



GAP 2 : an image processing package for UNIX
by Yuri Kalmykov

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in
Computer Science
Montana State University
© Copyright by Yuri Kalmykov (1997)

Abstract:

The project presented in this thesis is a UNIX/XWindows Motif-based image processing application. It implements a set of image processing and manipulation functions both in the spatial and frequency domains. The frequency domain transforms implemented are Fourier, Walsh and Wavelet transforms. The speed of those transforms is compared, the Walsh transform being the fastest. The Wavelet transform is one of the most recent developments in the image/signal processing. It is very effective in a wide variety of applications, including image coding/ compression, image segmentation and noise removal. Low and highpass filters are available for Fourier and Walsh transforms, and noise reduction filter is implemented for the Wavelet transform. A short theoretic overview is given for most of the implemented functions. The program works with the images in RAW format (256-level uncompressed grayscale images).

GAP 2:
AN IMAGE PROCESSING PACKAGE FOR UNIX

by
Yuri Kalmykov

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

of
Master of Science
in
Computer Science

MONTANA STATE UNIVERSITY-BOZEMAN
Bozeman, Montana

May 1997

N378
K1265

APPROVAL

of a thesis submitted by

Yuri Kalmykov

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

Denbigh Starkey

JD Starkey
(Signature)

5/21/97
Date

Approved for the Department of Computer Science

Denbigh Starkey

JD Starkey
(Signature)

5/21/97
Date

Approved for the College of Graduate Studies

Robert Brown

RB
(Signature)

6/3/97
Date

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University-Bozeman, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature *Copy / Yuri Kalmykov*

Date *May 19, 1997*

TABLE OF CONTENTS

	Page
1. INTRODUCTION	1
2. PROJECT DESCRIPTION	3
User Interface and Related Features	3
Menu Bar	3
Menu Bar Building	7
Information Panel	8
Toolbar	8
Lens	12
Histogram	12
Dialog Boxes	13
Data Management and Display	15
"Real" Image Data	15
Color Adjustment and Displayed Image	16
3. SPATIAL DOMAIN OPERATORS	19
Median Smoothing	19
Edge Detection	21
Image Sharpening	22
Masking	24

TABLE OF CONTENTS -- Continued

	Page
Histogram Equalization	27
Contrast Stretch	28
Binarization	31
Thinner	32
4. FREQUENCY DOMAIN	33
Fourier Transform	33
Viewing Fourier Spectrum	35
Butterworth Filtering	36
Walsh Transform	39
Wavelet Transform	41
General Overview of Wavelet Transform	42
Wavelet Transform Implementation	43
Wavelet Filtering	45
Performance Comparison	47
REFERENCES	49

LIST OF TABLES

Table	Page
1. MenuItem Structure	7
2. Histogram Structure	13
3. ActionAreaItem Structure	14
4. Color Adjustment Procedure	18
5. Commonly Used Image Processing Masks	25
6. Fast Fourier Transform Procedure	34
7. The Wavelet Based Noise Reduction Filter	47
8. Frequency Domain Transforms Performance Comparison	48

LIST OF FIGURES

Figure	Page
1. GAP 2 Main Window	5
2. The Structure of GAP2's Main Window Motif Widgets	6
3. Simple Dialog Box to Enter Only One Parameter	11
4. Memory Buffers Structure for GAP2	15
5. Median Smoothing	20
6. Edge Detection Results	22
7. Roberts Gradient Results	23
8. Masking Dialog Box	25
9. Masking	26
10. Histogram Equalization	27
11. Contrast Stretch Graph	28
12. Contrast Stretch Dialog Box	29
13. Area Contrast Stretch Graph	30
14. Contrast Stretch	31
15. Binarization and Thinner	32
16. Fourier Spectrum	36
17. Butterworth Filtering Dialog Box	38
18. Butterworth Filtering	39
19. Walsh Transform	41

LIST OF FIGURES - Continued

Figure	Page
20. The Structure of Wavelet Decomposed Image for the Level 2 Transform	44
21. Wavelet Decomposition	45
22. Wavelet Filtering Results	46

ABSTRACT

The project presented in this thesis is a UNIX/XWindows Motif-based image processing application. It implements a set of image processing and manipulation functions both in the spatial and frequency domains. The frequency domain transforms implemented are Fourier, Walsh and Wavelet transforms. The speed of those transforms is compared, the Walsh transform being the fastest. The Wavelet transform is one of the most recent developments in the image/signal processing. It is very effective in a wide variety of applications, including image coding/ compression, image segmentation and noise removal. Low and highpass filters are available for Fourier and Walsh transforms, and noise reduction filter is implemented for the Wavelet transform. A short theoretic overview is given for most of the implemented functions. The program works with the images in RAW format (256-level uncompressed grayscale images).

CHAPTER 1

INTRODUCTION.

The aim of this project was the creation of an image processing application intended to work under the UNIX operating system, with the X Windows system. The described application (called GAP2) is written in C and employs the OSF/Motif user interface toolkit.

The goal was two-fold:

- a). To create a sophisticated, full scale Motif application and
- b). To research and implement a variety of image processing techniques.

GAP2 is the further improvement and development of the GAP ("Great Aerophoto Processor") IBM PC/MS Windows application developed in 1993. That version was very simple, it was capable of working on slow PC's (AT 286/386) that were most common in Russia at that time. Only the simplest image processing algorithms were implemented: masking, median filtering and several other local operations. Only spatial domain procedures were implemented (the procedure for Walsh transform was written and tested, but was not included in the final version of the program). Gap had no 'histogram-based' procedures either. The image was broken into small parts that were processed separately, because otherwise processing of the whole image at once took too much time (median smoothing took around 30 minutes for 512x512 image on an AT 486 DX/33). Processing was interactive: the user chose what operation to apply next. The overall goal was to obtain binary images

showing contours of artificial objects on Aerophoto (buildings, roads, etc.).

Since GAP2 was developed on a much more powerful computer system (DEC Alpha), it processes the whole image at once. Major improvements were made to the interface and spatial domain functions. Several new spatial domain functions were added. Plus, GAP2 contains 'histogram-based' operations and frequency domain operators based on three transforms: Fourier, Walsh and Wavelet.

GAP2 works with gray scale images stored in RAW format (up to 256 levels).

CHAPTER 2

PROJECT DESCRIPTION

User Interface and Related Features

The main window consists of the following areas shown in Figure 1:

- 1). Menu bar
- 2). Information panel
- 3). Toolbar
- 4). Image window
- 5). Lens
- 6). Histogram window.

The structure of Motif widgets that form the user interface of GAP2 is shown in Figure 2.

Menu Bar.

Menu bar contains the following items:

- 1). File.

It is a set of common file manipulation functions to be applied to image files:

- a). Load Image
- b). Save Image
- c). Delete Image.

All three functions invoke the standard Motif file selection dialog box. Also, this menu item contains the Quit option.

2). Tools.

A set of image manipulation tools:

- a). Flip image horizontally or vertically. Creates the mirror image relative to the horizontal or vertical image axes. These functions just reorder the rows (columns for vertical flip) of the image in reverse order.
- b). Rotation, 90 degrees clockwise or counter-clockwise. These functions transpose the image. In the clockwise case the first column of the initial image becomes the first row of the resulting image (and so on), in the counter-clockwise case the first row of the initial image becomes the first column of the new image..

3). Spatial.

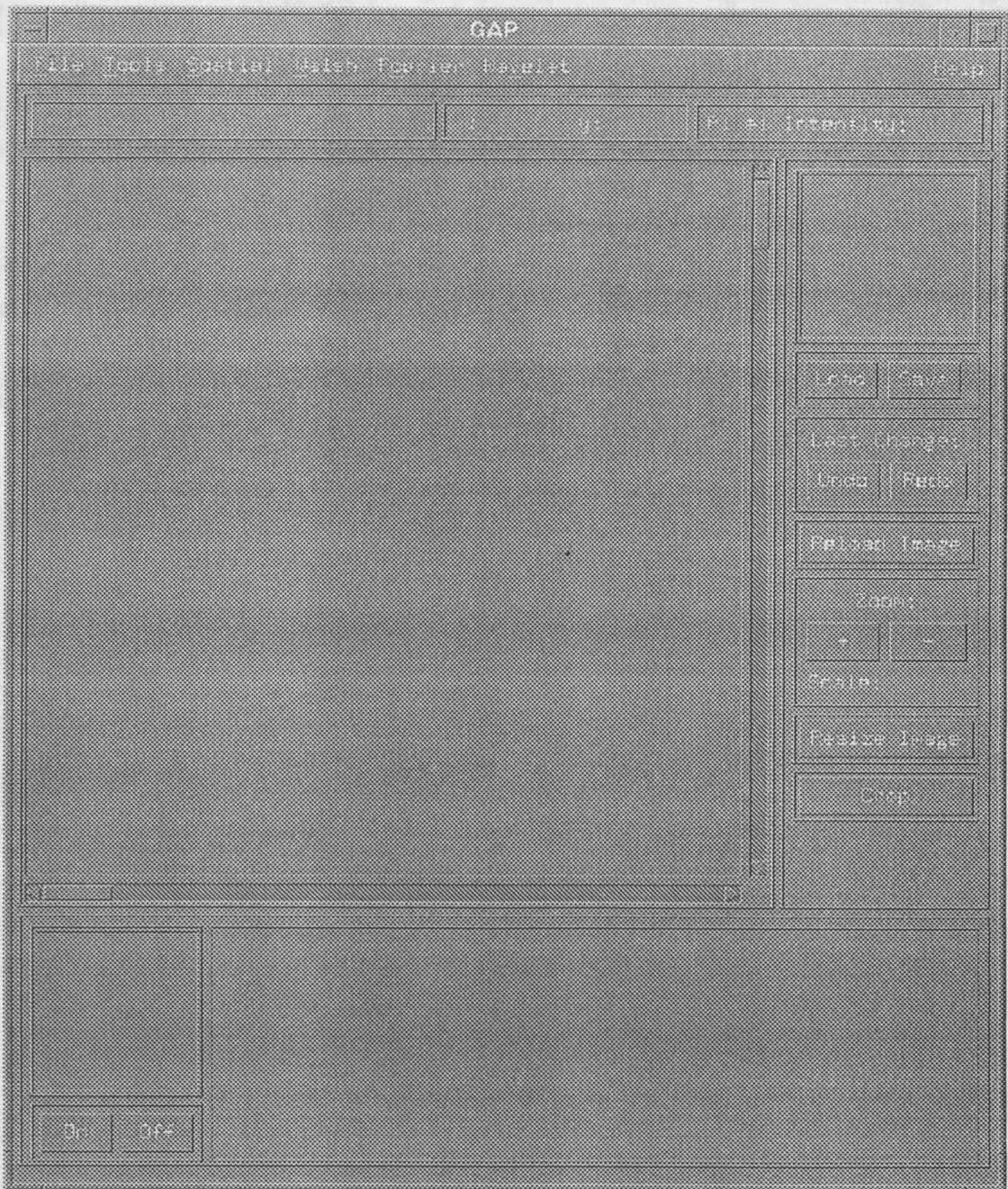
A set of spatial domain image processing functions.

4). Walsh, Fourier, Wavelet.

Image processing functions in frequency domain. All image processing functions will be discussed in detail in Chapter 2.

5). Help. User help information (was not implemented).

Figure 1. GAP2 Main Window



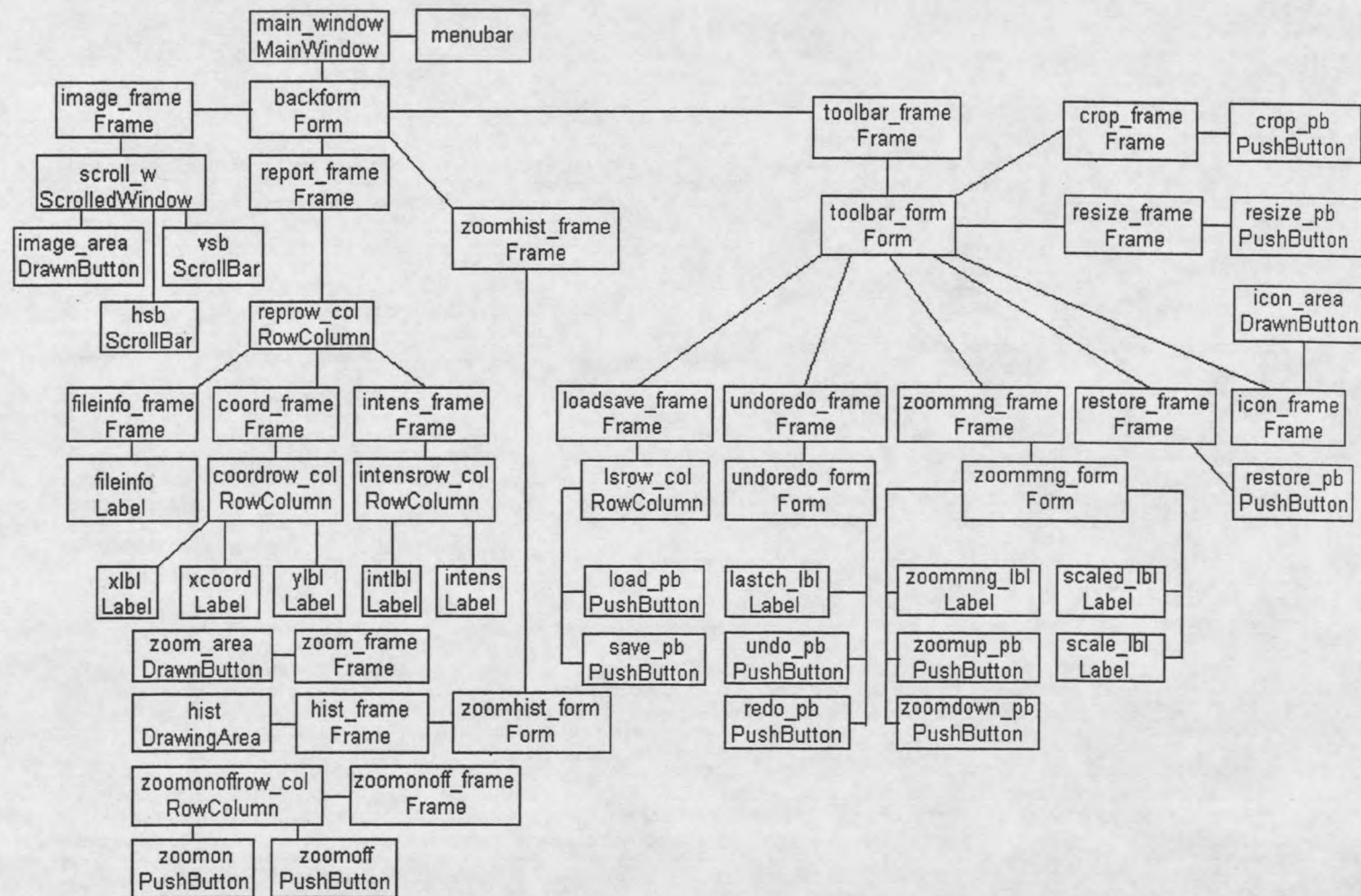


Figure 2. The Structure of GAP2's Main Window Motif Widgets.

Menu Bar Building.

For ease of menubar modifications (adding new functions, etc.), the special generic "menu bar builder" was used [1]. The basic "brick" of the menu is the pulldown menu, which, in its turn, is the set of buttons.

Each menu item (that is, pulldown menu), is the array of structures [1], shown in Table 1.

Table 1. MenuItem Structure.

```
typedef struct _menu_item
{
    char *label;           /* the label for the item of pulldown menu */
    WidgetClass *class;    /* pushbutton, label, separator... */
    char mnemonic;         /* "hot key"; NULL if none */
    char *accelerator;     /* accelerator; NULL if none */
    char *accel_text;      /* to be converted to compound string */
    void (*callback)();    /* routine to call; NULL if none */
    XtPointer callback_data; /* client data for callback() */
}
```

The actual menubar is built using two functions: CreateMenuBar and BuildPulldownMenu. The first one creates the upper level menu bar template and then "fills" it in, calling the second function to create all pulldown menus; arrays of MenuItem structures are used.

Thus, adding a new item to the pulldown menu is very easy: the new member must be added to the corresponding MenuItem array. The same is true for the whole new pulldown menu: it is necessary to define the new MenuItem array and to add one more call to BuildPulldownMenu in CreateMenuBar function.

Information Panel.

This panel consists of three parts.

1). File information.

The directory path and the name of the currently displayed image file; the dimensions of the image in pixels (rows x columns). These dimensions are always the dimensions of the initially loaded image, even if it was modified (resized or cropped).

2). Mouse pointer location on the image.

This data is derived from the actually displayed image, that is, if the image was resized, the shown position will reflect these changes.

3). Pixel intensity.

Shows the intensity of the pixel under the mouse pointer. The displayed intensity value is the "real" one, that is, the value that is contained in the source image file or the one that was derived from those values after image processing operations. This value might differ from those actually displayed on the screen. The discussion of display color adjustment will follow later in this Chapter.

Toolbar.

Toolbar is a set of buttons that allow the user to access routine interface features quickly. This form of control is generally more comfortable for the user than the menu. The following features are available:

1). Load and Save buttons.

These two just duplicate the menu functions, for the user's convenience.

2). Last change Undo and Redo.

This feature allows the user to undo or redo the last operation applied to the image. It affects any operation except zoom, since zoom doesn't actually change the image. Only one last operation can be undone or redone. Thus, these two buttons are mutually exclusive: when one is active, the other is disabled. If the user cancels some operation (the Redo button is active now), and applies some other image processing operation, the Redo button becomes inactive and the Undo button is activated. These operations are carried out by image memory buffers and image size values swaps between the current and previous ones. The memory management is discussed below.

3). Reload Image.

The same as "undo all changes," including zoom. After this button is clicked, the initial image is reloaded from the file and redisplayed.

4). Zoom.

This function enlarges or reduces the displayed size of the image without resizing it. Each consequent "+" button click enlarges the displayed size $(n-1)$ times, where n is the number of clicks. For instance, if the image was displayed in half-size, one click will return the image to its original size, two clicks will double its size, 3 clicks will enlarge the image three times and so on. The same is true for the "-" button: it reduces the currently displayed size by 2, 3, 4, ... times.

All pixel values for displaying zoomed/ reduced images are calculated from the initial image (the current zoom coefficient is stored in a special variable). This allows us to avoid unnecessary image distortions and data loss. When the image is zoomed twice, each pixel is

copied to an area of 2x2 pixels, 3 times zoom is copied into 3x3 pixels and so on. When the image is shrunk twice, each 2x2 pixel patch is averaged and this average value is assigned to the new pixel. 3 times shrink uses 3x3 patch. Thus, shrinking causes data loss. If we had based next zoom on the image that was previously reduced from its original size, that would cause very distorted results.

The current scale value of the image (in percent) is displayed under the buttons.

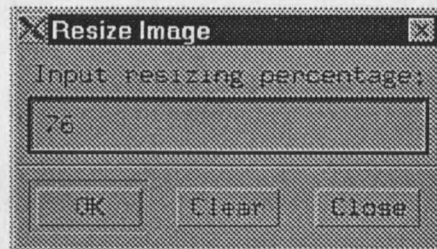
5). Resize.

This feature affects not only the display of the image, but also the image itself. The resizing coefficient might be not only integer, as in the case of zoom, but also fractional. Resizing coefficient is specified in percent: coefficient > 100% means enlargement, coefficient < 100% means reduction. Fractions of percent are not allowed. Resizing coefficients are entered through the dialog box. The resizing algorithm is simple. The percentile coefficients are represented in the form $\frac{n}{100}$. Then, the greatest common divisor of n and 100 is calculated. Both figures are divided by this GCD. Then, the image is firstly enlarged by the numerator of the resulting fraction, and reduced by the denominator. Due to the fact that these operations might take a lot of memory for the first step (enlargement) even after division by GCD, the whole operation is carried out in two passes. The first pass works with the rows of the initial image. Each row is enlarged and then reduced using the two coefficients. The result is saved in the intermediate buffer. Thus, we get an image with adjusted width. The second pass goes through the columns of the intermediate image. For each row or column: each pixel is copied into *numerator* pixels and then each *denominator* pixels of the resulting array are averaged into one pixel. If, after row or column zoom the number of pixels in it is

not a multiple of denominator, that is, the last group of pixels to be averaged is smaller than denominator, then just this number is averaged. Since a strip of just one pixel wide is affected, the possible error can be neglected.

Resizing is also done basing on the current image, not the initial one, due to potential problems of storing the image if it was processed before the resize. Thus, if the image was previously reduced compared to its initial size, it is not advisable to enlarge it back. Hence, it is better to do all image size adjustments immediately upon image loading. Otherwise, if one needs back enlargement there will be two choices: either get a distorted image or to lose all processing results because of image reload. If image resizing is not absolutely necessary, it is better to use the zoom function.

Figure 3. Simple Dialog Box Used to Enter Only One Parameter.



6). Crop.

This feature lets one extract a part of the image. "Rubberbanding" is used to specify the region to be cropped. Clicking and holding the left mouse button starts the rectangle. Dragging the mouse will draw it. The cropped image is taken from the original image. When the cropped region is displayed, its zoom scale is retained.

Lens.

Lens is a convenient feature that allows the user to see some particular small part of the image magnified without zooming the entire image (thus saving memory). This implementation picks a region of size 41x41 pixel with the center at the current mouse pointer location, and magnifies it three times.

Unfortunately, the X Windows programming library doesn't have a function allowing "direct" image magnification, without extraction of image data from XImage structure [2]. Because of this, two solutions are possible: either work with the *data* field of Ximage structure directly or use XGetPixel function that extracts data pixel by pixel. Since the image in question is small, and the second way is easier, it was used. Despite the use of slow XGetPixel function, the lens works amazingly fast, the picture in the lens window catches up with cursor movements easily, if the mouse is moved at a reasonable speed.

The same function in the program that operates the lens, also outputs mouse pointer coordinates and pixel intensity to the Information panel. The lens can be turned on or off using two buttons under the lens window.

Histogram.

For the users' convenience, the histogram of the current image is displayed in a separate panel of the main window. The histogram is calculated from the initial image, and is recalculated after every image processing operation that might change it (that is, except rotation or such). The histogram data is stored in the special structure shown in Table 2.

Table 2. Histogram Structure.

```
typedef struct
{
    int numpix;      /* image size */
    int count[256]; /* number of pixels of each color */
    int num_levels; /* number of gray levels of the image */
} histogram;
```

The histogram itself is drawn as a Pixmap image [1], [2]. The horizontal axis represents gray levels, the vertical axis is the number of pixels. Since we don't really need to know exact number of pixels for each gray level, but rather relative amounts, the vertical axis has only one label: the number of pixels of "the most represented" color.

Dialog Boxes.

The user enters parameters required by image processing functions using dialog boxes. Since most functions require user input, the number of dialog boxes is big. So, as in the case of the menu bar, it was useful to implement a generic procedure for dialog box creation. It was not as easy as for menus because dialog boxes might differ greatly from each other depending on their task. Nevertheless, all dialog boxes consist of two main parts: Control area and Action area. Control area (or Work area) contains Labels, Text fields, Toggle buttons and other Motif widgets for the user to control the application state. The Action area contains the set of PushButtons whose callback routines actually pick the values from the Control area widgets and take action on the application. The most typical Action area buttons are OK and Cancel. In reality, the "anatomy" of dialog boxes is much more complicated, but detailed discussion of dialog box programming is beyond the scope of this

thesis. See [1] for comprehensive information on this topic.

Since the Action area is being built nearly in the same way for every dialog box, the generic function `CreateActionArea` was used for this purpose [1]. As in the case of menu bar building, this function takes an array of special structures as its input. This structure format is shown in Table 3.

Table 3. `ActionAreaItem` Structure.

```
typedef struct
{
    char *label; /* button label */
    void (*callback)(); /* callback function */
    caddr_t *data; /* client data for callback() */
} ActionAreaItem;
```

The **data* member of the structure is the array of Widgets that form the Control area.

The procedure of dialog box creation is standard.

- 1). Write the main function for each dialog box containing:
 - a). Definition of `ActionAreaItems` array
 - b). Control area widget structure
 - c). Control area widgets are gathered into the array which is pointed to by the **data* member of each `ActionAreaItem` element
 - d). `CreateActionArea` function call
 - e). Dialog box invocation.
- 2). Write callback functions for each Action area button.

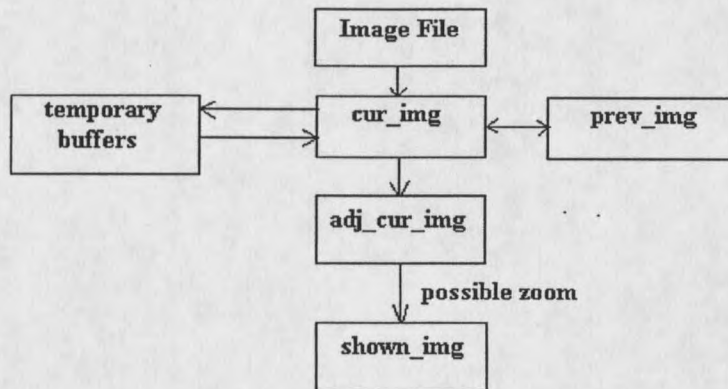
The above mentioned main dialog functions are in turn callback functions either for toolbar buttons, or pulldown menus' items.

Data Management and Display.

The structure of data buffers that exists throughout the application is shown on

Figure 4.

Figure 4. Memory Buffers Structure for GAP2.



"Real" Image Data.

Cur_img buffer contains the current image data, initially the one read from the image file. This data is changed throughout the program by applying resizes or image processing procedures. When any of those functions is applied:

- a). The copy of cur_img is sent to prev_img buffer
- b). This data might be copied to image processing functions' specific temporary working buffers
- c). The data is processed

- d). Processed data is sent back from those temporary buffers to cur_img.
- e). All intermediate buffers are freed.

If the Undo or Redo buttons are clicked, cur_img and prev_img buffers are swapped.

Color Adjustment and Displayed Image

The buffer shown_img contains the displayed image data, the data that is being used for creation of XImage structure that will be displayed. This data is obtained by "color adjustment" of the data contained in cur_img buffer and is affected by the zoom function.

What is color adjustment and why might it be needed? Unfortunately, despite the numerous advantages of the system that was used for development of this project, there was one serious disadvantage: the limit on the number of possible colors to be used simultaneously. This limit was caused by the graphics card installed and was equal to 256 colors. Many images used might have many less gray levels, but the range of the colors used might still be 256. The problem in this case is that the application can't use all available range of 256 colors because the system (Window manager) needs some of those colors for its own purposes: windows' decorations, background, etc. This is why some kind of tradeoff is inevitable.

The application is using the colormap (color palette) to display the images. When the colormap is being built, it uses color entries 40-255, leaving entries 0-39 for system purposes. The colormap creation procedure maps the range 0-255 to the range 40-255, where 40 is black, 255 is white, thus reducing the color range. The loss of 40 gray levels might not affect the display quality too much, but it might be significant for image processing operators which

are mainly based on pixel colors and pixel color differences. That's why two different data buffers are used for image display and processing.

Cur_img is the buffer being processed, it contains data of complete color range, as it was defined in the original file. Due to this fact, no data is lost for processing. This data is also used for display into the Pixel Intensity part of the Information panel, and for histogram calculation. Shown_img is adjusted image, which color range might be squashed to 216 gray levels. Adj_cur_img is the temporary buffer used for image adjustment. It is freed when image recalculation for display is over.

The color adjustment procedure is given in Table 4.

Table 4. Color Adjustment Procedure.

```

/*adjust color range of the image */
/*find min and max colors in the current image */
    for(I=0; i<256; I++)
    {
        if(H.count[i]>0)
        { mincol = I;
          break;}
    }
for(I=255; I>=0; I--)
{
    if(H.count[i]>0)
    { maxcol = I;
      break;}
}
img_range = maxcol-mincol; /* can be up tp 256 */
col_range = MAX_COLOR_INDEX-MIN_COLOR_INDEX; /* 215 */

/* creation of adjusted version of cur_img: template for shown_img*/
adj_cur_img = (unsigned char *) malloc(img_size*sizeof(char));
if(img_range>col_range)
{
    if(mincol>0) /*shift colors*/
    {
        for(I=0; i<img_size; I++)
            *(adj_cur_img+I) = *(cur_img+I)-mincol;
        maxcol-=mincol;
        mincol=0;
    }
    else /*just copy cur_img to intermediate buffer*/
        for(I=0; i<img_size; I++)
            *(adj_cur_img+I) = *(cur_img+I);

    /*find compression factor*/
    factor = (float)col_range/maxcol;

    /*compress the image plus shift it back*/
    for(I=0; i<img_size; I++)
        *(adj_cur_img+I)=
            (int)(ceil((float)*(adj_cur_img+I)*factor))+MIN_COLOR_INDEX;
}
else /*even if color range is OK, mincol may still be < 40: shift
      needed */
{
    if(mincol<MIN_COLOR_INDEX)
    {
        shift = MIN_COLOR_INDEX;
        for(I=0; i<img_size; I++)
            *(adj_cur_img+I) = *(cur_img+I)+shift;
    }
    else /* just use cur_img values, no adjustment is necessary*/
        for(I=0; i<img_size; I++)
            *(adj_cur_img+I) = *(cur_img+I)+mincol-MIN_COLOR_INDEX;
}
printf("adjustment done\n");

```

CHAPTER 3

SPATIAL DOMAIN OPERATIONS

"The term "spatial domain" refers to the aggregate of pixels composing an image, and spatial domain methods are procedures that operate directly on these pixels. Image processing functions in the spatial domain can be expressed as

$$g(x,y) = T[f(x,y)] \quad (1)$$

where $f(x,y)$ is the input image, $g(x,y)$ is the processed image, and T is an operator on f , defined over some neighborhood of (x,y) . " [3]

Median Smoothing.

The median smoothing algorithm belongs to the class of nonlinear image processing algorithms that sometimes are called "rank algorithms." These are fast algorithms based on local histograms of intensity distribution and their characteristics. This type of algorithms has several advantages in common with linear filtering algorithms. The main advantage is that these algorithms are adaptive since their parameters are functions of the local image histograms. At first glance it may seem that since rank algorithms reorder the data into the variational series, they don't use the spatial links between the elements of images, and this is their principal disadvantage. Amazingly though, this feature is actually an advantage. The spatial links between the image elements, defined by their belonging to the common detail, show themselves in the variational series through the parameters of the conditional histogram

of the signal values distribution in the neighborhood of the given element. This behavior of the spatial links does not depend on the details' orientation or configuration, thus eliminating the necessity of preliminary knowledge or measurement of details' orientation or configuration, that exists for optimal linear filters' design.

Figure 5. Median Smoothing: a). before, b). after.



The median smoothing procedure is represented by the formula:

$$MEAN(KNV(MED(M))), \quad (2)$$

where M is the spatial neighborhood of the current pixel, including it,

$MED(M)$ is the median of the neighborhood (in case of the neighborhood size 3×3 this is 5th largest element of M),

KNV is the "intensity neighborhood" of the current pixel, formed by the pixels that are closest to it by their intensity value. In my implementation, it is the set of 16 values, closest to the current one, chosen across the spatial neighborhood of size 7×7 around it. The closest values are searched among the brighter values as well as among the darker ones,

MEAN is an arithmetic average of the sample.

This procedure effectively removes small noise peaks in the image and almost doesn't blur it. Also, this particular realization of median smoothing processes the image completely, effecting even the boundary pixels. Median smoothing results are shown on Figure 5.

Edge Detection.

This rank algorithm employs another simple and popular technique of image processing: thresholding. The formula for edge detection is:

$$f(i) = M(R_i + e) - M(R_i - e), \quad (3)$$

where I denotes the current element of the image,

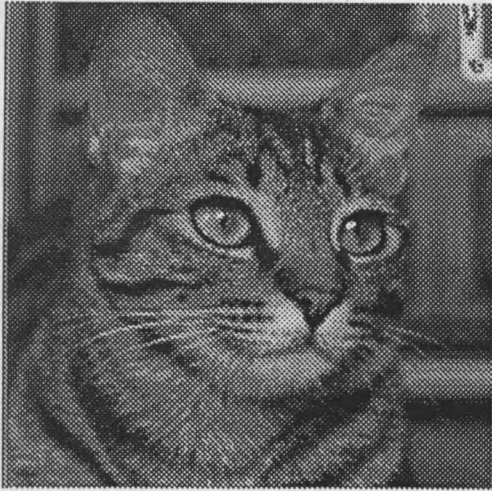
R_i is the rank (number) of the current element in the set of its 5x5 neighborhood members, sorted in ascending order; if there is more than one element of this value in the said set, the number of the first one is used,

e is the "radius" of the neighborhood of the current element in the sorted set; in the realization being discussed the value of 1 is used,

$M()$ is the value of the element of the set of the denoted rank,

$f(I)$ is the value to be compared with the threshold value (set by the user). If $f(I)$ is bigger than the threshold for some element, the current element is declared to belong to the edge and is colored white. Otherwise it is colored black. Though, in this case we would get the binary image, for which further preprocessing is limited. So, instead of coloring it black, its initial value is retained. Edge detection results are shown in Figure 6.

Figure 6. Edge Detection Results. The threshold value = 36.



a).



b).

Image Sharpening.

For spatial image sharpening, differentiation (gradient) is one of the most common methods. Given an image function $f(x, y)$, the gradient of f at coordinates (x, y) is the vector:

$$G[f(x, y)] = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]. \quad (4)$$

For the purposes of sharpening, the magnitude of the gradient is used:

$$g[f(x, y)] = \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{\frac{1}{2}}. \quad (5)$$

For an image, that is a discrete function, the derivatives in (5) are approximated by differences. The approximation used in the program is called Roberts Gradient [3]. It is defined on the 2x2 neighborhood, with the current pixel in the upper left:

$$g[f(x,y)] = |f(x,y) - f(x+1,y+1)| + |f(x,y) - f(x+1,y-1)|. \quad (6)$$

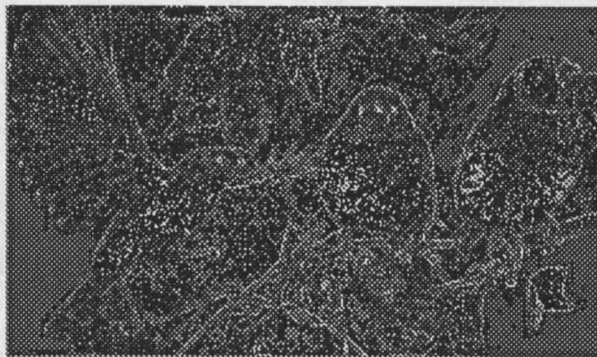
In order to preserve the smoothly colored regions from being displayed dark, a thresholding technique is used. The threshold value $T \geq 0$ is set by the user:

$$g(x,y) = \begin{cases} g[f(x,y)] & \text{if } g[f(x,y)] \geq T \\ f(x,y) & \text{otherwise} \end{cases} \quad (7)$$

Figure 7. Roberts Gradient Results. The threshold = 5.



a).



b).

Masking

Masking operations are discrete versions of convolution and are expressed by the general formula:

$$F(m_1, m_2) = \sum_{l_1=-L_1}^{L_1} \sum_{l_2=-L_2}^{L_2} S(m_1+l_1, m_2+l_2) H(L_1+l_1+1, L_2+l_2+1) \quad (8)$$

Masking operators are characterized by using relatively small neighborhoods of the element (m_1, m_2) , which usually do not exceed several percent or even fractions of percent of the image size. The most common sizes are 3x3 or 5x5. Masking operators are local and context free.

There exists many local masking operators of different types, often empirical or based on some models. For instance, in statistical local operators statistical characteristics of the neighborhood are taken into consideration: mean value, variance, etc., when the resulting elements $F(m_1, m_2)$ are being built. In this case image processing operations contain statistical characteristics as parameters (usually not higher than of the first order).

Several examples of masks are given in Table 5.

(A) is a noise reduction mask. This is just 3x3 neighborhood averaging,

(B) is an edge enhancement mask,

(C) is an edge and line enhancement mask,

(D) is the mask for enhancement of gradients,

(E) serves for obtaining pseudostereoscopic effects,

(F) enhances gradients with simultaneous noise reduction, this being a combination of (A) and (D).

Table 5. Commonly Used Image Processing Masks.

$$H = \begin{bmatrix} 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \end{bmatrix} \quad H = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad H = \begin{bmatrix} -1 & 2 & -1 \\ 2 & -3.8 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

(A)

(B)

(C)

$$H = \begin{bmatrix} -1 & -1 & 1 \\ -1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad H = \begin{bmatrix} -1 & 0 & -2 \\ 0 & 0 & 0 \\ 2 & 0 & 1 \end{bmatrix} \quad H = \begin{bmatrix} -0.11 & -0.11 & 0.11 \\ -0.11 & -0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \end{bmatrix}$$

(D)

(E)

(F)

Figure 8. Masking Dialog Box.



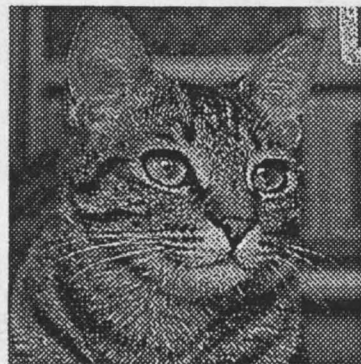
Figure 9. Masking: a). is the initial image, b). mask (A), c). mask (B), d). mask (C), e). mask (D), f). mask (E), g). mask (F).



a).



b).



c).



d).



e).



f).



g).

Masking operators are very popular due to ease of design and calculation simplicity and speed. But these operators do not improve the image quality significantly.

The masking function implemented in GAP2 package allows the user to input and apply masks of size 3x3. The dialog box is shown in Figure 8. The results of the masks' application is shown in Figure 9.

Histogram Equalization.

This fast and efficient operator allows to improve the quality of images with narrow color range significantly, stretching this range to the whole width of 255 levels. The number of gray levels remains the same, but dramatic range stretching lets one see many details that were previously hidden.

Figure 10. Histogram Equalization.



a).

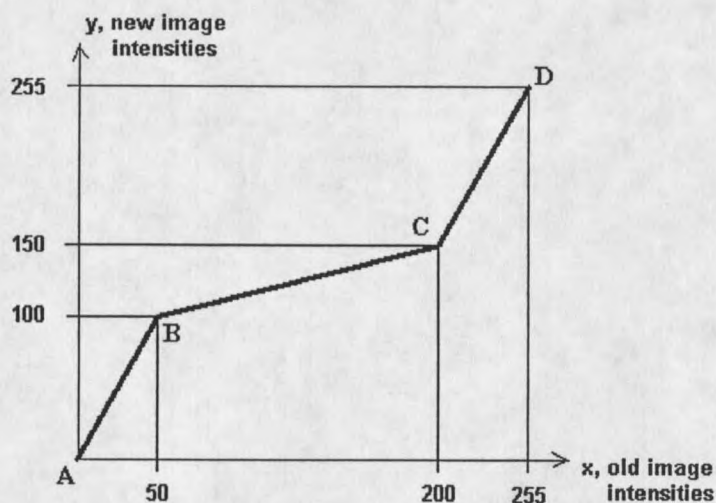


b).

Contrast Stretch

Contrast stretch is a more "general" version of histogram equalization that lets the user to "edit" the histogram locally, stretching some color ranges and shrinking others. The basic principle of contrast stretch is mapping the old image's color distribution to that of the new image. Figure 11 illustrates this technique.

Figure 11. Contrast Stretch Graph

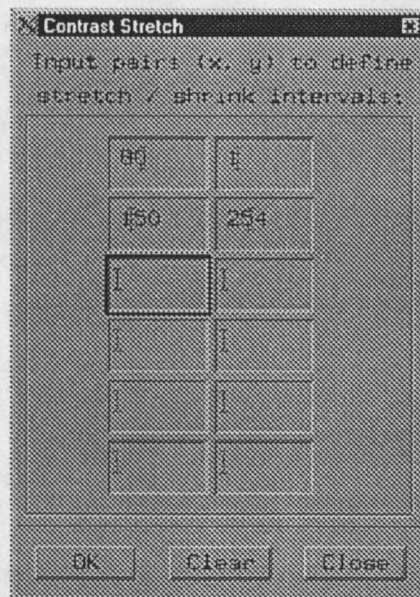


Suppose that both old and new images will be of full range of 255 gray levels. In order to obtain the new color range density for the new image, the piecewise-linear function of the old image's gray levels must be specified by defining points A, B, C and D (the number of points may vary). The function shown on Figure 11 stretches the 0-50 color range of the old

image into 0-100 for the new one, thus "equalizing" the darker part of it (A-B piece). The piece B-C shrinks 150 gray levels between 50 and 200 into 50 gray levels between 100 and 150. The last piece, C-D, stretches the remaining 55 gray levels between 200 and 255 into 105 gray levels between 150 and 255.

This operator is useful for the equalization of images that can't be equalized by the ordinary equalization operator. For instance, if the image has some elements of color 0, some of color 255 (or close to those), and the rest of it is in the range, say, 80-150. In this case, it is possible to set the following points: A(0, 0), B(80, 1), C(150, 254), D(255, 255). Contrast stretch operator with these parameters will result in a uniform histogram.

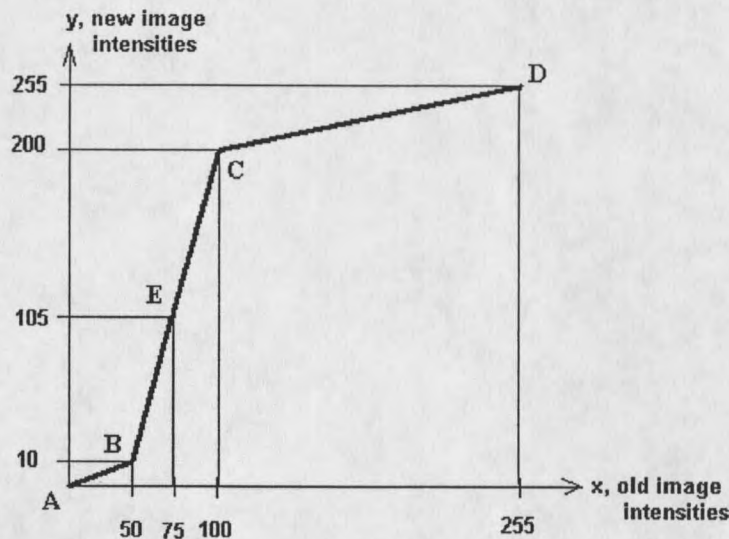
Figure 12. Contrast Stretch Dialog Box



Two versions of the contrast stretch operator were implemented in GAP2. The first is the above described one. The user can specify up to 6 points to define the piecewise-linear

function. The points (0, 0) and (255, 255) are assumed by the system as the first and the last one, so these do not need to be specified. The order of the points' input does not matter: the program will sort them in ascending order. But, these points must define a nondescending function (if not, an error message is given and the point set must be redefined). The dialog box for this procedure is shown in Figure 12.

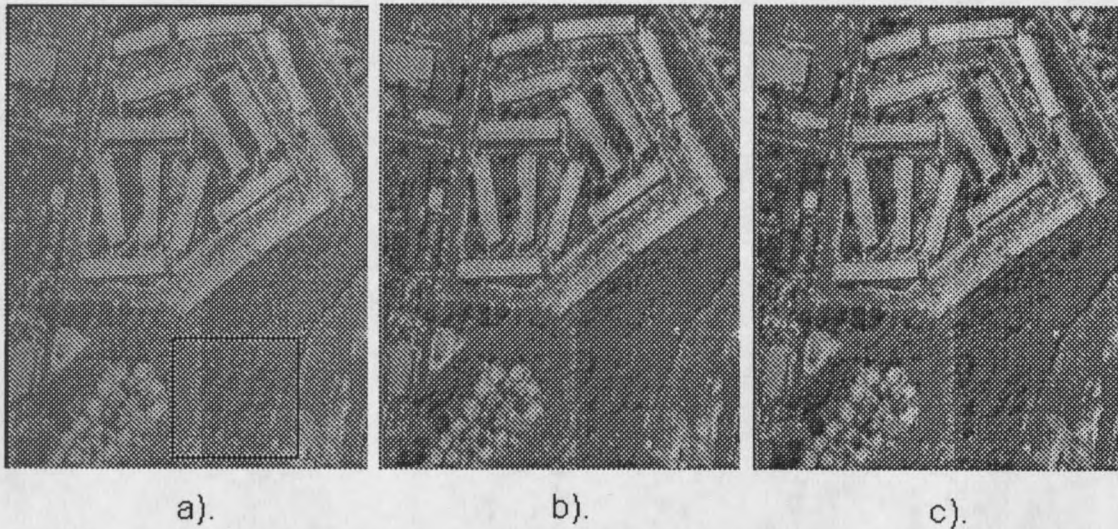
Figure 13. Area Contrast Stretch Graph



The second version is Area contrast stretch. In this case, only 4 points are used. The first and the last ones are as usual, (0, 0) and (255, 255), and the two intermediate ones are minimum and maximum intensities of some part of the image. The user defines the image area to get the minimum and maximum intensity values from, and the number of gray levels desired between those intensities on the new image. It is illustrated in Figure 13. The image region

is defined using the same rubberbanding technique as for cropping. The number of gray levels is set by the user through the simple dialog box. On the Figure 13: values 50 and 100 on the X axis were derived from the specified image part. The user only defined the new range (190 gray levels), and the values for the new image were calculated (10 and 200 on the Y axis). That is, the range 50-100 was symmetrically stretched relatively to the center point E. Contrast stretch results are shown in Figure 14.

Figure 14. Contrast Stretch: a). initial image with "reference area" for area contrast stretch, b). contrast stretch with points (60, 15) and (180, 210), c). area contrast stretch with parameter 200.



Binarization.

This is a simple thresholding procedure used to get the final contour binary image. The threshold value is set by the user.

$$f(x,y) = \begin{cases} 255, & \text{if } f(x,y) \geq T \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

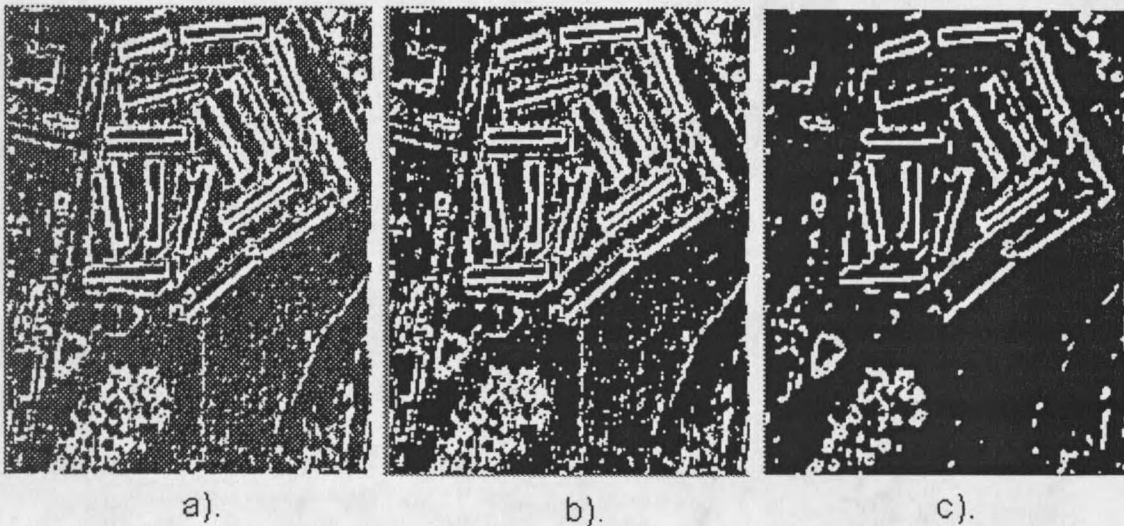
Thinner.

This procedure is applied to the binary image in order to obtain the thinner edges and to remove noise. The procedure uses a neighborhood of the 3x3 and works as follows:

- 1). If the current pixel is black, move on to the next pixel.
- 2). If the current pixel is white, we count the number of black ones around it. If this number exceeds some threshold value (5 in GAP2), this pixel is colored black, otherwise we leave it white.

This algorithm is not very fine, but gives satisfactory results, is simple and can be applied several times if necessary.

Figure 15. Binarization and Thinner: a). the image after edge detection, b). Corresponding binarized image with the threshold 80, c). previous image after 2 applications of thinner.



CHAPTER 4

FREQUENCY DOMAIN

Any image can be assumed as a spatial function of pixel intensity. Any function, including images, can be represented as a superposition of some other functions. Most often that set of decomposition functions forms an orthonormal basis. A big number of different transforms exist, on different bases. Since images are discrete functions, discrete versions of transforms are used. Three transforms were implemented in GAP2: Fourier transform, Walsh transform and Wavelet transform.

The Fourier transform module was originally written by Dr. Gary Harkin (Computer Science Department, MSU), and the Wavelet transform module was originally written by Ting-Chun Janet Liu (Department of Computer Science, Cornell University). Both modules were modified and adapted for use with GAP2. I would like to express my sincere appreciation to both authors for making their work publicly available.

Fourier Transform.

The Fourier transform is the most popular one used for image processing applications. It has a lot of implementations that were successfully used for years. Forward and inverse discrete Fourier transforms are represented by the pair of formulas [3]:

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \exp \left[\frac{-2\pi j(ux+vy)}{N} \right] \quad (10)$$

$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u, v) \exp \left[\frac{2\pi j(ux+vy)}{N} \right] \quad (11)$$

for $N \times N$ image $f(x, y)$.

The procedure implemented in the project is the well-known Fast Fourier Transform algorithm published by Cooley et al. in 1969. This algorithm employs a "successive doubling" method which means that the size of the input image must be the power of 2. The detailed discussion of the theoretical foundations of FFT is beyond the scope of this paper. Computation of FFT in GAP2 has the structure given in Table 6.

Table 6. Fast Fourier Transform Procedure.

```
fft_image /* forward transform */
{
/* Image size adjusting block. It makes the image to be the size of
the power of 2 by filling the extra space with copies of the last
column and the last row of the image. */
fft_cshift; /* centers the image data in the new window of
size NxN */
fft_2d /* forward transform of adjusted image */
{
fft_swapmap; /* building of the image pixel indices' swapping
lookup table */
fft_settrig; /* building of the trigonometry lookup table */
/* the separability property of Fourier transform allows to
calculate 2D FFT as a series of 1D FFT, applied to rows of the
image, then to its columns (or vice versa) */
fft (rows);
/* transpose the image */
fft (columns);
/* transpose back */
}
}
```

fft_swapmap and *fft_settrig* create lookout tables of array index reordering for the successive doubling and trigonometric coefficients values for later use by the *fft* function, which actually performs the fast Fourier transform of data.

Inverse transform uses the same set of functions, but they are applied in reverse order:

- a). *fft_2d*
- b). *fft_cshift*
- c). restoring of the original image size.

Viewing Fourier Spectrum

The Fourier transform yields the complex results, the transformed data can be presented in the form [3]:

$$F(u) = R(u) + jI(u) \quad (12)$$

for the one-dimensional function. The magnitude of this function:

$$|F(u)| = [R^2(u) + I^2(u)]^{\frac{1}{2}} \quad (13)$$

is called the Fourier spectrum of the image.

This function usually tends to decrease rapidly with increasing value of u . Thus, the high frequency components of the spectrum might not be displayed when the spectrum is visualized and it is recommended to display the function

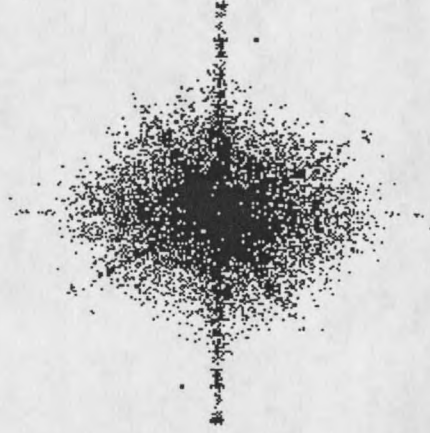
$$D(u) = \log(1 + |F(u)|). \quad (14)$$

This function "bumps" the nonzero values for the spectrum while preserving the zero values [3]. Fourier spectrum output from GAP2 is shown in Figure 16.

Figure 16. Fourier Spectrum: a). the image and b). its Fourier spectrum.



a).



b).

Butterworth Filtering

Image filtering techniques in the frequency domain are based on the convolution theorem which states that the convolution operation in the spatial domain reduces to multiplication in the frequency domain and vice versa. Thus, if we have two functions: $f(x)$ and $g(x)$ in the spatial domain and their corresponding Fourier transforms $F(x)$ and $G(x)$, the convolution theorem results are as follows:

$$f(x) * g(x) \Leftrightarrow F(u)G(u) \quad (15)$$

$$f(x)g(x) \Leftrightarrow F(u) * G(u) \quad (16)$$

Since filtering in the spatial domain is defined by the convolution between the image and the filter functions, in the frequency domain we use multiplication of the transformed image function and the transfer function of the filter:

$$g(x,y) = h(x,y) * f(x,y) \Leftrightarrow G(u,v) = H(u,v)F(u,v) \quad (17)$$

To obtain the filtered image, we must take the inverse Fourier transform of $G(u, v)$:

$$g(x, y) = F^{-1}(H(u, v)F(u, v)) \quad (18)$$

The image's sharp edges and noise contribute to the high frequency part of the spectrum, while uniformly colored areas add to the low frequency part. The bigger the value of (u, v) coordinates of the spectrum image are, to the higher frequencies of the initial image does this part of the spectrum correspond. Thus, if we apply the function that attenuates the central part of the spectrum, the restored image will have greatly enhanced high frequency components (sharpening). If we attenuate the peripheral parts of the spectrum, blurring (smoothing) of the initial image will be achieved.

The pair of transform functions known as the Butterworth filter can be used for these purposes [3]:

$$H(u, v) = \frac{1}{1 + \left[\frac{D(u, v)}{D_0} \right]^{2n}} \quad (19)$$

for lowpass filtering (BLPF), and

$$H(u, v) = \frac{1}{1 + \left[\frac{D_0}{D(u, v)} \right]^{2n}} \quad (20)$$

for highpass filtering (BHPF), where

$$D(u, v) = (u^2 + v^2)^{\frac{1}{2}}$$

is the distance of the current pixel from the spectrum plane origin,

D_0 is the cut-off frequency locus,

n is the order of the filter.

Unlike the ideal filter, the Butterworth filter does not cut off the frequencies sharply, thus allowing smoother filtering results. For this type of filter, the cut-off frequency locus designates the points for which the transfer function will have a certain fraction of its maximum value (the value range of Butterworth filters varies between 0 and 1). Thus, for the given functions, the value of D_0 defines the set of points for which 50% attenuation of the frequency will be obtained. Another value that is often used is $\frac{1}{\sqrt{2}}$ of the $H(u, v)$'s maximum value. For this, the formula (19) (and in the same way, (20)) must be modified:

$$H(u, v) = \frac{1}{1 + 0.414 \left[\frac{D(u, v)}{D_0} \right]^{2n}} \quad (21)$$

This type of filtering was implemented in GAP2 where the user defines all parameters using the dialog box shown on Figure 17. Filtering results are shown on Figure 18.

Figure 17. Butterworth Filter Dialog Box.

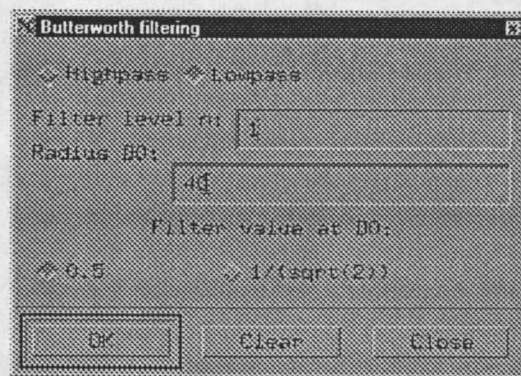
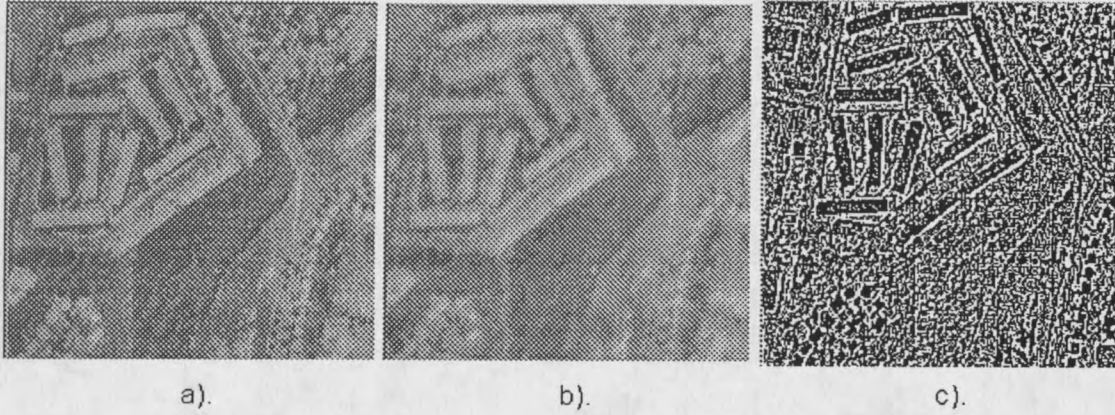


Figure 18. Butterworth Filtering: a). initial image, b). Butterworth lowpass, c). Butterworth highpass.



Walsh Transform

The other linear reversible transform on the orthogonal basis is the Walsh transform. It is defined by the square matrix H that is built in the basis of the set of Walsh functions. These functions are the full set of periodic orthonormalized functions that can have the value of 1 or -1 and are defined on the interval $[0, 2^p]$, where $p > 0$:

$$\varphi_n(x) = (-1)^{\sum_{i=0}^{p-1} n^{(p-1-i)} x^{(i)}}, \quad (22)$$

where $n^{(i)}$ and $x^{(i)}$ can be found from binary decompositions of n and x :

$$n = \sum_{i=0}^{p-1} n^{(i)} 2^{p-1-i}, \quad x = \sum_{i=0}^{\infty} n^{(i)} 2^{p-1-i} \quad (23)$$

The Walsh transform is built on the basis of the simple matrix:

$$H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (24)$$

Then, the matrix of the order $2N = 2^m$, where m is integer, will be:

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix} \quad (25)$$

This matrix is symmetrical and orthogonal.

Thus, after normalizing, the forward and inverse transforms will be:

$$[C] = [H][F][H]$$

$$[F] = \frac{1}{N^2} [H][C][H] \quad (26)$$

The Walsh transform is separable too, so it is applied to the image in the same way as the Fourier transform: first, the rows are processed by the 1D transform, then the image is transposed and the columns are processed, followed by another transpose.

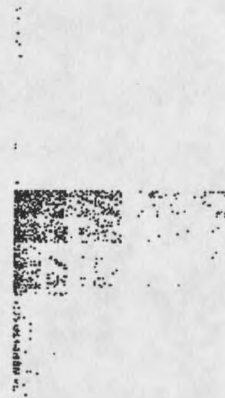
The Walsh transform can be computed by a fast successive-doubling algorithm similar to FFT. The difference is that all operations are real and trigonometric terms are set to 1. This is the way the Walsh transform is realized in GAP2. Due to this fact, the Walsh transform can be computed approximately 10 times faster than FFT, since only additions and subtractions of integers are employed.

Walsh spectrum viewing is also possible. For the filtering of the Walsh transformed image the same Butterworth filter function is used. Walsh spectrum and filtering results are shown in Figure 19.

Figure 19. Walsh Transform: a). initial image, b). Walsh spectrum, c). lowpass filtering, d). highpass filtering.



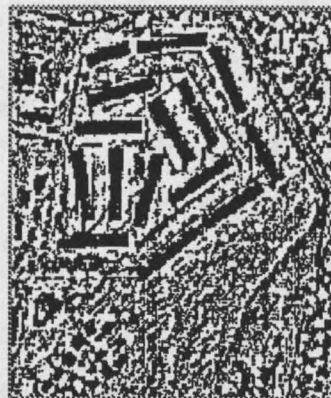
a).



b).



c).



d).

Wavelet Transform

Wavelets are a recent field of study that lately drew the interest of scientists and engineers from different backgrounds. Wavelet transform became a cutting edge technology in image and audio data processing. "The Wavelet transform is a tool that cuts up data or functions ... into different frequency components, and then studies each component with a

resolution matched to its scale". [5] The most common application of Wavelet transform seems to be image compression. Numerous algorithms were worked out where Wavelet transform is the basis [4], [6], [7]. Image filtering techniques also exist. One of them, described in [8], was implemented in GAP2 and will be described below.

General Overview of Wavelet Transform

Wavelets are functions generated from one single function ψ by dilations and translations:

$$\psi^{a,b}(t) = |a|^{-\frac{1}{2}} \psi\left(\frac{t-b}{a}\right). \quad (27)$$

ψ is called the mother wavelet and has to satisfy the condition:

$$\int \psi(x) dx = 0, \quad (28)$$

which implies that ψ has some oscillations. The definition of wavelets as dilations of one function means that high frequency wavelets correspond to $a < 1$, or narrow width, while low frequency ones have $a > 1$, or higher width.

The basic idea of the wavelet transform is the representation of any arbitrary function f as a superposition of wavelets (as for any frequency domain transform). Any such superposition decomposes f into different scale levels, where each level is then further decomposed with a resolution adapted to the level.

In a multiresolution analysis, there are two functions: the mother wavelet ψ and the scaling function ϕ . The scaling function is dilated and translated:

$$\phi_{m,n}(x) = 2^{-\frac{m}{2}} \phi(2^{-m}x - n). \quad (29)$$

See [9] for more information on multiresolution analysis and wavelets.

For fixed m , $\phi_{m,n}$ are orthonormal. Thus, both ψ and ϕ are orthonormal bases. A lot of different orthonormal wavelet bases were built. For this implementation, Daubechie's wavelet was used.

Wavelet Transform Implementation

In the present implementation, the Wavelet transform of the image is obtained using Quadrature Mirror Filters (QMF) [6]. These filters are built using H and G filters as a basis. These filters are used to decompose the source image into four subband images (after one transform pass). These are high pass subbands GG (diagonal high frequency image data), GH (horizontal high frequency data), HG (vertical high frequency data), and lowpass subband HH. The decomposition scheme and the decomposed image after 2 passes of the Wavelet transform are shown on the Figures 20 and 21.

The relationship between H and G filters is:

$$G(n) = (-1)^n H(1-n) \quad (30)$$

The forward transform uses filters H_bar and G_bar , the inverse transform employs H and G. The relationship between those is:

$$G(n) = G_bar(-n), \quad H(n) = H_bar(-n) \quad (31)$$

Daubechie's 6 taps wavelet is used for the GAP2 module. The filter coefficients for this type of wavelet are:

$$\begin{aligned} h(0) &= 0.332670552950 & h(1) &= 0.806891509311 \\ h(2) &= 0.459877502118 & h(3) &= -0.135011020010 \\ h(4) &= -0.085441273882 & h(5) &= 0.035226291882 \end{aligned} \quad (32)$$

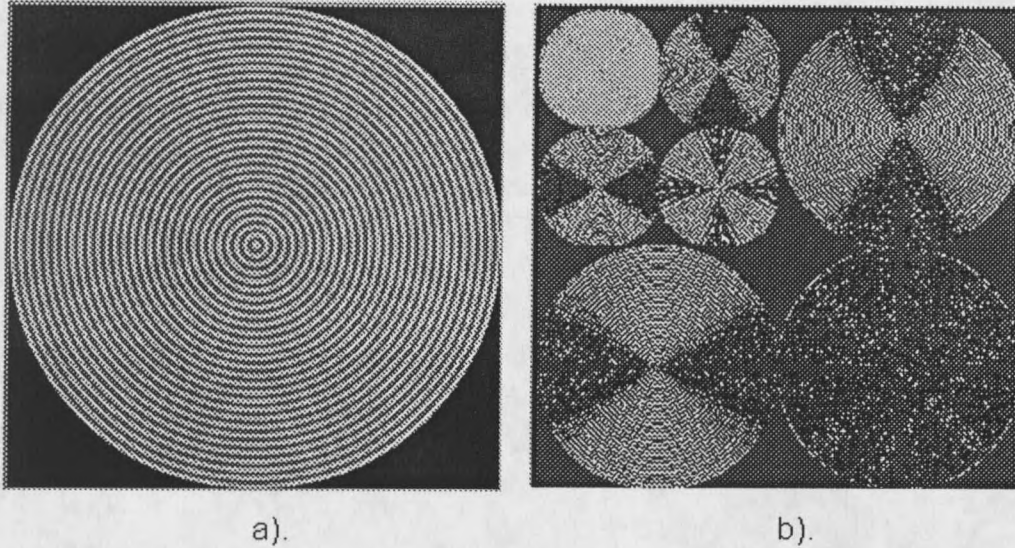
Figure 20. The Structure of Wavelet Decomposed Image for the Level 2 Transform.

HH2 1/8 resolution sub-image	GH1 1/4 resolution horizontal	GHD half-resolution horizontal orientation
HG1 1/4 resolution vertical	GG1 1/4 resolution diagonal	
HG0 half-resolution vertical orientation		GG0 half-resolution diagonal orientation

Two-dimensional Wavelet transform is implemented by two-pass one-dimensional transform: once for the rows and once for the columns. To obtain the result, one-dimensional data is convolved with the filter coefficients. After that the image is downsampled by 2. When four first subband images are ready, the next level of transform is applied to the HH image.

The inverse transform employs the same operations in reverse order plus adds upsampling.

Figure 21. Wavelet Decomposition: a). initial image, b). decomposed image (2 levels).



Wavelet Filtering

The way the Wavelet transform decomposes the image can be used for effective image filtering (noise reduction), and edge detection. The noise reduction algorithm was proposed in [8]. This algorithm uses "the direct multiplication of Wavelet transform data at adjacent scales to distinguish important edges from noise (both edges and noise are in the high frequency part of the spectrum), and accomplish the task of removing the noise from signals" [8]. The main principle of this algorithm is the fact that sharp edges are preserved over many wavelet scales, while the noise dies out fast as the scale number increases. The direct spatial correlation (33) of the Wavelet transform contents at the several adjacent scales is used to accurately detect the locations of the edges (1D data, for each pixel):

$$Corr_l(m, n) = \prod_{i=0}^{l-1} W(m+i, n), \quad (33)$$

where W is the Wavelet transform,

m is the starting scale index,

$n = 1, 2, \dots, N$,

N is the size of the image,

l is the number of scales involved in the direct multiplication.

Thus, for all pixels n ,

$$Corr_2(1, n) = W(1, n)W(2, n) \quad (34)$$

The smaller sized scale is resized to fit the larger one. The overall procedure can be described as a comparison of the normalized correlation data $|Corr_2(m, n)|$ with $|W(m, n)|$ to extract the edge information from $W(m, n)$ iteratively at the m -th wavelet scale. By repeating the procedure for all resolution scales, we acquire spatially filtered wavelet transform data to be reconstructed by the inverse transform. The absence of edges in a local region of the image allows the noise to be removed there. The filtering scheme in pseudocode is given in Table 7 (from [8], modified).

Figure 22. Wavelet Filtering Results: a). initial image, b). filtered image.



a).



b).

Table 7. The Wavelet Based Noise Reduction Filter

First, save a copy of $W(m, n)$ to $WW(m, n)$.
 Initialize the "spatial filter mask" mask (m, n) to zeros.

Loop for each wavelet scale m
 {
 Compute the power of $\text{Corr2}(m, n)$ and $W(m, n)$:

$$P_{\text{corr}}(m) = \sum_n \text{Corr2}(m, n)^2$$

$$PW(m) = \sum_n W(m, n)^2$$

 Rescale the power of $\text{Corr2}(m, n)$ to that of $W(m, n)$:
 Loop for each pixel point n :
 {
 New $\text{Corr2}(m, n) = \text{Corr2}(m, n) * \sqrt{\frac{PW(m)}{P_{\text{Corr}}(m)}}$
 }
 } end loop n
 Loop for each pixel point n
 {
 Compare pixel values in new $\text{Corr2}(m, n)$ and $\text{Corr2}(m, n)$:
 if $|\text{Corr2}(m, n)| > |W(m, n)|$
 {
 Extract edge information from $W(m, n)$ and $\text{Corr2}(m, n)$ and
 save it in the "spatial filter mask":
 $\text{Corr2}(m, n) = 0.0$; $W(m, n) = 0.0$;
 $\text{mask}(m, n) = 1$;
 }
 }
 }
 Apply the "spatial filter mask" to the saved copy, $WW(m, n)$ at
 the scale m , save the filtered data to $W_{\text{new}}(m, n)$:
 Loop for each pixel point n
 {
 $W_{\text{new}}(m, n) = \text{mask}(m, n) * WW(m, n)$;
 }
 } end loop n
 } end loop m

Performance Comparison.

The performance of all three transforms was compared using a 512x512 test image.

The results given in Table 8 are average ones. The actual figures differed greatly due to the

current system load. Time is given in seconds, running on DEC Alpha.

For Butterworth filters' tests: the level used was 1, the cut-off locus was 100, and the cut-off coefficient was 0.5.

For Wavelet transform tests: the transform level was 2, the Wavelet filter incorporated forward transform, filtering and inverse transform.

Table 8. Frequency Domain Transforms Performance Comparison.

Transform	Forward transform	Spectrum Calculation	Butterworth Lowpass	Butterworth Highpass	Wavelet filter
Walsh	0.6	0.9	1.4	1.7	N/A
Fourier	1.1	1.3	2.2	2.0	N/A
Wavelet	2.0	N/A	N/A	N/A	4.2

REFERENCES

- [1] Dan Heller, "Motif Programming Manual" O'Reilly & Associates, 1992
- [2] Nabajyoti Barkakati, "X Window System Programming" SAMS, 1992
- [3] Rafael C. Gonzalez, Paul Wintz, "Digital Image Processing" Addison-Wesley, 1987
- [4] Marc Antonini, Michel Barlaud, "Image Coding Using Wavelet Transform" IEEE Transactions on Image Processing, Vol. 1, No. 2, April 1992, pp. 205-220
- [5] I. Daubechies, "Ten Lectures on Wavelets" Society for Industrial and Applied Mathematics, 1995
- [6] A. Lewis, G. Knowles, "Image Compression Using the 2-D Wavelet" IEEE Transactions On Image Processing, Vol. 1, No. 2, 1992, pp. 244-250
- [7] A. Averbuch, D. Lazar, M. Israeli, "Image Compression Using Wavelet Transform and Multiresolution Decomposition" IEEE Transactions on Image Processing, Vol. 5, No. 1, 1996, pp. 4-15
- [8] Yansun Xu, J. B. Weaver, D. M. Healy, Jian Lu, "Wavelet Transform Domain Filters: a Spatially Selective Noise Filtration Technique" IEEE Transactions on Image Processing, Vol. 3, No. 6, 1994, pp. 747-758
- [9] S. Mallat, "A Theory for Multiresolution Signal Decomposition: the Wavelet Representation" IEEE Trans. Pattern Anal. Mach. Intell., Vol. 11, pp. 674-693, 1989
- [10] Charles A. Poynton, "Technical Introduction to Digital Video" Wiley & Sons, 1995

MONTANA STATE UNIVERSITY LIBRARIES



3 1762 10314211 1