IF(DBGFLG.EQ.6) THEN
    PRINT*, 'BAM MATCH CODE = ', CODE
  ENDIF
GOTO 30
C THEY ARE EQUAL SO RETURN
ENDIF
C
ELSE
C IF CNT > 50 FIX BITS
C
789 IF(DBGFLG.EQ.5) THEN
  PRINT*, 'BAM CALC FAILED TO CONVERGE CODE = ', CODE
ENDIF
C USING LAST BEST ASSIGNMENT (IF EXISTING) AND DISTRIBUTE
C REMAINING BITS
C RE-INITIALIZE BAM
DO X_LOC=XX,XX+N_COEF_ROW-1
  DO Y_LOC=1,N_COEF_ROW
    BAM(Y_LOC,X_LOC)=0
  ENDDO
ENDDO
C
IF(SOLN) THEN
  DISTORT_NOW = SAVE
C
DO X_LOC = XX,XX+N_COEF_ROW-1
  DO Y_LOC = 1,N_COEF_ROW
    IF(SDM(Y_LOC,X_LOC).NE.0) THEN
      TEMP = ALOG(SDM(Y_LOC,X_LOC))/ALOG(2.0)
    ENDIF
  ENDDO
ENDDO
BAM(Y_LOC,X_LOC) = INT(TMP-DISTORT_NOW)
IF(BAM(Y_LOC,X_LOC).GT.8) BAM(Y_LOC,X_LOC) = 8
IF(BAM(Y_LOC,X_LOC).LT.0) BAM(Y_LOC,X_LOC) = 0
POSSIBLE = .TRUE.
ELSE
BAM(Y_LOC,X_LOC) = 0
ENDIF
ENDDO
ENDDO
C FIND OUT HOW MANY BITS ARE REMAINING
NBPBLK = NBPBLK - BEST
ENDIF
C
C DO ASSIGNING OF BAM VALUES
C
NUM_DIS = INT(NBPBLK)
END_Y_LOC = 0
END_X_LOC = XX - 1
DO WHILE(NUM_DIS.GT.0)
END_Y_LOC = END_Y_LOC + 1
END_X_LOC = END_X_LOC + 1
IF(END_X_LOC.GT.XX + N_COEF_ROW - 1) END_X_LOC = XX
IF(END_Y_LOC.GT.N_COEF_ROW) END_Y_LOC = 1
DO X_LOC = XX, END_X_LOC
DO Y_LOC = 1, END_Y_LOC
IF(NUM_DIS.GT.0) THEN
IF(BAM(Y_LOC,X_LOC).LT.B) THEN
BAM(Y_LOC,X_LOC) = BAM(Y_LOC,X_LOC) + 1
NUM_DIS = NUM_DIS - 1
ENDIF
ENDIF
ENDDO
ENDDO
ENDDO
C
ENDIF
C
ENDDO
C END OF CODE LOOP
C
C WRITE BAM DATA TO FILE WHEN ALL RGB ARE DONE
C
PRINT*, 'COLOR', 'COLOR', COLOR
C
DO CODE = 1, 8
DO Y = 1, N_COEF_ROW
X_LOC = (CODE - 1) * N_COEF_ROW + 1
WRITE(15, 25)(BAM(Y, X), X = X_LOC, X_LOC + N_COEF_ROW - 1)
ENDDO
ENDDO
25 FORMAT(1X, '<N_COEF_ROW>12)
C
RETURN
END
SUBROUTINE CALC_DCT

THIS ROUTINE CALCULATES THE DISCRETE COSINE TRANSFORM
FOR EACH TRANSFORM BLOCK IN THE IMAGE
CALLED FOR EACH XFORM BLOCK

IMPLICIT NONE
INCLUDE 'COLOR.INC'

INTEGER J,K,M,N,R_BLK_NUM,C_BLK_NUM,X,Y,XX,YY
INTEGER P,Q,N_COEF
REAL C(0:511,0:511),PI,R_COEF,SQRT
REAL FJ,FK,FM,FN,RN,FREQ~SUM
REAL NORM

COEFFICIENTS TO RETAIN
IF(INITIAL FLG.EQ.0)THEN
500
PRINT*, 'NUMBER OF COEFFICIENTS PER ROW '
READ(5,*,ERR=500)N_COEF
N_COEF_ROW=N_COEF
IF(DBGFLG.EQ.1) PRINT*, ' N_COEF_ROW ',N_COEF_ROW
ENDIF

RN=FLOAT(BLK_SIZE)
PI = 3.1415926
E_SCALE=0.70710678
NORM=FLOAT(4)/(RN*RN)

CLEAR DCT ARRAY
DO N=0,NUM_BLKS*N_COEF_ROW-1
    DO M=0,NUM_BLKS*N_COEF_ROW-1
        DCT(M,N)=0
    ENDDO
ENDDO

DO R_BLK_NUM = 1,NUM_BLKS
    IF(DBGFLG.EQ.2) PRINT*, ' R_BLK_NUM = ',R_BLK_NUM
    XX=(R_BLK_NUM-1)*N_COEF_ROW
    X=(R_BLK_NUM-1)*BLK_SIZE
    DO C_BLK_NUM = 1,NUM_BLKS
        IF(DBGFLG.EQ.2) PRINT*, ' C_BLK_NUM = ',C_BLK_NUM
        YY=(C_BLK_NUM-1)*N_COEF_ROW
        Y=(C_BLK_NUM-1)*BLK_SIZE
        FN=-1.0
        DO N=XX,N_COEF_ROW+XX-1
            FN=FN+1.0
            FM=-1.0
            DO M=YY,N_COEF_ROW+yy-1
                FM=FM+1.0
                FJ=-1.0
                DO J=Y,Y+BLK_SIZE-1
                    FJ=FJ+1.0
                    C(J,M)=COS(((2.0*FJ+1.0)*PI*FM)/(2.*RN))
                ENDDO
                FK=1.0
                DO K=X,X+BLK_SIZE-1
                    FK=FK+1.0
                    C(K,N)=COS(((2.0*FK+1.0)*PI*FN)/(2.*RN))
                ENDDO
            ENDDO
        ENDDO
    ENDDO
ENDDO

DCT(M,N)=FLOAT(IMAGE(J,K))*C(J,M)*C(K,N)+DCT(M,N)
C DCT(M,N) = DCT(M,N) * NORM
C IF FIRST TERM CALC THE DC VALUE
IF(M.EQ.YY) THEN
  DCT(M,N) = DCT(M,N) * E SCALE
ENDIF
IF(N.EQ.XX) THEN
  DCT(M,N) = DCT(M,N) * E SCALE
ENDIF
END
END

C CALCULATED ALL OF THE COEFFICIENTS
ENDDO

C DONE FOR ALL BLOCKS
25 FORMAT(1X,2I5,1F12.6)
RETURN
END

C******************************************************************************
C SUBROUTINE NORMALIZE
C
C NORMALIZE EACH COEFFICIENT BY ITS CORRESPONDING SDM VALUE
C
IMPLICIT NONE
INCLUDE 'COLOR.INC'

INTEGER X,Y,XX,Y_LOC,X_LOC,R_NUM,C_NUM,N,M
REAL NORM
INTEGER BIN

C EACH BLOCK
DO R_NUM = 1,NUM_BK
  DO C_NUM = 1,NUM_BK
    X = (R_NUM - 1) * N_COEF_ROW
    Y = (C_NUM - 1) * N_COEF_ROW
    X_LOC=0
    DO N = X,N_COEF_ROW + (X-1)
      X_LOC=X_LOC+1
      Y_LOC=0
      DO M = Y,N_COEF_ROW + (Y-1)
        Y_LOC=Y_LOC+1
        XX = (AAC(C_NUM,R_NUM)-1)*N_COEF_ROW
        IF(DBGFLG.EQ.44) THEN
          BIN=BAM(Y_LOC,XX+X_LOC)
          IF(BIN.EQ.0) THEN
            PRINT*, ' DCT SOM ', DCT(M,N),SDM(Y_LOC,XX+X_LOC)
          ENDIF
        ENDIF
        IF(SDM(Y_LOC,XX+X_LOC).NE.0) THEN
          NORM = SDM(Y_LOC,XX+X_LOC)
          DCT(M,N) = DCT(M,N)/NORM
        ENDIF
      ENDDO
    ENDDO
  ENDDO
END
END

C******************************************************************************
C ENDDO
RETURN
END
C********************************************
SUBROUTINE QUANTIZE
C
C This routine is used to take all of the coefficients assigned
C the same number of bits and construct a histogram, perform
C Max's algorithm, and assign the quantization levels to be output.
C
IMPLICIT NONE
INCLUDE 'COLOR.INC'

INTEGER CNT(0:8), BIN, R_NUM, C_NUM, X, Y, N, M
INTEGER I, J, LEVEL, II, STARTI
INTEGER XLOC, YLOC
REAL MIN(0:8), MAX(0:8), FACTOR
INTEGER LOC, LEVELS, DELTA, END_INDEX
REAL IQLUT(0:8, 0:255)
REAL TEMP
INTEGER*2 K, L
INTEGER START(256)
REAL IQLUT_AVG
LOGICAL SWITCH

PRINT*, ' ENTERING QUANTIZE ';

DO BIN = 0, 8
  CNT(BIN) = 0
  MAX(BIN) = -99999
  MIN(BIN) = 99999
ENDDO

DO R_NUM = 1, NUM_BLKS
  DO C_NUM = 1, NUM_BLKS
    X = (R_NUM - 1) * N_COEF_ROW
    Y = (C_NUM - 1) * N_COEF_ROW
    XLOC = 0
    DO N = X, N_COEF_ROW + (X - 1)
      XLOC = XLOC + 1
      YLOC = 0
      DO M = Y, N_COEF_ROW + (Y - 1)
        YLOC = YLOC + 1
        LOC = AAC(C_NUM, R_NUM) - 1) * N_COEF_ROW + 1
        BIN = BAM(YLOC, LOC + XLOC - 1)
        CNT(BIN) = CNT(BIN) + 1
        PTRM(BIN, CNT(BIN)) = M
        PTRN(BIN, CNT(BIN)) = N
      END DO
    END DO
  END DO
END DO

IF(DBGFLG.EQ.69)
  PRINT*, ', BIG LOOP', NUM_BLKS, BLK_SIZE, N_COEF_ROW
ENDIF

C COLLECT ALL NORMALIZED COEFS TO BE CODED WITH THE SAME NUM OF BITS
C PER PIXEL (AS PER BAM)

C FIND MAX AND MIN VALUE FOR EACH BIN CATEGORY
IF(DCT(M,N) .LT. MIN(BIN))
  MIN(BIN) = DCT(M,N)
IF(DCT(M,N) .GT. MAX(BIN))
  MAX(BIN) = DCT(M,N)

END LOOP

C END K LOOP
ENDDO
C END C_NUM LOOP
ENDDO
C END R_NUM LOOP
ENDDO

C PRINT*, ' ENTERING SORT '
DO BIN = 1, 8
  PRINT*, ' SORTING BIN ', BIN
  IF(CNT(BIN) .GT. 0 .AND. MIN(BIN) .NE. MAX(BIN)) THEN
    SWITCH = .TRUE.
    STARTI = 1
    DO WHILE(SWITCH)
      SWITCH = .FALSE.
      DO I = STARTI, CNT(BIN) - 1
        IF(DCT(PTRM(BIN,I), PTRN(BIN,I)) .GT. DCT(PTRM(BIN,I+1), PTRN(BIN,I+1))) THEN
          M = PTRM(BIN,I)
          PTRM(BIN,I) = PTRM(BIN,I+1)
          PTRM(BIN,I+1) = M
          N = PTRN(BIN,I)
          PTRN(BIN,I) = PTRN(BIN,I+1)
          PTRN(BIN,I+1) = N
          IF(.NOT. SWITCH) STARTI = I - 1
          IF(SWITCH .AND. STARTI .LT. 1) STARTI = 1
          SWITCH = .TRUE.
        ENDIF
      ENDDO
    ENDDO
  ENDIF
ENDDO
END LOOP
ENDDO
ENDIF
ENDDO
ENDO

C PRINT*, ' EXITING SORT '
C LOAD THE IQLUT VALUES FOR BIN 0
C
C IF(DBGFLG.EQ.1000) WRITE(33,33)
33  FORMAT(' BIN 0 ',//,' DCT M N ')
  IF(CNT(0) .GT. 0) THEN
    DO I = 1, CNT(0)
      IQLUT_AVG = IQLUT_AVG + DCT(PTRM(0,I), PTRN(0,I))
    ENDDO
    IQLUT(0,0) = IQLUT_AVG / CNT(0)
    DO I = 1, CNT(0)
      IMAGE(PTRM(0,I), PTRN(0,I)) = 0
    ENDDO
  ELSE
    IQLUT(0,0) = 0.
  ENDIF
WRITE(12,329) IQLUT(0,0)
329  FORMAT(1X,F14.9)
C DETERMINE QLUT FOR BIT ASSIGNMENT
C FORM QLUTS

DO BIN=1,8
   IF(CNT(BIN).GT.0 .AND. MAX(BIN).NE.MIN(BIN)) THEN
      LEVELS=2**BIN
      START(1)=1
      DELTA=NINT(FLOAT(CNT(BIN))/FLOAT(LEVELS))
      IF(DELTA.LT.1) DELTA=1
      DO I=2,LEVELS
         START(I)=DELTA+START(I-1)
      ENDDO
   END
   DO I = 1, LEVELS
      IQLUT_AVG=0
      IF(START(I).LT.CNT(BIN)) THEN
         IF(I.EQ.LEVELS) THEN
            END_INDEX=CNT(BIN)
         ELSE
            END_INDEX=START(I+1)
         ENDIF
         IF(END_INDEX.GT.CNT(BIN)) END_INDEX=CNT(BIN)
         DO J=START(I),END_INDEX
            IQLUT_AVG=IQLUT_AVG+DCT(PTRM(BIN,J),PTRN(BIN,J))
            IMAGE(PTRM(BIN,J),PTRN(BIN,J))=I-1
         END DO
         IQLUT(BIN,I-1)=IQLUT_AVG/FLOAT(END_INDEX-START(I)+1)
      ELSE
         IQLUT(BIN,I-1)=99.
      ENDIF
      ENDDO
   ELSEIF(MAX(BIN).EQ.MIN(BIN)) THEN
      DO J=1,CNT(BIN)
         IMAGE(PTRM(BIN,J),PTRN(BIN,J))=0
      ENDDO
      IQLUT(BIN,0)=MAX(BIN)
   ENDIF
   WRITE(12,29)(IQLUT(BIN,LEVEL),LEVEL=0,255)
ENDDO
WRITE(12,29)(IQLUT(BIN,LEVEL),LEVEL=0,255)
FORMAT(1X,32(8(F14.9),/))
ENDDO
RETURN
PROGRAM DECOMP

THIS ROUTINE IS USED TO DECOMPRESS THE IMAGE FROM THE ADAPTIVE DCT METHOD

IMPLICIT NONE INCLUDE 'DECOMP.INC'

INTEGER*2 RED_BUF(0:511),GRE_BUF(0:511),BLU_BUF(0:511)
BYTE DC_RED_IMAGE(0:511),DC_GREEN_IMAGE(0:511)
BYTE DC_BLUE_IMAGE(0:511),BYTE
REAL MAX(0:8),LEV(8),NORM,NORM_DCT,RNO
REAL SORT,IQUT_ARRAY(0:8,0:255),SUM(16,128)
INTEGER LEVEL,C_BAM(16,128)
INTEGER*2 C_RED_IMAGE(0:511,0:511)
INTEGER AAC(128,128),COLOR
INTEGER*2 C_GREEN_IMAGE(0:511,0:511)
INTEGER*2 C_BLUE_IMAGE(0:511,0:511)
INTEGER BIN,XX,K,J,I,R_NUM,C_NUM,X,Y,N,M
INTEGER POS,FUNIT,IOS,CODE,YT,PIXEL,XLOC,CONSTANT
INTEGER TEMPX,YLOC,IMAGE_SIZE

EQUIVALENCE (BYTE,PIXEL)

C DEFINITION OF THE MAXIMUM VALUE POSSIBLE FOR EACH BIT
C ASSIGNMENT (0-8 BITS CORRESPONDS TO 0-255 LEVELS)

PRINT*, 'ENTER BLK SIZE (16,8,4)'
READ*,BLK_SIZE
PRINT*, 'IMAGE SIZE'
READ*,IMAGE_SIZE

NUM_BLKS=IMAGE_SIZE/BLK_SIZE
PRINT*, 'ENTER NUM COEF PER BLOCK'
READ*,N_COEF_ROW

OPEN(UNIT=17,NAME='AACFILE.DAT',TYPE='UNKNOWN', FORM='FORMATTED')

OPEN(UNIT=15,NAME='BAMFILE.DAT',TYPE='UNKNOWN', FORM='FORMATTED')

OPEN(UNIT=16,NAME='SDMFILE.DAT',TYPE='UNKNOWN', FORM='FORMATTED')

OPEN COMPRRESSED IMAGE FILE

OPEN(UNIT=14,NAME='CSCENE1.DAT',TYPE='UNKNOWN', FORM='UNFORMATTED',RECL=128,IOSTAT=IOS,ERR=22)

444 FORMAT(1X,8I4)
N=NUM_BLKS*N_COEF_ROW
DO I=0,N-1
READ(14,ERR=23,IOSTAT=IOS)(C_RED_IMAGE(K,I),K=0,N-1)
WRITE(70,444)(C_RED_IMAGE(K,I),K=0,N-1)
ENDDO
DO I=0,N-1
READ(14,ERR=23,IOSTAT=IOS)(C_GREEN_IMAGE(K,I),K=0,N-1)
WRITE(70,444)(C_GREEN_IMAGE(K,I),K=0,N-1)
ENDDO
DO I=0,N-1
READ(14,ERR=23,IOSTAT=IOS)(C_BLUE_IMAGE(K,I),K=0,N-1)
WRITE(70,444)(C_BLUE_IMAGE(K,I),K=0,N-1)


ENDDO
C C CLOSE FILE
CLOSE(UNIT=14)
C
C OPEN THE IQLUT FILE
OPEN(UNIT=12,NAME='IQLUT.DAT',TYPE= 'UNKNOWN',FORM='FORMATTED')
C
C PROCESS THE RED, GREEN AND BLUE IMAGE SEPARATELY
DO K=1,3
  IF(K.EQ.1)COLOR=1
  IF(K.EQ.2)COLOR=2
  IF(K.EQ.3)COLOR=3
  PRINT*, 'COLOR=', COLOR
  DO J=0,NUM*BLKS*N_COEF_ROW-1
    DO I=0,NUM*BLKS*N_COEF_ROW-1
      IF(K.EQ.1) IMAGE(I,J)=C_RED_IMAGE(I,J)
      IF(K.EQ.2) IMAGE(I,J)=C_GREEN_IMAGE(I,J)
      IF(K.EQ.3) IMAGE(I,J)=C_BLUE_IMAGE(I,J)
    ENDDO
  END DO
ENDDO
C
C READ IN THE IQLUT ARRAY
READ(12,37) IQLUT_ARRAY(0,0)
FORMAT(1X,F14.9)
DO BIN=1,8
  PRINT*, 'BIN=', BIN
  READ(12,28)(IQLUT_ARRAY(BIN,LEVEL),LEVEL=0,255)
  FORMAT(1X,32(8(F14.9),//))
ENDDO
C
C READ IN AAC
DO C_NUM=1,NUM*BLKS
  READ(17,234)(AAC(C_NUM,R_NUM),R_NUM=1,NUM*BLKS)
ENDDO
C
C READ IN BAM
DO CODE=1,8
  XLOC=(CODE-1)*N_COEF_ROW+1
  DO Y=1,N_COEF_ROW
    READ(15,235)(BAM(Y,X),X=XLOC,XLOC+N_COEF_ROW-1)
  ENDDO
ENDDO
C
C READ IN SDM
DO CODE=1,8
  XLOC=(CODE-1)*N_COEF_ROW+1
  DO Y=1,N_COEF_ROW
    READ(16,236)(SDM(Y,X),X=XLOC,XLOC+N_COEF_ROW-1)
  ENDDO
ENDDO
234 FORMAT(1X,<NUM*BLKS>12)
235 FORMAT(1X,<N_COEF_ROW>12)
236 FORMAT(1X,<N_COEF_ROW>F8.3)
C
C PROCESS EACH TRANSFORM BLOCK
DO R_NUM=1,NUM*BLKS
  X=(R_NUM-1)*N_COEF_ROW
  DO C_NUM=1,NUM*BLKS
    Y=(C_NUM-1)*N_COEF_ROW
    XLOC=0
    DO N=X,N_COEF_ROW+K-1
C HAVE AN OVERHEAD FILE ASSOCIATED WITH EACH IMAGE
C CONTAINING THE AAC ARRAY AND IQLUT ARRAY FOR THE IMAGE,
C ALSO HAVE THE BAM AND SDM.
C
C INVERSE QUANTIZATION
LEVEL=IMAGE(M,N)
C
C DETERMINE BIT ASSIGNMENT
TEMpx=(AAC(C NUM,R NUM)-1)*N COEF ROW
BIN=BAM(YLOC,TEMPX+XLOC)
C
C CALCULATE THE NORMALIZED DCT COEFFICIENT
NORM DCT=IQLUT ARRAY(BIN,LEVEL)
IF(NORM DCT.EQ.99.0) THEN
PRINT*, 'BIN LEVEL =99 ',BIN,LEVEL
ENDIF
C INVERSE NORMALIZATION
NORM=SDM(YLOC,TEMPX+XLOC)
IF(NORM.EQ.0) THEN
DCT(M,N)=NORM DCT
ELSE
DCT(M,N)=NORM*NORM DCT
ENDIF
C
ENDDO
ENDDO
C
PERFORM THE INVERSE DCT
CALL CALC_IDCT
C
DO I=0,IMAGE SIZE-1
DO J=0,IMAGE SIZE-1
IF(K.EQ.1) THEN
C RED IMAGE(I,J)=INT(IDCT(I,J))
ELSEIF(K.EQ.2) THEN
C GREEN IMAGE(I,J)=INT(IDCT(I,J))
ELSEIF(K.EQ.3) THEN
C BLUE IMAGE(I,J)=INT(IDCT(I,J))
ENDIF
ENDO
C END OF K LOOP
ENDDO
C
CLOSE THE IQLUT ARRAY
CLOSE(UNIT=12)
C
OPEN NEW COLOR IMAGE FILE
PRINT*, 'ABOUT TO OPEN NEW FILE '
CALL LIBS$GET LUN(FUNIT)
OPEN(UNIT=FUNIT,NAME='DCMTLEPT.SCN',TYPE='UNKNOWN',
FORM='UNFORMATTED',DEFAULTFILE='.SCN',
RECL=128,RECORDTYPE='FIXED',IOSTAT=IOS,ERR=100)
PRINT*, 'OPENED DC FILE '
C CONVERT TO BYTE FORMAT FROM INTEGER
C
DO I=0,IMAGE_SIZE-1
DO J=0,IMAGE_SIZE-1
   PIXEL=(C_RED_IMAGE(I,J))
   DC_RED_IMAGE(J)=BYTE
   PIXEL=(C_GREEN_IMAGE(I,J))
   DC_GREEN_IMAGE(J)=BYTE
   PIXEL=(C_BLUE_IMAGE(I,J))
   DC_BLUE_IMAGE(J)=BYTE
ENDDO
C
C WRITE IN ALTERNATING RGB
WRITE(FUNIT)(DC_RED_IMAGE(POS),
   & POS=0,IMAGE_SIZE-1)
WRITE(FUNIT)(DC_GREEN_IMAGE(POS),
   & POS=0,IMAGE_SIZE-1)
WRITE(FUNIT)(DC_BLUE_IMAGE(POS),
   & POS=0,IMAGE_SIZE-1)
ENDDO
C
CLOSE(UNIT=FUNIT)
CALL LIB$FREE_LUN(FUNIT)
C
STOP
22 PRINT*," ERROR OPEN FILE -C IMAGE " IOSTAT=",IOS
STOP
23 PRINT*," ERROR READING C IMAGE " IOSTAT=",IOS
STOP
100 PRINT*," ERROR OPEN DC IMAGE " IOSTAT=",IOS
STOP
101 PRINT*," ERROR WRITING DC IMAGE " IOSTAT=",IOS
STOP
C
END
C END PROGRAM
C
*****************************************************************************
C SUBROUTINE CALC_IDCT
C
C THIS SUBROUTINE CALCULATES THE INVERSE DCT FOR THE\nC DECOMPRESSION
C
IMPLICIT NONE
INCLUDE 'DECOMP.INC'
INTEGER J,K,M,N,R_NUM,C_NUM,X,Y,XX,YY,I
REAL FJ,FK,FM,FN,SQRT,PI,RN,C(O:S11),O:S11)
REAL SUM,FREQ,E_SCALE
C
DO J=0,BLK_SIZE*NUM_BLKS-1
   DO I=0,BLK_SIZE*NUM_BLKS-1
      IDCT(I,J)=0
   ENDDO
ENDDO
C
PI=3.1415926
RN=FLOAT(BLK_SIZE)
E_SCALE=.70710678
C
DO R_NUM=1,NUM_BLKS
X=(R_NUM-1)*BLK_SIZE
XX=(R_NUM-1)*N_COEF_ROW
DO C_NUM=1,NUM.BLKS
Y=(C_NUM-1)*BLK_SIZE
YY=(C_NUM-1)*N_COEF_ROW
FK=-1.0
DO K=X,BLK_SIZE+X-1
FK=FK+1.0
FJ=-1.0
DO J=Y,BLK_SIZE+Y-1
SUM=0.0
FJ=FJ+1.0
FM=-1.0
DO M=YY,YY+N_COEF_ROW-1
FM=FM+1.0
C(J,M)=COS((2.0*FJ+1.0)*PI*FM/(2.*RN))
FN=-1.0
DO N=XX,XX+N_COEF_ROW-1
FN=FN+1.0
C(K,N)=COS((2.0*FK+1.0)*PI*FN/(2.*RN))
FREQ=DCT(M,N)
IF(M.EQ.YY) THEN
FREQ=FREQ*E_SCALE
ENDIF
IF(N.EQ.XX) THEN
FREQ=FREQ*E_SCALE
ENDIF
SUM=SUM+FREQ*C(J,M)*C(K,N)
ENDDO
ENDDO
C END OF LOOP THRU COEFFICIENTS
ENDDO
ENDDO
C END OF THE PIXELS FOR THE XFORM BLOCK
ENDDO
ENDDO
C END OF BLOCKS
PRINT*, ' LEAVING IDCT '
C RETURN
C
REFERENCES


