Fall 1993

Design and implementation of IPIS : an X-Window based image processing interactive system

Eduardo Morales
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

INFORMATION TO USERS

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

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

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

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 1356273

Design and implementation of IPIS: An X-window based image processing interactive system

Morales, Eduardo, M.S.

New Jersey Institute of Technology, 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

ABSTRACT

Design and Implementation of IPIS: an X-Window Based Image Processing Interactive System

by
Eduardo Morales

Most of image processing systems are based on command line functions or can only display one image at a time. This is a serious inconvenience for those who need an interactive system session or want to compare two images processed by different techniques at the same time.

The system was designed with these problems in mind. It is able to display the processed image right after an operation and to display several images simultaneously, making it simple to compare techniques. The system was also created with the purpose to be used in an academic environment. Its structured design makes it easy to understand and to aggregate new functions and features. Used properly it may be a valuable learning tool for the areas of image processing and X/Motif programming.

Future work will expand the system in order to process color and multispectral images. An object oriented approach is being considered to achieve such goal.

DESIGN AND IMPLEMENTATION OF IPIS: AN X-WINDOW BASED IMAGE PROCESSING INTERACTIVE SYSTEM

by
Eduardo Morales

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

January 1994

APPROVAL PAGE

DESIGN AND IMPLEMENTATION OF IPIS: AN X-WINDOW BASED IMAGE PROCESSING INTERACTIVE SYSTEM

Eduardo Morales

Dr. Frank Y. Shih, Thesis Adviser — Date
Associate Professor of Computer and Information Science, NJIT

Dr. Peter A. Ng, Committee Member — Date
Professor of Computer and Information Science and
Chairperson of the Department of Computer and
Information Science, NJIT

Dr. James A. McHugh, Committee Member — Date
Professor of Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author: Eduardo Morales

Degree: Master of Science in Computer Science

Date: January 1994

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1994
- Bachelor of Science in Computer Science,
  Indiana University of Pennsylvania, Indiana, PA, 1991

Major: Computer Science

This thesis is dedicated to
my parents, Willy and Ladbye,
and to my sister, Lorena.

ACKNOWLEDGMENT

I would like to thank Dr. Frank Shih for his guidance, support, and specially for introducing me into the field of image processing.

I would like to thank to all the professors from whom I had the opportunity to learn invaluable knowledge.

I also would like to thank to Richard T. Tracy and his parents for providing me a home away from my country.

Last but not least, I would like to thank my parents for providing me moral and economic support. It is needless to say that I owe them everything that I have achieved in life.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

OPERATING ENVIRONMENT

The behavior and performance of the Image Processing Interactive System are directly related to its operating environment. It requires of certain needs in order to operate appropriately. Those needs can be grouped in four areas: operating system, X Windows, OSF/Motif toolkit and hardware.

1.1 Operating System

IPIS was developed on SunOs 4.1.2. It is generally intended to be used with UNIX systems. Although it has not been tested in other platforms, the program was written in ANSI C to make it as portable as possible. To further increase its portability to other flavors of UNIX, and even to other operating systems supporting X Windows such as VMS, no system calls have been used in the application. However, if ported outside of UNIX, the nonstandard libraries used by the program, such as libmorph, libtiff and libjpeg, will have to be ported too. Porting these libraries may not be an easy task, luckily there may be a port in the public domain. Nevertheless, with minor changes, the system can still perform adequately without those libraries. The ability to read/write JPEG and TIFF files will be lost as well as most—but not all—morphology routines. Anything else should work as planned.

The system requires only one environment variable. This variable is named IPT_HELP_PATH and contains the directory name where the help

files reside. Failure to define this variable will only restrict the ability to access the on-line help in the program.

1.2 X Windows

The system uses X Windows to display its several capabilities. The whole application is window oriented—use of a terminal screen may only be needed to create test images in ASCII mode as well as structuring elements for mathematical morphology—and therefore takes full advantage of the features provided by the window system.

One of the features is network transparency. As all X Windows applications, IPIS is able to run on a remote machine and display the results on the local monitor. This feature enables us to take full advantage of hardware configurations to improve the performance of the IP system. For example, a personal computer running an X server will be able to increase the performance of IPIS by running it on a more powerful remote computer, such as a SUN SPARCstation and display the results locally. Academic environments can enormously benefit from this feature. Students can run the software from their homes, just by using a PC with an X server and a SLIP or PPP connection to the academic computing facilities at their respective schools.

The system can display multiple windows, and that perhaps is its biggest advantage. This feature allows us to be able to compare simultaneously several images that have been processed by different techniques. Also dialogs and menus are displayed through windows to provide ease of use. Windows also enable us to make better use of screen space since they can be overlaid or they can be iconified.

The application is highly configurable. X applications can alter their default values by reading a file called the resource file. This is only possible if the default resources have not been hardcoded in the application. In IPIS, little resource information has been hardcoded, allowing the behavior and appearance to be modified by the user at will. Among the resources that the user can customize are the foreground and background colors, the window geometry, the widget dimensions, the label strings, etc. Perhaps the most useful of the configurable resources may be the label strings. Almost all labels in the application can be set through a resource file. This may prove to be very helpful when porting the program into an environment where the native language is not English. Just by changing labels for buttons, dialogs and window titles, we have an application in the end user's native language.

For more information on X Windows refer to Barkakati (1) or any other X Windows book.

The application has been successfully tested in X11R4 and X11R5.

1.3 OSF/Motif Toolkit

The OSF/Motif toolkit—Motif for short—is responsible for the look and feel of the application. The program was developed using Motif 1.1, and adheres to the OSF/Motif Style Guide in order to be consistent with other Motif applications.

The decision to implement the application using Motif over OPEN LOOK, or any other toolkit, was based in the fact that Motif has become the industry standard.

The application should work with any window manager without any major problem. Nevertheless, it will perform better with the Motif Window Manager (MWM) since it is able to process a close window function from the MWM. The only other window manager tested is TWM, which performed normally.

1.4 Hardware

The IP system was designed with the objectives of processing 256 gray scale images in the present and of processing color images in the future. Keeping this in mind, it was decided not to support 1-bit and 4-bit displays since they do not provide enough colormap entries to adequately represent the targeted images on screen. The decision is also supported by the fact that computer displays with depth of less than 8-bits are less common nowadays and eventually will be phase out.

Hardware, with the exception mentioned above, should not effect the IP system in any way other than computational speed. This property can be attributed to the fact that X Windows hides hardware details, such us little-endian and big-endian CPUs, under the X protocol.

CHAPTER 2

SYSTEM OVERVIEW

IPIS was designed with the purpose to process gray level images at first and to aggregate color capabilities at a later date. Currently it only processes gray level images.

It is intended to be a comprehensive tool, covering all the fields in image processing. An effort to include most of the IP techniques has been attempted, but due to its large number some remain to be implemented. Later versions will include missing techniques and as well as new ones.

Another of its goals is to be an instructional tool. The IP system has a structured form which makes it very easy to upgrade and modify. Students can learn about different techniques by inspecting the code and can enhance the system by either implementing new techniques not supplied by the system or improving the ones that are supplied.

It was also intended to be interactive. Results of techniques are displayed soon after they are applied with its response time depending on the computational complexity of the technique. The system is able to display several images at the same time, making it possible to compare images processed by different methods.

2.1 The Main Window

The principal interface to the program is the main window. It consists of three areas: the menubar, the message area and the action area. Figure 2.1 shows the window.

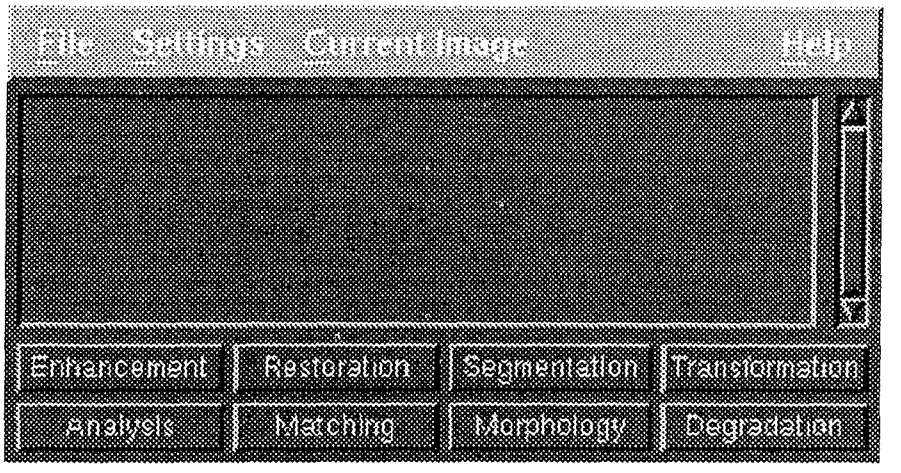

Figure 2.1 The Main Window.

2.1.1 The Menubar

In the menubar are located four pulldown menus. The first one is used to call the read and write dialogs, and to exit the program. The second menu is used to set some global attributes that will affect the behavior of the program. With the first option, we can control how pixels with values less than 0 and greater than 255 will be processed. Possible choices are to scale them so that the smallest value is mapped to 0, the largest is mapped to 255, and all remaining values are linearly mapped in between. Another option is to truncate the pixel values. A pixel less than 0 or greater than 255 will be truncated to 0 or 255 respectively. The last option is do nothing. In this case, the value will be derived from the assignment of the integer holding the exceeding value to the unsigned char (pixel value). This assigned value will be the one in the least significant byte of the integer. The last option is faster than the previous two and is useful when it is known that all resulting pixel values will be within bounds. Examples:

Original 7x1 image:	[0 100 170 255 170 100 0]
Filtered image with [-1 3 -1]:	[-100 130 155 425 155 130 -100]
Scaled filtered image:	[0 112 124 255 124 112 0]
Truncated filtered image:	[0 130 155 255 155 130 0]
Nothing done:	[156 130 155 169 155 130 156]

The second and last option in the same pulldown menu lets the application know which colormap to use when allocating images. There are two possible choices: the standard colormap and a private colormap. The advantage of using a private colormap is that all of the gray levels can be allocated, so 256 gray level images can be accurately displayed on the screen. The disadvantage is that the X server can only use one colormap at a time; so when an image window with a private colormap gains input focus, all other applications running with the X server will lose its colors and display gray levels instead. This colormap mode despite being very useful, it turns to be quite annoying. On the other hand, using the standard colormap does not cause this kind of side effect, but the application has to share the colormap with other applications running with the X server. This means that it will not be possible to allocate the 256 gray levels. Refer to section 2.4 to see how is this problem handled by IPIS.

The third pulldown menu selects which of the available images will become the current one. The current image is the one in which a IP technique is performed. Refer to section 2.3 for more information.

The last pulldown menu in the menubar provides help for the main window and information about the application. The first option pops a help dialog, while the second pops a dialog with some information such as the name of the application, the version number, the author, and the copyright.

2.1.2 The Message Area

The message area consists of a scroll window and a text widget. It provides information to the user about the tasks the application is carrying out, such as reading and writing files, errors detected by the program, image information, etc. All information displayed is kept in the text widget. To refer

to any previously displayed message just slide the scrollbar until the text is found. The text widget is output only.

2.1.3 The Action Area

The action area is a set of buttons that call different dialogs. These dialogs contain specific functions and information about the selected IP field. The fields present in the action area are: enhancement, analysis (not yet implemented), restoration (not yet implemented), matching (not yet implemented), segmentation, morphology, transformation and degradation. Pressing on a button will cause its respective dialog to pop-up. If the dialog is dismissed it can it poped-up again by pressing the respective button. Only one instance for each dialog is allowed.

2.2 The Read and Write Dialogs

These dialogs are used to select a file to read or write an image from or to disk, respectively. Both use the file selection box widget, this widget encapsulates the task of opening a directory file, reading its entries and traversing the directory tree. The contents are displayed in two list widgets, one holds the directory files within a given directory and the other holds ordinary, link and device files. Selecting a file and traversing the directory tree is done by clicking an item in the appropriate list widget. Figure 2.2 shows the dialogs.

The write dialog contains a modified file selection box. Besides it usual features, the write dialog provides a toggle box to allow the user specify the format in which a given image will be stored. Formats supported by the IP system include PGM, in raw and ASCII mode, GIF, JFIF, TIFF and PostScript (read only).

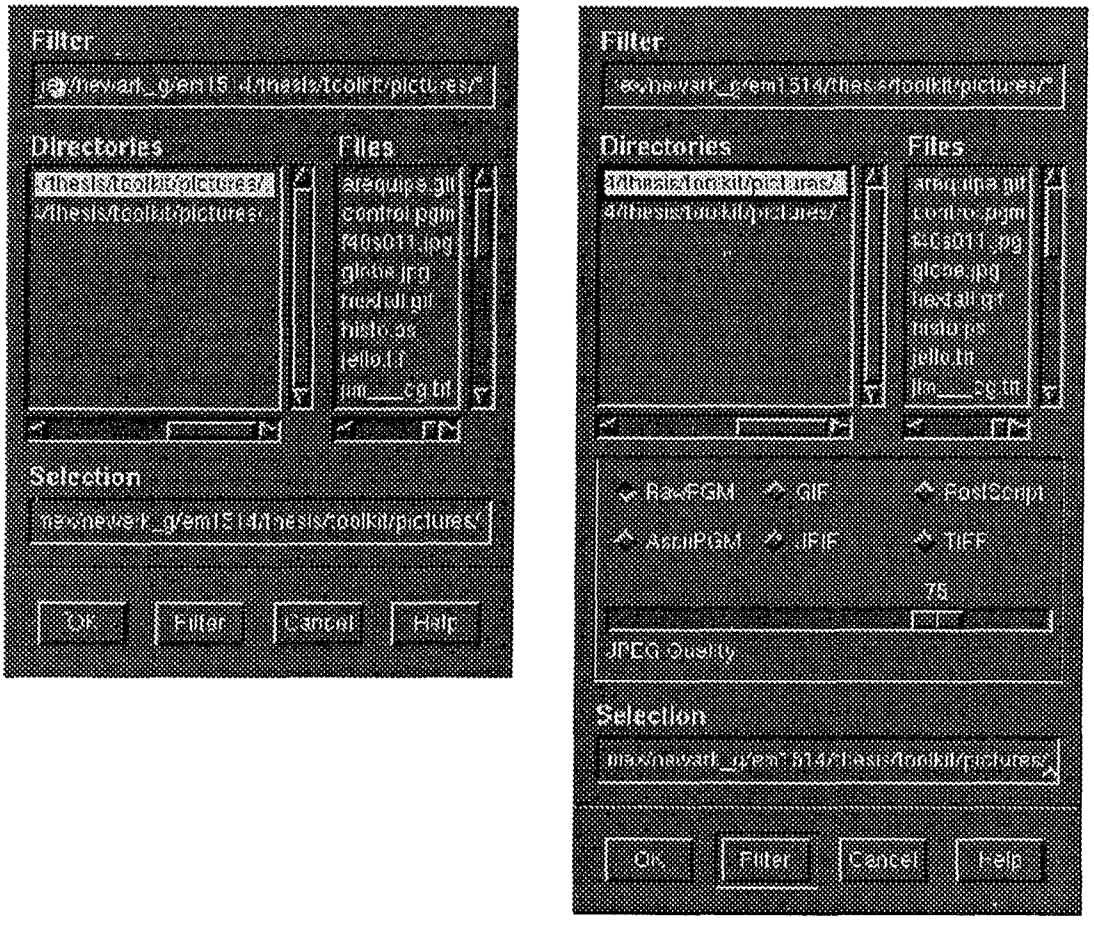

Figure 2.2 The Read and Write Dialogs.

PGM performs no compression in the image. JFIF format allows for JPEG compression, the lower the quality the greater compression ratio is achieved. GIF provides LZW compression. TIFF reads several compression techniques such as LZW, CCITT Group 3 Facsimile, PackBits and no compression at all, but only writes in LZW compression. Later application versions will provide more writing choices.

Each dialog provides its own help dialog with instructions how to use its features such as how to select files and traverse directories. The help dialog also provides information about all the file formats that IPIS supports.

2.3 Image Window Management

When an IP system displays only one image at a time, then its manipulation is trivial. If it is decided to process an image; there is no question to which image we are referring to, since there is only one. Similarly, when an image is processed and therefore producing a new image that needs to be displayed, there is only one old image that can be replaced. In general, decisions about image window manipulation are simplified because there is no need to develop a policy to choose between images.

In a multi-window IP system, this manipulation task is more complicated. There may be several images on display. If it is decided to perform an operation on an image, then the system must have the means to single out the image. This process can be automatic or user-assisted.

The number of images IPIS can display depends on the macro NUM_OF_IMAGES, which is defined at compile time. The default value is 6, computer systems with large amounts of memory can increase the value accordingly. However, the limit of images that IPIS can hold at once depends on the memory available at the server at that precise time.

In order to know which image to use when performing an operation. The system uses the concept of a ''current'' image. The current image is the latest accessed image unless the user selects another one by using the current image pulldown menu. The concept is implemented by creating a variable `current`. This variable indexes a data structure that contains all the information about the image. When an operation is performed, the indexed data structure is accessed to retrieve all the required information.

When an operation is performed it yields a new image to be displayed. The following steps are used to display a new image:

- Find an empty data structure.
- Set the data structure fields.
- Create a window with a drawing area.
- Place image in the drawing area.

As soon an empty data structure is found, `current` is updated making the newly created image the current one.

On the other hand, when an image is destroyed—done by pressing mouse button three in the image area—the following steps must be performed:

- Free or update resources such as memory and colormap cells.
- Destroy the drawing area widget.

If the image destroyed is the current one, `current` will be updated so that it points at the first non-empty image data structure it finds.

IPIS uses the function `get_current()` to automatically update the `current` variable. The code is shown in figure 2.3. The code first tries to find an empty `ximage` data structure. If successful, it allocates memory for the image histogram and return the index to the empty data structure—this value becomes the new current. If it fails, it means that IPIS is already displaying the maximum of images allowed. If that is the case, to display a new image it is necessary to destroy an old one. The application destroys the latest accessed image (current), later version may implement an alternate algorithm such as LRU, to make room for the new one.

Destroying an image is not as simple at it seems, certain resources must be updated or freed. IPIS uses the callback function `destroy()` to maintain integrity. The `destroy()` function is called automatically by the server after it receives a destroy event for the drawing area widget. The

```
int get_current(void)
{
 int    i;

 /*                                    Find an empty ximage structure. */
 for (i=0; (i < NUM_OF_IMAGES) && (ximage[i]); i++);

 /*                                     Check if search was successful */
 if (i < NUM_OF_IMAGES)
 {
  /*           Success. Allocate memory and return the new current. */
  hist[i] = (unsigned int *)XtCalloc(256, sizeof(unsigned int));
  return i;
 }
 else
 {
  /*  Failure. Maximum number of images on display. Need to destroy */
  /*                                one to make room for the new one. */
  XtRemoveCallback(area[current], XmNdestroyCallback, destroy, NULL);
  XtDestroyWidget(gettopshell(top[current]));
  destroy(area[current], &i, NULL);

  /*        Allocate memory for the histogram and return the value. */
  hist[current] = (unsigned int *)XtCalloc(256, sizeof(unsigned int));
  return current;
 }
}
```

Figure 2.3 Implementation of `get_current()`.

server does not call this function immediately, it waits until other functions are done processing before the X server actually makes the call.

Since the function `get_current()` destroys an image "manually", it must remove the image's destroy callback, else the server will use the callback at the wrong time. IPIS wants the image destroyed immediately so it can display the new one. The drawing area widget is tagged to be destroyed by `XtDestroyWidget()`, the widget will not actually be destroyed until the functions `get_current()` and the IP function that called it are done processing. In order to maintain integrity `get_current()` must call `destroy()` itself. Finally, memory for the histogram is allocated and the same current value is returned.

2.4 Colormap Management

Similarly to image window management, managing the colormap in a single-image system is trivial. If a private colormap is used there is nothing to be done since all gray levels are allocated. If a standard colormap is used, all gray levels can not be allocated because other running X applications are using some of the 256 colormap cells. To avoid running out of colormap cells, only the set of gray levels $G1$ used by the image must be allocated. When the number of elements in the set, $N(G1)$, is still greater than the number of available colormap cells, the application will have to allocate the closest gray levels instead or switch to a private colormap.

When the application needs to display a new image with set of gray levels $G2$, then it deallocates the set $G1–G2$ and allocates the set $G2–G1$. If $N(G2–G1)$ is greater than the number of available colormap cells, then the application will have to allocate the closest gray levels instead or switch to a private colormap.

In a multi-image system, managing the colormap is more complicated. The allocated colormap cells must be useful to more than one image. IPIS takes advantage in the fact that the human eye has difficulty in detecting more than 64 gray levels at a given time.

The Colormap is initialized at the start of the application. A minimum of 86 gray levels are allocated and are never relinquished. This number should be enough to adequately represent all images in the worst case scenario. If IPIS fails to allocate this minimum, a message will notify the user to quit some X applications in order to free colormap cells and restart the program; or to continue by using a private colormap.

The 86 initial gray scales are allocated evenly between the 0 and 255 gray levels by three levels intervals. Let A be the set of the basic allocated values. Example:

$$A = \{0,3,6,9,12,15,...,246,249,252,255\}$$

This arrangement is very convenient because when the colormap runs out of cells and the closest gray level must be allocated, it happens to be that the closest gray level will always be within one. Example:

- Let say there is no more room in the colormap and we need to allocate gray levels 100 and 101.
- Let the set $A = \{0,3,6,...,96,99,102,105,...,249,252,255\}$
- The closest previously allocated gray level to 100 is 99. Which is within one.
- The closest previously allocated gray level to 101 is 102. Which is also within one.

In general, if gray level x is not in A, $x+1$ or $x-1$ will be in A.

Other gray level values that are not in set A are allocated and deallocated on demand. When IPIS reads or calculates a new image, two variables are updated to reflect the need of a given gray level value. One is used to build a histogram and also is used as a reference of which gray levels and how many of them the image is using. The other one keeps count of the total use of a given gray level by all the images that are currently displayed. These variables are `hist[image_index][value]` and `use[value]`, respectively.

```
void set_colormap(int n)
{
 int i;
 int count=0;           /* Number of gray levels used by the image. */
 int close=0;           /* Number of closely allocated gray levels. */
 XColor gray;

 for (i=0; i < 256; i++)
 {
  if (hist[n][i])                  /* if the grey level is used ... */
  {
   count++;
   if (!allc[i])             /* if it has not yet been allocated ... */
   {
    gray.red   = (unsigned short)(i * 257);   /* Assign RGB values. */
    gray.green = (unsigned short)(i * 257);
    gray.blue  = (unsigned short)(i * 257);
    gray.flags = DoRed | DoGreen | DoBlue;

    /*                          Allocate color in standard colormap. */
    if (!XAllocColor(theDisplay, theColormap, &gray))
    {
     /*          If allocation failed. Allocate closest gray level. */
     if (i % 3 == 1)
      cmap[i] = cmap[i-1];
     else
      cmap[i] = cmap[i+1];
     close++;
    }
    else
    {
     /*  Success. Do not forget to keep the colormap cell position. */
     cmap[i] = gray.pixel;
     allc[i] = 1;
    }
   }
  }
 }
 wprint("%d gray levels. Allocated %d true, %d close.0,
        count, count-close, close);
}
```

Figure 2.4 Implementation of `set_colormap()`.

When an image is ready to display, IPIS sets the colormap by looping through the `hist[][]` variable. If the image is using a gray level, then checks if that gray level has been allocated before. If it has not, IPIS sends a request to the X server to allocate the gray level in question. If the

request fails, the application assigns the closest gray level available. If the request was successful, the application saves the position in the colormap where the gray level is located and updates the allocation flag. Figure 2.4 shows the code to allocate gray values.

Colormap entries are deallocated when an image is destroyed. The check if as gray value x must be deallocated, IPIS subtracts the values in `hist[n][x]` from `use[x]`. If `use[x]` is equal to zero, it means that no other image is using that particular gray level, therefore it can be deallocated.

2.5 Other Features

IPIS is able to display histograms from images by pressing shift-mouse button 1 on the desired image. If a image is destroyed and its respective histogram is also mapped on the screen, then the histogram will also be destroyed. It would be inaccurate if the histogram remains on the screen and a new image replaces the old one.

The application can also extract subimages from an image. Mouse button 1 is used to define the upper left corner, while button 2 is used for the bottom right corner. Once the subimage is extracted, it becomes an image on its own right; i.e. all the operations and rules applying to an ordinary image apply to the subimage.

Similarly, the decimal values of the pixels can be displayed on screen. Mouse button 1 sets the upper left corner, and shift-mouse button 2 sets the bottom right corner. If the user simply presses button 2 or shift-button 2, the origin will be consider as the upper left corner.

CHAPTER 3

IMAGE PROCESSING ROUTINES

IPIS contains a large number of IP routines. These can be grouped in enhancement, segmentation, transformation, morphology and degradation. Later versions may provide more routines in these IP fields as well as may implement restoration , matching and analysis routines. Each IP field is provided with a dialog to call all the functions relative to that particular field.

3.1 Enhancement Routines

Image enhancement can be performed in two different aspects. It is possible to work on the contrast of the image by modifying the histogram; or to filter the image in order to get a smoother or sharper result. Figure 3.1 shows the dialog.

3.1.1 Histogram Processing

The routines provided are stretching, global equalization and local equalization. The simplest routine is histogram stretching, where the lowest pixel value is mapped to 0 and the largest one is mapped to 255. All other values are linearly mapped in between.

In histogram equalization the gray scale distribution is modified by a transformation function which can be expressed as

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

where g is the processed image, f is the original image and T is the

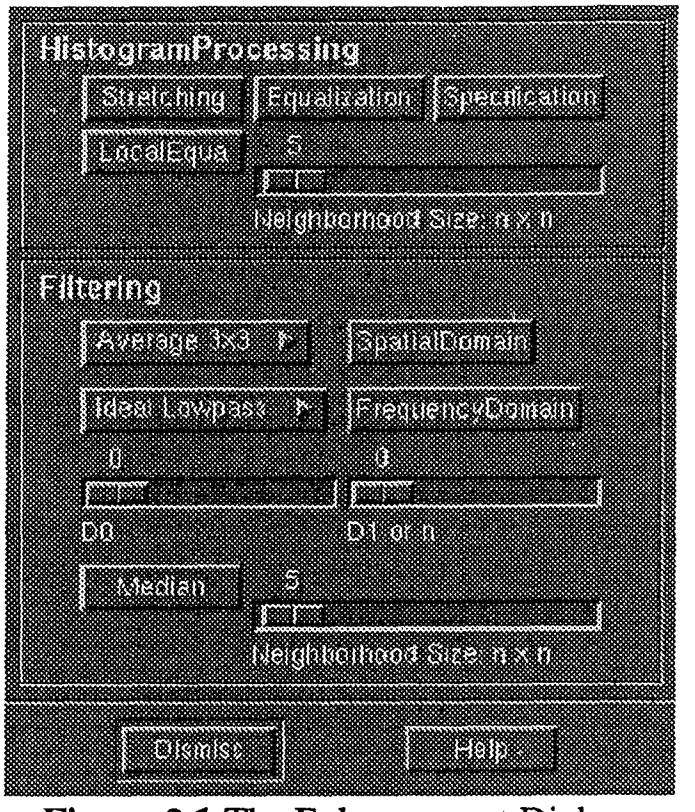

Figure 3.1 The Enhancement Dialog.

transformation function which converts what is usually a normal gray level distribution into a uniform distribution. Therefore equalizing the possibility of use between gray levels.

Local histogram equalization applies the same process as histogram equalization to local neighborhoods in the image. The scale next to its activation button is used to provide the neighborhood size for the operation. Histogram specification has not yet been implemented.

3.1.2 Filtering

Filtering is possible in the spatial and frequency domain. The operation is performed by convolving the input image with a user selected mask—also known as a filter—element. IPIS provides the following spatial filters for user selection:

Average 3x3: $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ Smooth 3x3: $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ Sharpen 3x3: $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Average 5x5: $\frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$ Smooth 5x5: $\frac{1}{60}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 4 & 4 & 4 & 1 \\ 1 & 4 & 12 & 4 & 1 \\ 1 & 4 & 4 & 4 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$

Average 7x7: $\frac{1}{49}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$

Sharpen 5x5: $\begin{bmatrix} 0 & -1 & 1 & -1 & 0 \\ -1 & 2 & -4 & 2 & -1 \\ 1 & -4 & 13 & -4 & 1 \\ -1 & 2 & -4 & 2 & -1 \\ 0 & -1 & 1 & -1 & 0 \end{bmatrix}$ Laplacian 5x5: $\begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 25 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$

Laplacian 3x3: $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ Prewitt left: $\frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ Prewitt top: $\frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

Sobel left: $\frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ Sobel top: $\frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

The MAX and RMS versions of the Prewitt and Sobel operators are the combination of the top and left filters such as the result is the maximum value or the square root of the sums, respectively.

The filters are activated by selecting the appropriate filter from the option menu and pressing in the ''SpatialDomain'' button.

For the frequency domain the image is applied the Fourier transform and then filtered by attenuating a specific range of frequencies. The 2-D Fourier transform consist of two one-dimensional transforms applied successively. The columns are transformed first and then the rows.

The following filters are available on IPIS:

- Ideal lowpass and highpass filter
- Butterworth lowpass and highpass filter.
- Exponential lowpass and highpass filter.
- Trapezoidal lowpass filter
- A variation of the trapezoidal highpass filter.

All the filters, except the modified trapezoidal filter, can be found on Gonzales and Wintz (2), Pratt (3), and Rosenfeld and Kak (5).

The traditional trapezoidal filter is defined by

$$H(u,v)=\begin{cases}0 & \text{if } D(u,v)<D_1\\ \dfrac{D(u,v)-D_1}{D_0-D_1} & \text{if } D_1\le D(u,v)\le D_0\\ 1 & \text{if } D(u,v)>D_0\end{cases}$$

while the new variation is

$$H(u,v)=\begin{cases}\dfrac{1-D_1}{D_0}D(u,v)+D_1 & \text{if } D(u,v)<D_0\\ 1 & \text{if } D(u,v)\ge D_0\end{cases}$$

Figure 3.2 shows the radial cross for both filters. To the left is the traditional trapezoidal filter, and to the right is the new trapezoidal filter. The traditional one tends to lose too much frequency information, and when the image is transformed back to the spatial domain it is sometimes incomprehensible. The new filter sharpens the image better while it keeps more information.

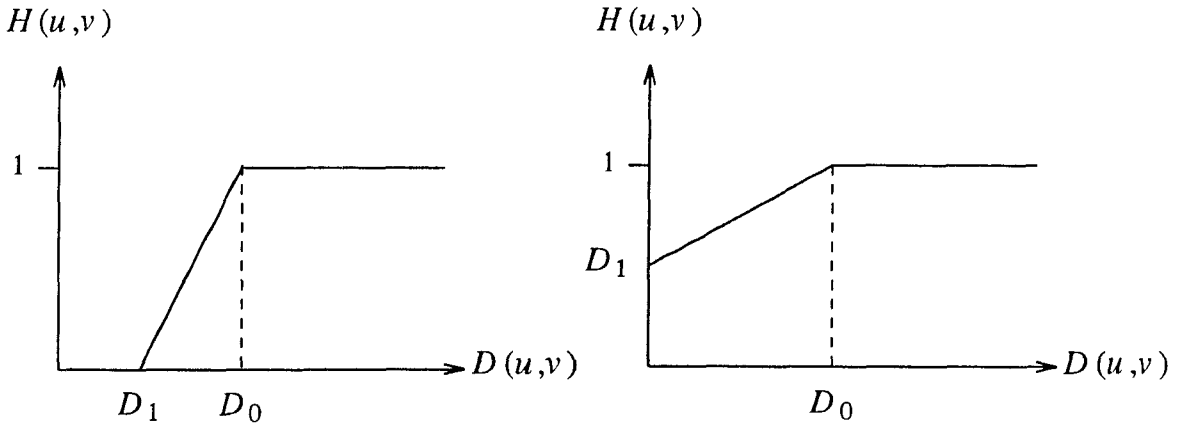

Figure 3.2 Traditional and New Trapezoidal Filters.

The D_0 and D_1 cutoff distances are provided by the scales with the same name. For the Butterworth and exponential filters the "D1" scale provides the exponential parameter.

The last filter in the dialog is a fast median filter. The neighborhood size information is provided by the scale to the right of the button.

3.2 Segmentation

The segmentation routines were perhaps the most difficult to implement. The problem was that initially the application was not designed to modify an image that was already displayed on the screen, IPIS could access the image information through the `ximage[]` structure but it could not modify the pixmap were the on-screen image was being stored by the X server. If an image needed to be modified, the new modified image would be displayed in a new window with a frame of its own.

This approach proved to be unacceptable if the user needed to apply an interactive bilevel segmentation by sliding a scale. Unfortunately there was no time to redesign the whole application and solve the problem from

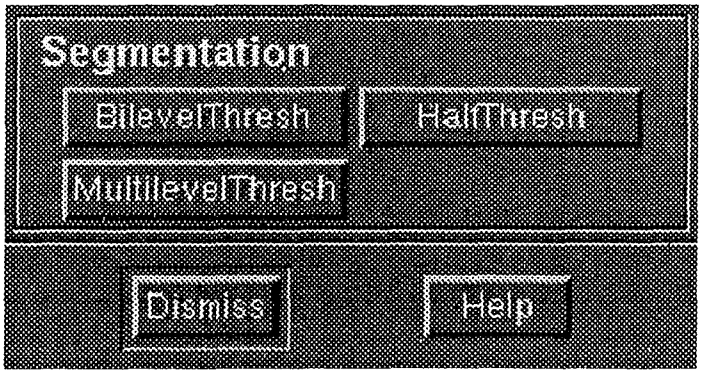

Figure 3.3 The Segmentation Dialog.

its roots. Instead an approach that would solve the problem, but would create another one easier to control, was chosen.

This approach consisted in declaring the drawing area widget—the Motif widget where the image is displayed—and the pixmap structure—the data structure where the X server stores the image and uses to redraw it when expose events occur—global variables. By making them global they could be accessed from any point in the application.

So far this would not create any problems, making these variables global will allow the application to get hold of the window where a given image is being displayed and to the data structure where it is being stored. Although some encapsulation was lost.

The problem now is that the application has the means to alter information on a window, the routine altering the image would have to remap the new pixel values to the proper position in the colormap—gray level 125 is not necessarily stored in colormap cell number 125—and to allocate by itself the new gray levels in the colormap, as well as to deallocate the gray levels that are no longer used. This is a critical situation since if some other routine tries to update the colormap in the mean time, the colormap will become corrupted. One routine is not going to be aware of what the other

is doing, one may try to allocate a gray level and think it has succeeded, while other may deallocate the same gray level, causing the other to display an incorrect result.

To assure that only one routine is modifying the colormap as well as the on-screen image, the segmentation dialogs are application modal, i.e., no other operation can be performed while the interactive segmentation is taking place.

To keep the following three routines simple neither of them updates the colormap. They do their best to display the segmented images with the gray levels available. This does not mean that the critical period has been avoided, since all three of them update the `use[]` variable. If this variable gets corrupted, IPIS may deallocate the wrong gray levels at a later time, or may not deallocate the gray levels that are not longer used.

3.2.1 Bilevel Thresholding

This kind of thresholding assigns white pixels to any gray level above the threshold indicated by the scale. When the desired effect has been achieved the modal dialog can be destroyed by pressing the ''Done'' button.

3.2.2 Half Thresholding

Similar to bilevel thresholding. The difference is that the gray levels below the threshold remain the same.

3.2.3 Multilevel Thresholding

This is the most functional of all the tresholding subroutines. It works by defining cutoff points and assigning them a gray level value. To define a cutoff point, press the mouse button 1 on the histogram drawing area, a bar will appear signaling the cutoff. The user may assign up to 256 cutoff

points. To assign a value, slide the scale until it is shown on screen, then press mouse button 2 on the area left of the chosen cutoff bar—if the cutoff bar already has a value, this old value will be replaced by the new one.

Once the desired cutoff points as well as its gray level values has been chosen, the image will be segmented once the user presses the 'Apply'' button. To start all over again, the user must press the ''Clear'' button. To destroy the modal dialog the button ''Done'' must be pressed.

3.3 Mathematical Morphology

Most of the image morphology routines provided by this dialog use the libmorph library. This library was created by Dr. Richard A. Peters at Vanderbilt University. The functions provided are:

- Dilation.
- Erosion.
- Opening.
- Closing.
- Order statistic filter (Rank filter).
- Lower-upper-middle filter (LUM).
- LUM smoothing filter.
- LUM sharpening filter.
- Image minus opening (Tophat).
- Closing minus image (Bothat).

Erosion and Dilation can be negated creating the following actions:

- Not dilate = Dilation && !Image
- Not Erode = Image && !Erosion

The rank filter, uses the scale of the same name to obtain the order of the statistical filter to perform. If the rank value is zero, it applies a median filter. If the value is one, then a dilation is performed, If the value supplied is the same as the support of the structuring element, then an erosion is performed. Otherwise the statistical filter is performed.

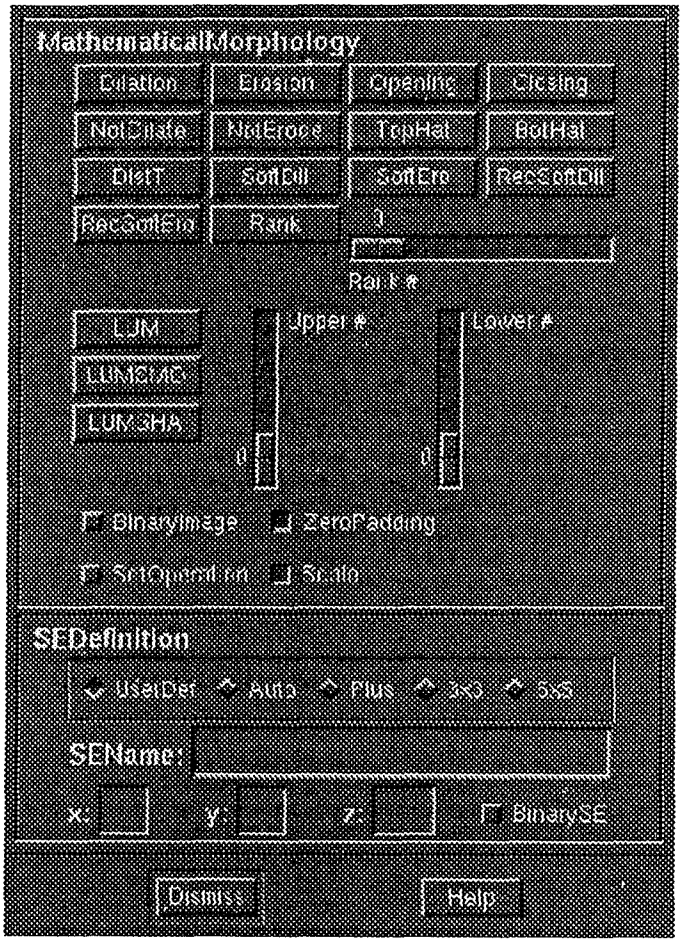

Figure 3.4 The Morphology Dialog.

The lower-upper-middle filters use the values provided by the scales ''Upper #'' and ''Lower #'' to obtain the values for the upper and lower LUM parameters, respectively. In any other operation the scales provide the upper and lower values for a bilevel segmentation. In between gray levels are set to white, while others are set to black. This segmentation is performed before the morphological function.

The structuring element for these operations is specified in the ''SE Definition'' section. To provide the routines with a user defined SE, the file name containing such SE must be typed in the ''SE name'' text field; also the ''UserDef'' toggle button must be set to true. For convenience, the libmorph library provides four ''canned'' SEs to use with its routines. To select one, it is only necessary to set the proper toggle button. The possible choices are:

- Auto. Defines a circular SE with its horizontal and vertical dimensions provided by the x and y text fields. The z field provides the gray level of the origin.
- Plus. Defines a 3 by 3 SE with the shape of a ''+''.
- 3x3. Defines a 3 by 3 square SE.
- 5x5. Defines a 5 by 5 almost circular SE.

Figure 3.4 shows the morphology dialog. For more information on the libmorph library routines. Refer to the on-line help menu.

Among the morphological functions not provided by the libmorph library are the euclidean distance transformation and the four soft morphological operations.

The distance transform finds the euclidean distance between the object pixels of an image and the background. Since it only applies to binary images, the routine uses the ''Upper #'' and ''Lower #'' scales to perform a segmentation before it actually does any calculations. The distance is found by iteratively applying erosions until no change is detected. Finally the square root is performed for each individual pixel.

The soft morphological operations are another kind of order statistics combined with morphological concepts. The function implementations are based on the work done by Shih and Pu (6). These functions use the ''rank #'' scale in order to obtain an ordering value. Dilation selects the nth order largest value, while erosion selects the nth order smallest value.

Recursive soft morphology differs in the fact that it updates the input image with the output as the results are being obtained. Both kinds of soft morphological functions split the structuring element B into two sets. One core set A, i.e., $A \subseteq B$ and the soft boundary $B-A$. The SE can be read by setting the ''UserDef'' toggle button and providing the name of the ASCII file containing the SE. The file first contains the number of elements in the soft morphological boundary ($B-A$), followed by the ordered pairs specifying the coordinates for $B-A$. Next is the number of elements in the core set (A), followed by the coordinates for A.

3.4 Transformation

This dialog allows the user to manipulate the image from three different aspects: image manipulation, pixel manipulation and domain manipulation. Figure 3.5 shows the dialog.

3.4.1 Image Manipulation

The image can be flipped vertically or horizontally. These operations are very simple, to obtain the result it is only necessary to copy the pixels onto the new image in the correct order. Using the same technique, the image can be rotated 90 degrees either clockwise or counter clockwise.

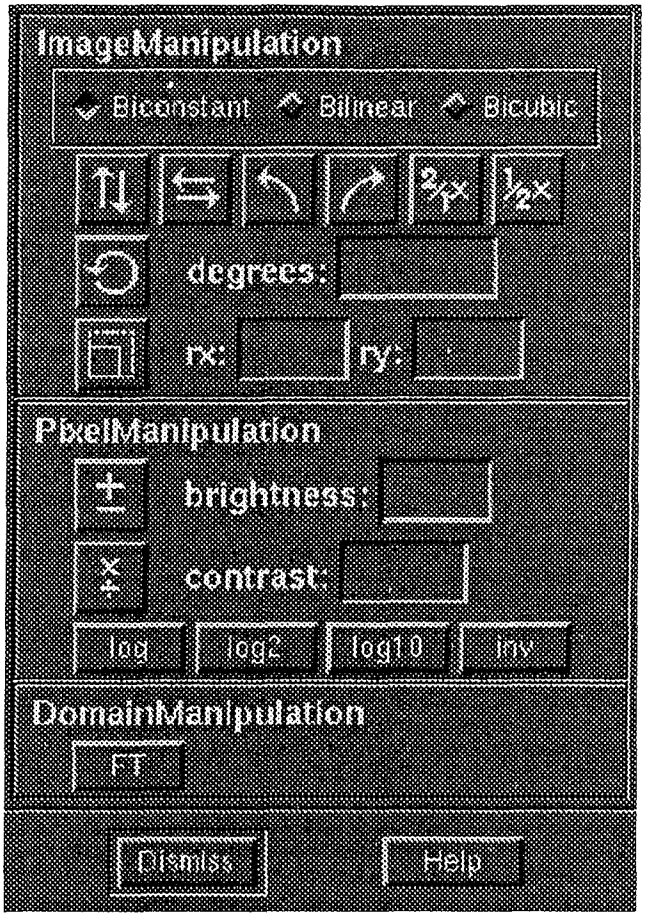

Figure 3.5 The Transformation Dialog.

Rotating an image an arbitrary number of degrees is a more challenging task. An affine transformation is used to accomplish such task. The matrix used is

$$\begin{bmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{bmatrix}$$

The problem is that after applying the transformation some pixels of the same image overlap each other, and some other pixels in the image are left without an assigned value.

The solution to this problem is to interpolate the gray levels into their final position. Three pixel interpolations are possible: biconstant, bilinear and bicubic.

Bicubic interpolation offers the best results but it also is the most computational complex of all. Bilinear is the fastest of the three, but offers the lowest quality. Probably the best choice is bilinear, it is relatively fast and the results are of good quality.

To activate the rotation just press the rotation button or press the return key in the text widget. The rotation is specified in degrees. It accepts positive and negative values. If the value is zero, the function will not be performed since zero is a no-op.

Images can also be scaled. Another affine transformation is used to achieve this goal. The matrix used is

$$\begin{bmatrix} rx & 0 \\ 0 & ry \end{bmatrix}$$

Here the problem is that some pixels in the new image are not assigned a value if the image is being scaled up, or some pixels overlap if the image is being scaled down.

Interpolation is also the solution here. As in rotation, biconstant and bilinear are provided. Bicubic is not yet implemented. To activate the scaling press the scale button or press return at the second text widget. Two buttons for fast operation are provided, these double or half the image size.

3.4.2 Pixel Manipulation

This routines are used to operate on each pixel and not to the image as a whole. It is possible to increment each pixel with a given amount to alter the brightness. Similarly, each pixel can be multiplied by a given ratio to alter the contrast. To obtain a negative of a picture is only necessary to obtain the complement of the pixel.

The function log, log2 and log10 are just provided to assist in the remapping of gray levels. To apply the following gray level remapping function

$$new = 32*log\,(old+2)$$

it is needed to increment the image brightness by two, then apply the log by pressing the button of the same name and finally the contrast is modified by a ratio of 32.

The final result will be an approximation to the actual values since floating point information is lost when the output of the logarithmic function are rounded to be displayed.

3.4.3 Domain Manipulation

The routine provided in this section displays the magnitude of the Fourier domain of a given image. This is the only purpose. To operate in the Fourier domain see the filtering section on image enhancement.

3.5 Degradation

IPIS can degrade images by adding gaussian noise to each pixel. The dialog allows the user to specify the mean and the standard deviation of the operation. The maximum values in the scale can be customized by providing the respective resources. Figure 3.6 shows the dialog.

The `guass()` function is used to add noise to the image. The random seed is initialized with the current time. The function loops until all pixels have been visited. In each loop a random number is obtained and is added to the pixel.

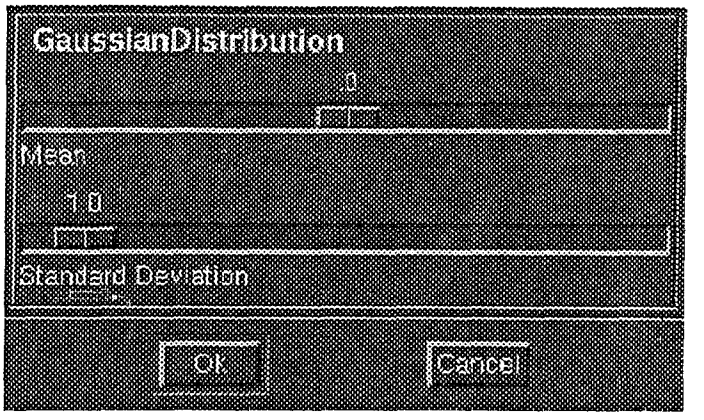

Figure 3.6 The Degradation Dialog.

The number generator function uses the standard C function `rand()` to obtain 12 random numbers which will be normalized and added together. The expected value, 6, is substracted from the added number and then is multiplied by the standard deviation and incremented by the mean.

CHAPTER 4

EXTENDING IPIS

One of the goals of the application is to become a comprehensive image processing tool. In order to cover all possible fields and include the latest techniques, people related to IP must be able to contribute code.

The structured design of IPIS makes it very easy to understand and extend. The programmer needs to be aware the global variables that provide information about the state of the application, the image and the X server. A generic function template can be modified to implement most of the IP routines. All functions currently implemented use this generic template. Last but not least, the window dialogs must be modified to provide an interface to the newly created routines.

4.1 Global Variables

The global variables can be divided in three categories. The first would be variables that provide information about the X server. The second category would be variables that provide information about the state of IPIS and the last group would be variables that provide information about the image. All these variables are declared in the header file.

4.1.1 Server Variables

These variables provide basic information about the display and resources of of the server. These variables are:

- theDisplay
- theScreen
- theRootWindow
- theVisual
- theDepth
- theColormap
- myColormap
- theGC
- watch;
- top_level

Most of these variables are useful when using Xlib functions. For example all functions that make a request to the server have as their first argument a pointer to the display. Variable `theDisplay` is such pointer. Similarly, `theScreen` —which holds the screen number in the display—it is also useful for some Xlib functions. Variable `theRootWindow` is a pointer to the root window of the display, `theVisual` holds the information about the capabilities of the display. IPIS only supports pseudocolor visuals. The depth of the screen is represented by `theDepth`, IPIS only supports 8-bit displays.

Pointers to the colormaps used by the application are in `theColormap` and `myColormap`. The former points to the standard colormap, while the other points to private colormap with all 256 gray levels allocated.

The graphic context provides information for drawing text and graphics in a window, `theGC` is useful when drawing lines, when setting the foreground and background or when copying pixmaps. The `watch`

variable is the shape of the mouse cursor when the application is busy making computations.

The variable `top_level` does not really belong in this category, but has less in common with the other global variables. This variable represents the widget of the main window. It is useful when creating dialog shells and when changing the mouse cursor to a watch when the application is busy performing an operation. The variable `top_level` is a global variable just to provide speed when changing cursors. It is possible to obtain the widget of the main window—the `top_level` widget— by using the function `gettopshell()`.

4.1.2 State Variables

These variables provide information about the current state of the application. These variables are:

- use[]
- cmap[]
- current
- image_count
- mode

The `use[]` array has a very important role in the application. It is in charge to count the number of times a certain gray level is being used in the application. The standard colormap depends of it in order to deallocate colormap cells. The programmer should always update this array when it is going to display a new image.

The `cmap[]` array was not meant to be a global variable at the beginning. It was converted into a global one in order to implement the

segmentation routines. This array contains the positions in the colormap for the allocated gray levels, i.e., if cmap[6] is equal to 78, it means that gray level 6 is at position 78 of the colormap. The programmer should avoid using this variable as much as possible, since it may be hidden in later releases.

The `current` variable was already discussed at image window management. It indexes the "current" `ximage[]` structure. The simplest way to update it is by using the `get_current()` function. The `current` variable should always point at the latest image accessed by the system.

The `image_count` has the only purpose of letting the programmer know if there are any images in which operations can be performed. If the user requests for an IP function but there no current image to work on, i.e., `image_count == 0`, then the function must return. The programmer must not altered the value of the variable. Once again this variable is just used as a flag.

The `mode` variable holds several pieces of information. The application uses this variable to find out if it should allocate a private colormap or should use the standard colormap. However, the programmer should not worry about colormap management and allocation—he/she should only make sure to update `use[]` accordingly. The programmer should use this variable to know how he/she must deal with pixel roundoffs. The possibilities are: scale, truncate and do nothing. The macros SCALE, TRUNCATE and NOTHING are defined in the ipt.h header file for this purpose, just bitwise the macros and the variable accordingly to find out the action to follow.

4.1.3 Image Information Variables

These variables provide information about the image itself, useful for retrieving data and performing IP functions. These variables are:

- hist[][]
- ximage[]
- pixmap[]
- histogram[]
- area[]
- top_hist[]

The `hist[][]` is a two dimensional array, where the first index is used for images and the second for gray levels. This variable stores image histogram information. Memory is allocated through the function `get_current()`. It was designed to obtain its memory dynamically since it would provide better memory management for systems which compiled with a large NUM_OF_IMAGES macro. The programmer must make sure to initialize this variable properly, since it is also used to allocate gray levels.

The `ximage[]` structure contains all the information about the image, including width, height, depth and the image data.

The `pixmap[]` is used to store the image in memory and redraw the image after every expose event. It was made a global variable only for the segmentation routines. The programmer should only use them when it is absolutely necessary. Later versions may hide this variable.

The `histogram[]` is the pixmap used for the histogram window. There is no strong reason for this variable to be global. It will be hidden in the next version.

The `area[]` is another variable that was made global for the segmentation routines. The programmer should avoid using it. It may get hidden in later versions.

The last variable is used for the top level shell of the histogram window. It is necessary to destroy the histogram window when its corresponding image is destroyed. It should not be of much use to the programmer.

4.2 Function Template

The template in figure 4.1 should be useful even with complex procedures. It has been oversimplified; some steps can be merged depending on the nature of the task.

Normally a function like this will be called from a dialog, i.e., it will be a callback function. The first parameter contains the widget that originated the call. In most cases it will be a button widget.

The second parameter is the callback data. This data is assigned when the callback function is added to the widget. It is useful when the function needs a parameter supplied by the use. For example, the client data for local histogram equalization would be the neighborhood size. It is possible to pass multiple values through the client data. For example, the gaussian noise function receives a struct data type which contains two members, the mean and the standard deviation.

The third parameter depends of the widget producing the callback. Most of time it will be from a button widget and will not have any use. A few times, like in a scale callback, it will have a value that must be retrieved. Each widget has its own callback struct data type. Refer to Heller (3) or any other Motif text for more information.

```
void ip_function(Widget w, XtPointer data, XmAnyCallbackStruct *cbs)
{
 /* Check if there is any image to work on    */
 if (!image_count)
 {
  wprint("No image present.0);
  return;
 }

 /* Set the busy cursor.                       */
 XDefineCursor(theDisplay, XtWindow(top_level), watch);
 XFlush(theDisplay);

 /* Obtain all image information necessary.   */
 ix     = ximage[current]->width;
 iy     = ximage[current]->height;
 image  = (PIXEL *)ximage[current]->data;

 /* Allocate memory for new image.            */
 result = res = (PIXEL *)XtCalloc(ix*iy, sizeof(PIXEL));

 /* retrieve function parameters. if any.     */
 parameters = (parameters_type *)data;

 /* Body of the function. Do processing       */
 for(start; end; step);

 /* Get a new current value.                  */
 current = get_current();

 /* Update statistics for hist and use.       */
 for (y=0; y < iy; y++)
  for (x=0; x < ix; x++)
  {
   use[*res]++;
   hist[current][*(res++)]++;
  }

 /* Allocate new ximage structure.            */
 ximage[current] = XCreateImage(theDisplay, theVisual, 8,
                   ZPixmap, 0, (char *)result, ix, iy, 8, 0);
 ximage[current]->byte_order       = MSBFirst;
 ximage[current]->bitmap_bit_order = MSBFirst;

 /* Display the image.                        */
 display_image(current);

 /* Reset the cursor to normal                */
 XUndefineCursor(theDisplay, XtWindow(top_level));
}
```

Figure 4.1 Template Function for IPIS.

The template in figure 4.1 shows the several parts that an IPIS function is composed of. The first part would be to check that there is an image to work on. The second would be to set the cursor to a watch, so that the user knows that IPIS is working. The third would be to access all the necessary image information. Most of the time, the width, the height and the raw data are enough.

The next step would be to allocate memory for the result. Notice that IPIS does not use two dimensional arrays, instead a pointer to a one dimensional array is used. This is done because it is faster to access the data by incrementing the pointer one by one than using indexes with a conventional array. For that reason we use two pointers to refer to the data, `result` and `res`. The latter is used to access the data by incrementing it, when it is needed to start again from the beginning, `result` is used to reset the `res` pointer.,

The fifth part would be retrieve the parameters used by the function, if any. Then we are ready to process the data accordingly to the technique being implemented. Once it is finished, it is time to set the resulting image as the current one. The function `get_current()` is used for that purpose. From this point on the input image is an old one, and the output image is the current.

The eight step is to update the histogram and pixel usage information. In most cases this part will be merged with the processing of the image (sixth step) to increase throughput. Here is shown separately to make it simple. Have in mind that the seventh step HAS to precede the eighth step, if it is decided to merge steps eighth and sixth, step seven will have move up as well.

Step nine is to create a `ximage[]` structure to store the new image information. Step ten, displays the image in its own window. Finally, set the cursor back to its original shape.

As you can there is no need to deal with some internals such as colormap allocation or any drawing primitives. Once the display function is called IPIS takes care to allocate the needed gray levels, create a window and draw the image. Similarly, the user or the programmer do not have to worry about destroying an image, since IPIS will do all house cleaning automatically.

4.3 Modifying Dialogs

Modifying the dialogs is relatively simple. The best advice is to read a book about X and Motif and figure out how it works all together. Nevertheless, the programmer should be able to figure out how to create widgets and assign callback functions by just looking at the code.

Creating good looking user interfaces is another matter. Some experience is required, it is somewhat tedious to align widgets and make them look like one is expecting.

CHAPTER 5

FUTURE WORK

Large amounts of work are left to be done, there are always new ideas to be implemented or new requirements to be met. The near goal is to finish implementing all the functions that are left such as histogram specification, bicubic interpolation, restoration techniques, etc.

The next step is more ambitious, initially when I designed IPIS my objective was to do some color processing. In order to achieve that goal the program will have to be rewritten.

The way is implemented now, IPIS can only handle gray scale images. It can not even handle true binary images, IPIS emulates binary images by only allocating two gray levels, black and white, to a 256 gray level image. This results in waste of memory space.

The new approach will have to be object oriented in order to take advantage of inheritance, polymorphism and encapsulation. The language of choice is C++. The program will not have to be rewritten from scratch, many of the current routines can be modified to be useful. The overall skeleton of the program will be maintained.

Inheritance will be advantageous because complex image classes may be derived from simpler or basic image classes. Figure 5.1 shows a possible class hierarchy. Certain function and data types may be enherited from a basic class, while more specific functions are defined in the current class.

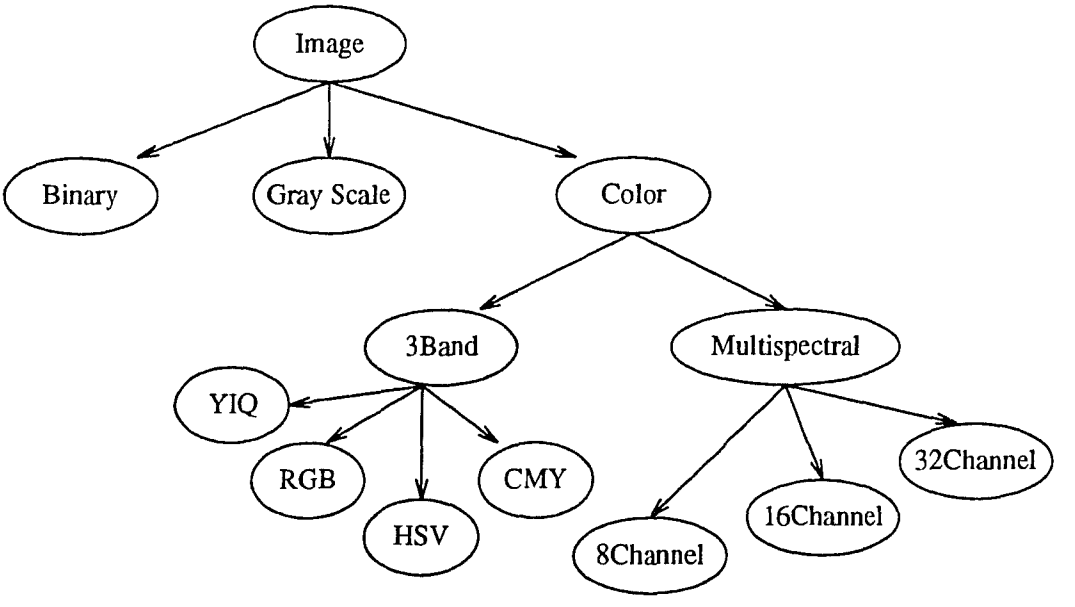

Figure 5.1 Possible Object Class Hierarchy.

Encapsulation along with data abstraction will also be advantageous. These features will produce cleaner code and will allow to hide implementation detail from users and programmers. For example, assume that a user wishes to perform histogram equalization on an image. Since functions are member of the objects, a color image may have a function called hist_equa, similarly a gray scale image may also have a function called hist_equa. Although both functions have the same name, and in principle do the same thing (equalize histograms), the color function implementation is totally different from the gray scale one. But for the user or the programmer the difference will be hidden. From their point of view they will be applying a histogram equalization to an "image" regardless of its type.

Polymorphism will be useful because will allow the programmers to treat the different kind of images, just by what they are: images. For example, There will not be a need for a write function that takes as an argument a color image object, and another write function that takes as an argument a

binary image object. A single write function with an image object as an argument will suffice, because after all color and binary image objects are image objects, both are derived from the same basic image class.

There is still a long way to go before final design decisions are made, in the mean time new alternatives will be considered.

APPENDIX A

WIDGET HIERARCHY

The widget hierarchy is important in order to customize the application. The list is given by the widget name and between parenthesis the widget's class. If the widget was created with a utility function and the widget's children are unknown, then the utility function will be between parenthesis.

```
Ipis (topLevelShell)
    main (XmMainWindow)
        menubar (XmRowColumn)
            button_0 (XmCascadeButton)
                filemenu (XmRowColumn)
                    button_0 (XmPushButton)
                    button_1 (XmPushButton)
                    separator_0 (XmSeparatorGadget)
                    button_2 (XmPushButton)
            button_1 (XmCascadeButton)
                settmenu (XmRowColumn)
                    button_0 (XmCascadeButton)
                        pullmenu (XmRowColumn)
                            button_0 (XmToggleButton)
                            button_1 (XmToggleButton)
                            button_2 (XmToggleButton)
                    button_1 (XmToggleButton)
            button_2 (XmCascadeButton)
```

```
                curr_menu (XmRowColumn)
                    button_0 (XmPushButton)
                    button_1 (XmPushButton)
                    button_2 (XmPushButton)
                    button_3 (XmPushButton)
                    button_4 (XmPushButton)
                    button_5 (XmPushButton)
            button_3 (XmCascadeButton)
                help_menu (XmRowColumn)
                    button_0 (XmPushButton)
                    button_1 (XmPushButton)
    column (XmRowColumn)
        scroll (XmScrolledWindow)
            text (XmText)
        form (XmForm)
            Enhancement    (XmPushButton)
            Restorationt   (XmPushButton)
            Segmentation   (XmPushButton)
            Transformation (XmPushButton)
            Analysis       (XmPushButton)
            Matching       (XmPushButton)
            Morphology     (XmPushButton)
            Degradation    (XmPushButton)
    Open (XmCreateFileselectionBoxDialog)
    Write (XmDialogShell)
        selbox (XmFileSelectionBox)
            frame (XmFrame)
```

column (XmRowColumn)
toggle_box (XmRowColumn)
RawPGM (XmToggleButton)
AsciiPGM (XmToggleButton)
GIF (XmToggleButton)
JFIF (XmToggleButton)
PostScript (XmToggleButton)
TIFF (XmToggleButton)
scale (XmScale)
About (XmDialogShell)
pane (XmPanedWindow)
form1 (XmForm)
tower (XmLabelGadget)
name (XmLabelGadget)
author (XmLabelGadget)
copyright (XmLabelGadget)
form2 (XmForm)
logo (XmLabelGadget)
caption (XmLabelGadget)
Enhancement (XmDialogShell)
pane (XmPanedWindow)
column (XmRowColumn)
frame1 (XmFrame)
form1 (XmForm)
HistogramProcessing (XmLabelGadget)
Stretching (XmPushButton)
Equalization (XmPushButton)

Specification (XmPushButton)

LocalEqua (XmPushButton)

scale1 (XmScale)

frame2 (XmFrame)

form2 (XmForm)

Filtering (XmLabelGadget)

option1 (XmCreateSimpleOptionMenu)

SpatialDomain (XmPushButton)

option2 (XmCreateSimpleOptionMenu)

FrequencyDomain (XmPushButton)

scale2 (XmScale)

scale3 (XmScale)

Median (XmPushButton)

scale4 (XmScale)

form3 (XmForm)

Help (XmPushButton)

Dismiss (XmPushButton)

Segmentation (XmDialogShell)

pane (XmPanedWindow)

frame (XmFrame)

form1 (XmForm)

Segmentation (XmLabelGadget)

BilevelThresh (XmPushButton)

HalfThresh (XmPushButton)

MultilevelThresh (XmPushButton)

form2 (XmForm)

Help (XmPushButton)

Dismiss (XmPushButton)
Transform (XmDialogShell)
pane (XmPanedWindow)
board (XmForm)
frame1 (XmFrame)
form1 (XmForm)
ImageManipulation (XmLabelGadget)
frame2 (XmFrame)
toggle_box (XmRowColumn)
Biconstant (XmToggleButton)
Bilinear (XmToggleButton)
Bicubic (XmToggleButton)
updown (XmPushButton)
leftright (XmPushButton)
ccw (XmPushButton)
cw (XmPushButton)
double (XmPushButton)
half (XmPushButton)
rotate (XmPushButton)
degrees: (XmLabelGadget)
degrees_t (XmTextField)
scale (XmPushButton)
rx: (XmLabelGadget)
x_t (XmTextField)
ry: (XmLabelGadget)
y_t (XmTextField)
frame3 (XmFrame)

form2 (XmForm)
PixelManipulation (XmLabelGadget)
increase (XmPushButton)
brightness: (XmLabelGadget)
off_t (XmTextField)
scale (XmPushButton)
contrast: (XmLabelGadget)
weight_t (XmTextField)
log (XmPushButton)
log2 (XmPushButton)
log10 (XmPushButton)
inv (XmPushButton)
frame4 (XmFrame)
form3 (XmForm)
DomainManipulation (XmLabelGadget)
FT (XmPushButton)
form4 (XmForm)
Help (XmPushButton)
Dismiss (XmPushButton)
Morphology (XmDialogShell)
pane (XmPanedWindow)
column (XmRowColumn)
frame1 (XmFrame)
form1 (XmForm)
MathematicalMorphology (XmLabelGadget)
Dilation (XmPushButton)
Erosion (XmPushButton)

Opening (XmPushButton)

Closing (XmPushButton)

NotDilate (XmPushButton)

NotErode (XmPushButton)

TopHat (XmPushButton)

BotHat (XmPushButton)

DistT (XmPushButton)

SoftDil (XmPushButton)

SoftEro (XmPushButton)

RecSoftDil (XmPushButton)

RecSoftEro (XmPushButton)

Rank (XmPushButton)

scale1 (XmScale)

LUM (XmPushButton)

LUMSMO (XmPushButton)

LUMSHA (XmPushButton)

scale2 (XmScale)

scale3 (XmScale)

toggle_box1 (XmRowColumn)

BinaryImage (XmToggleButton)

SetOperation (XmToggleButton)

ZeroPadding (XmToggleButton)

Scale (XmToggleButton)

frame2 (XmFrame)

form2 (XmForm)

SEDefinition (XmLabelGadget)

frame3 (XmFrame)

```
                    toggle_box2 (XmRowColumn)
                    UserDef (XmToggleButton)
                    Auto (XmToggleButton)
                    Plus (XmToggleButton)
                    3x3 (XmToggleButton)
                    5x5 (XmToggleButton)
                SEName: (XmLabelGadget)
                name_t (XmTextField)
                x: (XmLabelGadget)
                x_t (XmTextField)
                y: (XmLabelGadget)
                y_t (XmTextField)
                z: (XmLabelGadget)
                z_t (XmTextField)
                BinarySE (XmToggleButton)
        form3 (XmForm)
            Help (XmPushButton)
            Dismiss (XmPushButton)
Noise (XmDialogShell)
    pane (XmPanedWindow)
        frame (XmFrame)
            form1 (XmForm)
                GaussianDistribution (XmLabelGadget)
                scale1 (XmScale)
                scale2 (XmScale)
        form2 (XmForm)
            Cancel (XmPushButton)
```

Ok (XmPushButton)

Bilevel (XmDialogShell)

form1 (XmForm)

scale (XmScale)

Done (XmPushButton)

HalfLevel (XmDialogShell)

form1 (XmForm)

scale (XmScale)

Done (XmPushButton)

Multilevel (XmDialogShell)

form (XmForm)

frame1 (XmFrame)

area1 (XmDrawingArea)

frame2 (XmFrame)

area2 (XmDrawingArea)

scale (XmScale)

Clear (XmPushButton)

Apply (XmPushButton)

Done (XmPushButton)

Image (topLevelShell)

scroll (XmScrolledWindow)

area (XmDrawingArea)

Charview (topLevelShell)

pane (XmPanedWindow)

scroll (XmScrolledWindow)

text (XmText)

form (XmForm)

```
            Help (XmPushButton)
            Dismiss (XmPushButton)
Hist (topLevelShell)
    area (XmDrawingArea)
```

APPENDIX B

SELECTED PROGRAM LISTINGS

A selected set of listings is included to provide an inner look to the application itself. Contact the Computer and Information Science Departmant at NJIT for a complete code set.

ipt.h

```
#include <stdio.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/LabelG.h>
#include <Xm/PushB.h>

#define  PIXEL unsigned char
#define  MY_COLORMAP    0x01
#define  SCALE          0x02
#define  TRUNCATE       0x04
#define  NOTHING        0x08
#define  LINE           80
#define  LONG_LINE      255
#define  DEFAULT_NEIGHBORHOOD 5
#define  DEFAULT_MEAN   0
#define  DEFAULT_STD    10
#define  DEFAULT_THRESHOLD 100
```

```
#ifndef NUM_OF_IMAGES
#define  NUM_OF_IMAGES 6
#endif
#ifndef MAX
#define  MAX(a,b) ((a>b)?(a):(b))
#endif
#ifndef MIN
#define  MIN(a,b) ((a<b)?(a):(b))
#endif

typedef struct
{
 float mean;
 float std;
} gaussian;

typedef struct
{
 int class;
 int qlty;
} imgfmt;

typedef struct
{
 int  imgtype;
 int  nzpad;
 int  lthresh;
 int  uthresh;
```

```
 char sename[LINE];
 int  autose;
 int  sx;
 int  sy;
 int  sz;
 int  setype;
 int  sorf;
 int  rank;
 int  noscale;
} mparameters;

typedef struct
{
 int ffn;
 int d0;
 int d1;
} freqinfo;

Widget gettopshell(Widget);
void   unmap_shell(Widget, Widget);
void   destroy_shell(Widget, Widget);
int    get_current(void);
void   display_image(int);
void   wprint(char *, ...);
void   help_dialog(Widget, int,
                   XmAnyCallbackStruct *);

#ifndef FLAG_IPT
```

```
extern Display  *theDisplay;
extern int      theScreen;
extern Window   theRootWindow;
extern Visual   *theVisual;
extern int      theDepth;
extern Colormap theColormap;
extern Colormap myColormap;
extern GC       theGC;
extern Cursor   watch;

extern unsigned int use[256];
extern int      cmap[256];
extern unsigned int *hist[NUM_OF_IMAGES];
extern int      current;
extern int      image_count;
extern int      mode;
extern XImage   *ximage[NUM_OF_IMAGES];
extern Pixmap   pixmap[NUM_OF_IMAGES];
extern Pixmap   histogram[NUM_OF_IMAGES];
extern Widget   area[NUM_OF_IMAGES];
extern Widget   top_hist[NUM_OF_IMAGES];
extern Widget   top_level;
#endif
```

ipis.c

```
#define  FLAG_IPT
#include "ipt.h"
#include <X11/cursorfont.h>
#include <Xm/MainW.h>
#include <Xm/FileSB.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/PanedW.h>
#include <Xm/ScrolledW.h>
#include <Xm/DialogS.h>
#include <Xm/Frame.h>
#include <Xm/Scale.h>
#include <Xm/ToggleB.h>
#include <stdarg.h>
#include <string.h>

Widget   top_level;     /* Widget for top shell. */
Display  *theDisplay; /* Pointer to the display. */
Screen   *theScreen;   /* Pointer to the screen. */
Window   theRootWindow;       /* Root window ID. */
Visual   *theVisual;   /* Pointer to the visual. */
int      theDepth;          /* The screen depth. */
Colormap theColormap;  /* The standard colormap. */
Colormap myColormap;    /* The private colormap. */
GC       theGC;         /* The graphics context. */
```

```
Cursor   watch;                    /* Busy cursor. */
unsigned int use[256];  /* Amount of grays used. */
int      cmap[256];    /* Index to the std cmap. */
         /* Histograms for images. */
unsigned int *hist[NUM_OF_IMAGES] = {NULL};
int      current;           /* The current image. */
int      image_count = 0;         /* Image count. */
int      mode = TRUNCATE;   /* The program mode. */
XImage   *ximage[NUM_OF_IMAGES];/* Img. structs. */
Pixmap   pixmap[NUM_OF_IMAGES]; /* Img. pixmaps. */
           /* Histogram pixmaps. */
Pixmap   histogram[NUM_OF_IMAGES];
          /* Image drawing areas. */
Widget   area[NUM_OF_IMAGES];
         /* Dialogs for histograms. */
Widget   top_hist[NUM_OF_IMAGES];
         /* Main window text area. */
static   Widget   top_text;
         /* Format info for writes. */
static   imgfmt   format_info = {0, 75};

void main(int argc, char *argv[])
{
 XtAppContext app;        /* Application context. */
           /* Widgets used to in */
 Widget        main_win, menu_bar, sett_menu, rc;
```

```
            /* the main window. */
Widget      curr_menu, widget, scroll, form;
char        i, name[4];
              /* Error */
int         x_error(Display *, XErrorEvent *);
             /* handlers. */
void        xt_error(char *);
void        init_colormap(void);
           /* Widget callbacks. */
void        file_cb(Widget, int);
void        help_cb(Widget, int);
void        sett_cb(Widget, int);
void        pixel_cb(Widget, int);
void        current_cb(Widget, XtPointer,
                    XmRowColumnCallbackStruct *);
void        enhance_dialog(Widget);
void        morpho_dialog(Widget);
void        transform_dialog(Widget);
void        segment_dialog(Widget);
void        stat_scale_dialog(Widget, XtPointer,
                          XmAnyCallbackStruct *);

/* Connect to the X server and execute command  */
/* line options.                                */
 top_level = XtVaAppInitialize(&app, "Ipis",
             NULL, 0, &argc, argv, NULL, NULL);
```

```
/* Set the error handlers.                                    */
 XtAppSetErrorHandler(app, xt_error);
 XtAppSetWarningHandler(app, xt_error);
 XSetErrorHandler(x_error);

/* Init. server dependent global variables.                   */
 theDisplay    = XtDisplay(top_level);
 theScreen     = XtScreen(top_level);
 theRootWindow = RootWindowOfScreen(theScreen);
 theVisual     = DefaultVisualOfScreen(theScreen);
 theDepth      = DefaultDepthOfScreen(theScreen);
 theColormap   = DefaultColormapOfScreen(theScreen);
 theGC         = DefaultGCOfScreen(theScreen);
 watch         = XCreateFontCursor(theDisplay,
                                   XC_watch);

/* Create the main window widget.                             */
 main_win  = XtVaCreateManagedWidget("main",
             xmMainWindowWidgetClass, top_level,
             NULL);

/* Create the menubar and its pulldown and                    */
/* pullright menus.                                           */
 menu_bar  = XmVaCreateSimpleMenuBar(
             main_win, "menubar",
             XmVaCASCADEBUTTON, NULL, NULL,
             XmVaCASCADEBUTTON, NULL, NULL,
             XmVaCASCADEBUTTON, NULL, NULL,
```

```
                XmVaCASCADEBUTTON, NULL, NULL,
                NULL);
widget     = XtNameToWidget(menu_bar, "button_3");
XtVaSetValues(menu_bar, XmNmenuHelpWidget,
                widget, NULL);

XmVaCreateSimplePulldownMenu(
                menu_bar, "filemenu", 0, file_cb,
                XmVaPUSHBUTTON, NULL, NULL, NULL,
                                NULL,
                XmVaPUSHBUTTON, NULL, NULL, NULL,
                                NULL,
                XmVaSEPARATOR,
                XmVaPUSHBUTTON, NULL, NULL, NULL,
                                NULL,
                NULL);

sett_menu = XmVaCreateSimplePulldownMenu(
                menu_bar, "settmenu", 1, sett_cb,
                XmVaCASCADEBUTTON, NULL, NULL,
                XmVaTOGGLEBUTTON,  NULL, NULL, NULL,
                                   NULL,
                NULL);
XmVaCreateSimplePulldownMenu(
                sett_menu, "pullmenu", 0, pixel_cb,
                XmVaRADIOBUTTON, NULL, NULL, NULL,
                                 NULL,
                XmVaRADIOBUTTON, NULL, NULL, NULL,
```

```
                                NULL,
                XmVaRADIOBUTTON, NULL, NULL, NULL,
                                NULL,
                XmNradioBehavior, True,
                XmNradioAlwaysOne, True,
                NULL);

 curr_menu = XmVaCreateSimplePulldownMenu(
                menu_bar, "curr_menu", 2, NULL,
                NULL);
 XtAddCallback(curr_menu,
                XmNentryCallback, current_cb, NULL);
/* Add dynamically a menu option for each       */
/* image that can be displayed.                 */
 for (i=0; i < NUM_OF_IMAGES; i++)
 {
  sprintf(name, "%d", i+1);
  XtVaCreateManagedWidget(name,
                xmPushButtonWidgetClass, curr_menu,
                XmNuserData, i,
                NULL);
 }

 XmVaCreateSimplePulldownMenu(
                menu_bar, "help_menu", 3, help_cb,
                XmVaPUSHBUTTON, NULL, NULL, NULL,
                                NULL,
                XmVaPUSHBUTTON, NULL, NULL, NULL,
```

```
                                    NULL,
            NULL);

XtManageChild(menu_bar);

/* Create Message Area.                                    */
rc        = XtVaCreateManagedWidget(
            "column", xmRowColumnWidgetClass,
            main_win,
            NULL);
scroll    = XtVaCreateManagedWidget(
            "scroll", xmScrolledWindowWidgetClass,
            rc,
            XmNscrollingPolicy,
            XmAPPLICATION_DEFINED,
            XmNvisualPolicy,
            XmVARIABLE,
            XmNscrollBarDisplayPolicy, XmSTATIC,
            XmNshadowThickness,        0,
            NULL);
top_text = XtVaCreateManagedWidget(
            "text", xmTextWidgetClass, scroll,
            XmNeditable,               False,
            XmNeditMode,   XmMULTI_LINE_EDIT,
            XmNscrollHorizontal,       False,
            XmNwordWrap,               True,
            XmNautoShowCursorPosition, True,
            XmNcursorPositionVisible,  False,
```

```
            NULL);

/* Create the window's push buttons.                    */
 form   = XtVaCreateManagedWidget(
            "form", xmFormWidgetClass, rc,
            XmNfractionBase, 4,
            NULL);
 widget = XtVaCreateManagedWidget(
            "Enhancement", xmPushButtonWidgetClass,
            form,
            XmNtopAttachment,    XmATTACH_POSITION,
            XmNtopPosition,      0,
            XmNbottomAttachment, XmATTACH_POSITION,
            XmNbottomPosition,   2,
            XmNleftAttachment,   XmATTACH_POSITION,
            XmNleftPosition,     0,
            XmNrightAttachment,  XmATTACH_POSITION,
            XmNrightPosition,    1,
            NULL);
XtAddCallback(widget, XmNactivateCallback,
            enhance_dialog, NULL);
 widget = XtVaCreateManagedWidget("Restoration",
            xmPushButtonWidgetClass,
            form,
            XmNtopAttachment,    XmATTACH_POSITION,
            XmNtopPosition,      0,
            XmNbottomAttachment, XmATTACH_POSITION,
```

```
            XmNbottomPosition,    2,
            XmNleftAttachment,    XmATTACH_POSITION,
            XmNleftPosition,      1,
            XmNrightAttachment,   XmATTACH_POSITION,
            XmNrightPosition,     2,
            NULL);
    widget = XtVaCreateManagedWidget(
            "Segmentation", xmPushButtonWidgetClass,
            form,
            XmNtopAttachment,     XmATTACH_POSITION,
            XmNtopPosition,       0,
            XmNbottomAttachment,  XmATTACH_POSITION,
            XmNbottomPosition,    2,
            XmNleftAttachment,    XmATTACH_POSITION,
            XmNleftPosition,      2,
            XmNrightAttachment,   XmATTACH_POSITION,
            XmNrightPosition,     3,
            NULL);
    XtAddCallback(widget, XmNactivateCallback,
            segment_dialog, NULL);
    widget = XtVaCreateManagedWidget(
            "Transformation", xmPushButtonWidgetClass,
            form,
            XmNtopAttachment,     XmATTACH_POSITION,
            XmNtopPosition,       0,
            XmNbottomAttachment,  XmATTACH_POSITION,
            XmNbottomPosition,    2,
```

```
            XmNleftAttachment,    XmATTACH_POSITION,
            XmNleftPosition,      3,
            XmNrightAttachment,   XmATTACH_POSITION,
            XmNrightPosition,     4,
            NULL);
    XtAddCallback(widget, XmNactivateCallback,
            transform_dialog, NULL);
    widget = XtVaCreateManagedWidget("Analysis",
            xmPushButtonWidgetClass, form,
            XmNtopAttachment,     XmATTACH_POSITION,
            XmNtopPosition,       2,
            XmNbottomAttachment,  XmATTACH_POSITION,
            XmNbottomPosition,    4,
            XmNleftAttachment,    XmATTACH_POSITION,
            XmNleftPosition,      0,
            XmNrightAttachment,   XmATTACH_POSITION,
            XmNrightPosition,     1,
            NULL);
    widget = XtVaCreateManagedWidget("Matching",
            xmPushButtonWidgetClass, form,
            XmNtopAttachment,     XmATTACH_POSITION,
            XmNtopPosition,       2,
            XmNbottomAttachment,  XmATTACH_POSITION,
            XmNbottomPosition,    4,
            XmNleftAttachment,    XmATTACH_POSITION,
            XmNleftPosition,      1,
            XmNrightAttachment,   XmATTACH_POSITION,
```

```
            XmNrightPosition,     2,
            NULL);
widget = XtVaCreateManagedWidget("Morphology",
            xmPushButtonWidgetClass,
            form,
            XmNtopAttachment,     XmATTACH_POSITION,
            XmNtopPosition,       2,
            XmNbottomAttachment,  XmATTACH_POSITION,
            XmNbottomPosition,    4,
            XmNleftAttachment,    XmATTACH_POSITION,
            XmNleftPosition,      2,
            XmNrightAttachment,   XmATTACH_POSITION,
            XmNrightPosition,     3,
            NULL);
XtAddCallback(widget, XmNactivateCallback,
            morpho_dialog, NULL);
widget = XtVaCreateManagedWidget("Degradation",
            xmPushButtonWidgetClass,
            form,
            XmNtopAttachment,     XmATTACH_POSITION,
            XmNtopPosition,       2,
            XmNbottomAttachment,  XmATTACH_POSITION,
            XmNbottomPosition,    4,
            XmNleftAttachment,    XmATTACH_POSITION,
            XmNleftPosition,      3,
            XmNrightAttachment,   XmATTACH_POSITION,
            XmNrightPosition,     4,
```

```
            NULL);
 XtAddCallback(widget, XmNactivateCallback,
            stat_scale_dialog, NULL);

/* Set the main window's widgets and make them  */
/* visible.                                     */
 XtVaSetValues(main_win,
               XmNmenuBar, menu_bar,
               XmNworkWindow, rc,
               NULL);
 XtRealizeWidget(top_level);

/* Initialize the colormap with a minimun of    */
/* gray levels.                                 */
 init_colormap();

/* Loop waiting for events.                     */
 XtAppMainLoop(app);
}

void file_cb(Widget w, int n)
{
 void open_file(void);
 void write_file(void);

 switch(n)
 {
  case 0 : open_file();
           break;
```

```
  case 1 : write_file();
           break;
  case 2 : XtCloseDisplay(theDisplay);
           exit(0);
 }
}

void sett_cb(Widget w, int n)
{
 switch(n)
 {
  case 0: break;
 /* Can never happen. Pullright menu is called  */
 /* instead.                                    */
  case 1: mode ^= MY_COLORMAP; break;
 /* Toggle colormap mode.                       */
 }
}

void pixel_cb(Widget w, int n)
{
 switch(n)
 {
  case 0:
  mode &= MY_COLORMAP; mode |= SCALE;     break;
  case 1:
  mode &= MY_COLORMAP; mode |= TRUNCATE; break;
  case 2:
```

```
    mode &= MY_COLORMAP; mode |= NOTHING;  break;
  }
 }

void current_cb(Widget w, XtPointer client_data,
                XmRowColumnCallbackStruct *cbs)
{
 int n;

/* Find out which button was selected from the  */
/* pulldown menu.                               */
 XtVaGetValues(cbs->widget, XmNuserData, &n, NULL);

/* if it exists, set it the current image.      */
 if (ximage[n])
  current = n;

}

void help_cb(Widget w, int n)
{
 void about_dialog(Widget);

 if (n)
  about_dialog(w);
 else
  help_dialog(w, 0, NULL);
}

void open_file(void)
```

```
{
 static Widget dialog;
 void input_image(Widget, XtPointer,
                  XmFileSelectionBoxCallbackStruct *);

/* If the open_file dialog has been created     */
/*                    before then skip this code. */
 if (!dialog)
 {
  dialog = XmCreateFileSelectionDialog(top_level,
           "Open", NULL, 0);
  XtAddCallback(dialog, XmNokCallback,
                input_image, NULL);
  XtAddCallback(dialog, XmNhelpCallback,
                help_dialog, 6);
  XtAddCallback(dialog, XmNcancelCallback,
                XtUnmanageChild, NULL);
 }
/* Make sure that the dialog is visible.        */
 XtManageChild(dialog);
 XtPopup(XtParent(dialog), XtGrabNone);
}

void write_file(void)
{
 Widget frame, rc, box, toggle, scale;
 static Widget dialog, selbox;
 void set_format(Widget, int,
```

```
                XmToggleButtonCallbackStruct *);
 void set_quality(Widget, XtPointer,
                XmScaleCallbackStruct *);
 void write_image(Widget, XtPointer,
                XmFileSelectionBoxCallbackStruct *);

/* If write_file dialog has been created before */
/* then make it visible.                        */
 if (dialog)
 {
  XtManageChild(selbox);
  XtManageChild(dialog);
  XtPopup(dialog, XtGrabNone);
  return;
 }
/* If not, create dialog.                       */
 dialog = XtVaCreatePopupShell("Write",
          xmDialogShellWidgetClass,
          top_level,
          XmNdeleteResponse, XmUNMAP,
          NULL);
 selbox = XtVaCreateWidget("selbox",
          xmFileSelectionBoxWidgetClass, dialog,
          XmNautoUnmanage,   True,
          NULL);
 frame  = XtVaCreateManagedWidget("frame",
          xmFrameWidgetClass, selbox,
```

```
            XmNshadowType,      XmSHADOW_ETCHED_IN,
            NULL);
rc      = XtVaCreateManagedWidget("column",
            xmRowColumnWidgetClass, frame,
            NULL);
box     = XtVaCreateManagedWidget("toggle_box",
            xmRowColumnWidgetClass, rc,
            XmNradioBehavior,   True,
            XmNradioAlwaysOne,  True,
            XmNpacking,         XmPACK_COLUMN,
            XmNnumColumns,      3,
            NULL);
toggle  = XtVaCreateManagedWidget("RawPGM",
            xmToggleButtonWidgetClass, box,
            XmNset,             True,
            NULL);
XtAddCallback(toggle, XmNvalueChangedCallback,
            set_format, 0);
toggle  = XtVaCreateManagedWidget("AsciiPGM",
            xmToggleButtonWidgetClass,
            box, NULL);
XtAddCallback(toggle, XmNvalueChangedCallback,
            set_format, 1);
toggle  = XtVaCreateManagedWidget("GIF",
            xmToggleButtonWidgetClass, box,
            NULL);
XtAddCallback(toggle, XmNvalueChangedCallback,
```

```
            set_format, 2);
toggle = XtVaCreateManagedWidget("JFIF",
            xmToggleButtonWidgetClass, box,
            NULL);
XtAddCallback(toggle, XmNvalueChangedCallback,
            set_format, 3);
toggle = XtVaCreateManagedWidget("PostScript",
            xmToggleButtonWidgetClass,
            box, NULL);
XtAddCallback(toggle, XmNvalueChangedCallback,
            set_format, 4);
toggle = XtVaCreateManagedWidget("TIFF",
            xmToggleButtonWidgetClass, box,
            NULL);
XtAddCallback(toggle, XmNvalueChangedCallback,
            set_format, 5);
scale  = XtVaCreateManagedWidget("scale",
            xmScaleWidgetClass, rc,
            XmNorientation,     XmHORIZONTAL,
            XmNminimum,         5,
            XmNmaximum,         95,
            XmNvalue,           format_info.qlty,
            XmNshowValue,       True,
            NULL);
XtAddCallback(scale, XmNvalueChangedCallback,
            set_quality, NULL);
XtManageChild(selbox);
```

```
 XtAddCallback(selbox, XmNokCallback,
          write_image, &format_info);
 XtAddCallback(selbox, XmNhelpCallback,
          help_dialog, 7);
 XtAddCallback(selbox, XmNcancelCallback,
          XtUnmanageChild,
          XtParent(selbox));
 XtManageChild(dialog);
 XtPopup(dialog, XtGrabNone);
}

#include "bitmaps/tower"
#include "bitmaps/njit"

void about_dialog(Widget w)
{
 Pixmap         bitmap;
 Pixel          fg, bg;
 static Widget  dialog, pane;
 Widget         form, label1, label2, label3;
 XmFontList     list;
 XFontStruct    *font;

/* If dialog has been created them make it      */
/* visible.                                     */
 if (dialog)
 {
  XtManageChild(pane);
```

```
    XtManageChild(dialog);
    XtPopup(dialog, XtGrabNone);
    return;
  }
/* If not, create dialog.                                    */
  dialog = XtVaCreatePopupShell("About",
           xmDialogShellWidgetClass,
           top_level,
           XmNdeleteResponse, XmUNMAP,
           NULL);
  pane   = XtVaCreateWidget("pane",
           xmPanedWindowWidgetClass, dialog,
           XmNsashWidth,      1,
           XmNsashHeight,     1,
           NULL);
  form   = XtVaCreateManagedWidget("form1",
           xmFormWidgetClass, pane,
           XmNfractionBase,   19,
           NULL);
  XtVaGetValues(form,
           XmNforeground,     &fg,
           XmNbackground,     &bg,
           NULL);
/* Create monochrome a pixmap of depth 8.                    */
  bitmap = XCreatePixmapFromBitmapData(
           theDisplay, theRootWindow,
           tower_bits, tower_width,
```

```
            tower_height, 0, bg, 8);
label1 = XtVaCreateManagedWidget("tower",
            xmLabelGadgetClass, form,
            XmNtopAttachment,   XmATTACH_FORM,
            XmNleftAttachment,  XmATTACH_FORM,
            XmNlabelType,       XmPIXMAP,
            XmNlabelPixmap,     bitmap,
            NULL);
label2 = XtVaCreateManagedWidget("name",
            xmLabelGadgetClass, form,
            XmNtopAttachment,   XmATTACH_POSITION,
            XmNtopPosition,     3,
            XmNleftAttachment,  XmATTACH_WIDGET,
            XmNleftWidget,      label1,
            XmNlabelString,
            XmStringCreateSimple(
            "IPIS - Version 0.5"),
            NULL);
label3 = XtVaCreateManagedWidget("author",
            xmLabelGadgetClass, form,
            XmNtopAttachment,   XmATTACH_WIDGET,
            XmNtopWidget,       label2,
            XmNleftAttachment,  XmATTACH_WIDGET,
            XmNleftWidget,      label1,
            XmNalignment,       XmALIGNMENT_BEGINNING,
            XmNlabelString,
            XmStringCreateLtoR(
```

```
            "Developed by Eduardo Morales\nat the
            Computer Vision Laboratory.", "charset"),
            NULL);
label2 = XtVaCreateManagedWidget("copyright",
            xmLabelGadgetClass, form,
            XmNtopAttachment,   XmATTACH_WIDGET,
            XmNtopWidget,       label1,
            XmNtopOffset,       5,
            XmNleftAttachment, XmATTACH_FORM,
            XmNlabelString,
            XmStringCreateSimple(
            "Copyright 1993 New Jersey Institute of
            Technology"),
            NULL);
form   = XtVaCreateManagedWidget("form2",
            xmFormWidgetClass, pane,
            XmNfractionBase, 22,
            NULL);
bitmap = XCreatePixmapFromBitmapData(
            theDisplay, theRootWindow, njit_bits,
            njit_width, njit_height, 0, bg, 8);
label1 = XtVaCreateManagedWidget("logo",
            xmLabelGadgetClass, form,
            XmNtopAttachment,   XmATTACH_FORM,
            XmNleftAttachment, XmATTACH_POSITION,
            XmNleftPosition,    3,
            XmNlabelType,       XmPIXMAP,
```

```
            XmNlabelPixmap,     bitmap,
            NULL);
 font    = XLoadQueryFont(theDisplay,
            "-adobe-times-bold-i-normal--14-140-75-
            75-p-77-iso8859-1");
 list    = XmFontListCreate(font, "charset1");
 label2  = XtVaCreateManagedWidget("caption",
            xmLabelGadgetClass, form,
            XmNtopAttachment,   XmATTACH_POSITION,
            XmNtopPosition,     3,
            XmNleftAttachment,  XmATTACH_WIDGET,
            XmNleftWidget,      label1,
            XmNfontList,        list,
            XmNlabelString,
            XmStringCreateSimple("A Public Research
            University."),
            NULL);

/* According to the Motif Style Guide,          */
/* action area must not be resized.             */
 {
  Dimension h;
  XtVaGetValues(label1, XmNheight, &h, NULL);
  XtVaSetValues(form,
            XmNpaneMaximum, h,
            XmNpaneMinimum, h,
            NULL);
```

```
 }
 XtManageChild(pane);
 XtManageChild(dialog);
 XtPopup(dialog, XtGrabNone);
}

void set_format(Widget w, int num,
                XmToggleButtonCallbackStruct *cbs)
{
 if (cbs->set)
  format_info.class = num;
}

void set_quality(Widget w, XtPointer client_data,
                 XmScaleCallbackStruct *cbs)
{
 format_info.qlty = cbs->value;
}

Widget gettopshell(Widget w)
{
 while (w && !XtIsWMShell(w))
  w = XtParent(w);
 return w;
}

void unmap_shell(Widget w, Widget shell)
{
 XtUnmapWidget(shell);
```

```
}

void destroy_shell(Widget w, Widget shell)
{
 XtDestroyWidget(shell);
}

/* This function works like printf but instead   */
/* of printing to the stdout it prints           */
/* the message in a text widget.                 */
void wprint(char *fmt, ...)
{
 char msgbuf[LONG_LINE];
 static XmTextPosition wpr_position;
 va_list args;

 va_start(args, fmt);
 vsprintf(msgbuf, fmt, args);
 va_end(args);

 XmTextInsert(top_text, wpr_position, msgbuf);
 wpr_position += strlen(msgbuf);
 XtVaSetValues(top_text,
 XmNcursorPosition, wpr_position, NULL);
 XmTextShowPosition(top_text, wpr_position);
}

int x_error(Display *dpy, XErrorEvent *err)
{
```

```
 char buf[LINE];

 XGetErrorText(dpy, err->error_code, buf,
               sizeof(buf));
 wprint("X Error: %s\n", buf);
 return 0;
}

void xt_error(char *mess)
{
/* We can disregard the "Cannot allocate..."    */
/* error message because it's granted that we   */
/* will run out of colormap cells (only 256     */
/* available).                                  */
 if (strstr(mess,
     "Cannot allocate colormap entry for") == NULL)
  wprint("Xt Error: %s\n", mess);
}
```

degradeD.c

```
#include "ipt.h"
#include <Xm/DialogS.h>
#include <Xm/RowColumn.h>
#include <Xm/Frame.h>
#include <Xm/PanedW.h>
#include <Xm/Scale.h>

/* Contains the mean and std.                              */
static gaussian distrib;

void stat_scale_dialog(Widget w, XtPointer
             client_data, XmAnyCallbackStruct *cbs)
{
 static Widget dialog;
 Widget pane, frame, form, label, scale1, scale2;
 Widget action;
 void   gauss(Widget, XtPointer,
              XmAnyCallbackStruct *);
 void   set_mean(Widget, XtPointer,
              XmScaleCallbackStruct *);
 void   set_std(Widget, XtPointer,
              XmScaleCallbackStruct *);

/* Check if it has been created, if true then   */
/* make it visible.                             */
 if (dialog)
 {
```

```
  XtMapWidget(dialog);
  return;
 }

/* Create the dialog.                                    */
 dialog = XtVaCreatePopupShell("Noise",
          xmDialogShellWidgetClass,
          gettopshell(w),
          XmNdeleteResponse,  XmDO_NOTHING,
          NULL);
 pane   = XtVaCreateWidget("pane",
          xmPanedWindowWidgetClass, dialog,
          XmNsashWidth,        1,
          XmNsashHeight,       1,
          NULL);
 frame  = XtVaCreateManagedWidget("frame",
          xmFrameWidgetClass,  pane,
          XmNshadowType,       XmSHADOW_ETCHED_IN,
          NULL);
 form   = XtVaCreateManagedWidget("form1",
          xmFormWidgetClass, frame,
          XmNfractionBase,     19,
          NULL);
 label  = XtVaCreateManagedWidget(
         "GaussianDistribution",
          xmLabelGadgetClass, form,
          XmNtopAttachment,    XmATTACH_FORM,
```

```
            XmNtopOffset,        2,
            XmNleftAttachment,   XmATTACH_FORM,
            XmNleftOffset,       5,
            NULL);
distrib.mean = DEFAULT_MEAN / 10.0;
scale1 = XtVaCreateManagedWidget("scale1",
            xmScaleWidgetClass, form,
            XmNorientation,      XmHORIZONTAL,
            XmNdecimalPoints,    1,
            XmNvalue,            DEFAULT_MEAN,
            XmNshowValue,        True,
            XmNtopAttachment,    XmATTACH_WIDGET,
            XmNtopWidget,        label,
            XmNleftAttachment,   XmATTACH_FORM,
            XmNrightAttachment,  XmATTACH_FORM,
            NULL);
XtAddCallback(scale1, XmNvalueChangedCallback,
            set_mean, NULL);
distrib.std = DEFAULT_STD / 10.0;
scale2 = XtVaCreateManagedWidget("scale2",
            xmScaleWidgetClass, form,
            XmNorientation,      XmHORIZONTAL,
            XmNdecimalPoints,    1,
            XmNvalue,            DEFAULT_STD,
            XmNshowValue,        True,
            XmNtopAttachment,    XmATTACH_WIDGET,
            XmNtopWidget,        scale1,
```

```
            XmNleftAttachment,   XmATTACH_FORM,
            XmNrightAttachment,  XmATTACH_FORM,
            NULL);
XtAddCallback(scale2, XmNvalueChangedCallback,
            set_std, NULL);
form    = XtVaCreateManagedWidget("form2",
            xmFormWidgetClass, pane,
            XmNfractionBase,     5,
            NULL);
action = XtVaCreateManagedWidget("Cancel",
            xmPushButtonWidgetClass, form,
            XmNtopAttachment,     XmATTACH_FORM,
            XmNbottomAttachment,  XmATTACH_FORM,
            XmNleftAttachment,    XmATTACH_POSITION,
            XmNleftPosition,      3,
            XmNrightAttachment ,  XmATTACH_POSITION,
            XmNrightPosition,     4,
            XmNshowAsDefault,     False,
            XmNdefaultButtonShadowThickness, 1,
            NULL);
XtAddCallback(action, XmNactivateCallback,
            unmap_shell, dialog);
action = XtVaCreateManagedWidget("Ok",
            xmPushButtonWidgetClass, form,
            XmNtopAttachment,     XmATTACH_FORM,
            XmNbottomAttachment,  XmATTACH_FORM,
            XmNleftAttachment,    XmATTACH_POSITION,
```

```
            XmNleftPosition,      1,
            XmNrightAttachment,   XmATTACH_POSITION,
            XmNrightPosition,     2,
            XmNshowAsDefault,     True,
            XmNdefaultButtonShadowThickness, 1,
            NULL);
 XtAddCallback(action, XmNactivateCallback, gauss,
            (XtPointer) &distrib);

/* According to the Motif Style Guide,          */
/* action area must not be resized.             */
 {
  Dimension h;
  XtVaGetValues(action, XmNheight, &h, NULL);
  XtVaSetValues(form,
            XmNpaneMaximum, h,
            XmNpaneMinimum, h,
            NULL);
 }
 XtManageChild(pane);
 XtPopup(dialog, XtGrabNone);
}

/* Update the mean every time the value is      */
/* changed.                                     */
void set_mean(Widget w, XtPointer client_data,
              XmScaleCallbackStruct *cbs)
{
```

```
 distrib.mean = cbs->value / 10.0;
}

/* Update the standard deviation every time the  */
/* value is changed.                             */
void set_std(Widget w, XtPointer client_data,
             XmScaleCallbackStruct *cbs)
{
 distrib.std = cbs->value / 10.0;
}
```

image.c

```
#include "ipt.h"
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <Xm/DrawingA.h>
#include <Xm/MwmUtil.h>
#include <Xm/ScrolledW.h>

int     allc[256] = {0};
static Widget top[NUM_OF_IMAGES];

void input_image(Widget w, XtPointer client_data,
           XmFileSelectionBoxCallbackStruct *cbs)
{
 char  *filename;
 char  *dire;
 FILE  *file;
 PIXEL buf[5];
 int    flag;
 int    read_JPEG(FILE *);
 int    read_RPGM(FILE *);
 int    read_APGM(FILE *);
 int    read_TIFF(char *);

 XDefineCursor(theDisplay, XtWindow(top_level),
               watch);
 XFlush(theDisplay);
```

```
/* Get filename to open from the                    */
/* FileSelectionBox structure.                      */
 XmStringGetLtoR(cbs->value,
                 XmSTRING_DEFAULT_CHARSET, &filename);

 if (!(file = fopen(filename, "rb")))
 {
  wprint("Error opening input file: %s\n",
          filename);
  return;
 }

/* Print filename on the screen, but first get      */
/* rid of the pathname.                             */
 XmStringGetLtoR(cbs->dir,
                   XmSTRING_DEFAULT_CHARSET, &dire);
 wprint("Reading file %s\n",
         filename+strlen(dire));

 current = get_current();

/* Read the file's first two characters to find     */
/* its type.                                        */
 fread(buf, sizeof(char), 2, file);
/* Rewind the file. Some library functions use      */
/* the first two chars.                             */
 rewind(file);
```

```
/* Determine file type and call appropiate      */
/* function.                                    */
 if (*buf == 0xff || *buf == 0xd8)
  flag = read_JPEG(file);
 else if (*buf == 'P' && *(buf+1) == '5')
  flag = read_RPGM(file);
 else if (*buf == 'P' && *(buf+1) == '2')
  flag = read_APGM(file);
 else if (*buf == 'I' || *buf == 'M')
 {
/* read_TIFF requires a closed file.            */
  fclose(file);
  flag = read_TIFF(filename);
 }
 else
 {
  fclose(file);
  wprint("Error: Unknown file format.\n");
  return;
 }

/* Check if there were any errors. Don't worry  */
/* about messages, error handlers should have   */
/* taken care of that.                          */
 if (!flag)
  return;

 display_image(current);
```

```
/* Unmap FileSelection dialog.                          */
 XtUnmanageChild(w);
 fclose(file);
 XtFree(filename);
 XtFree(dire);
 XUndefineCursor(theDisplay, XtWindow(top_level));
}

void display_image(int num)
{
 int       x, y, ix, iy;
 char      name[9];
 char      *img;
 PIXEL     *temp, *t;
 XImage    *xtemp;
 Widget    sw;
 Colormap colormap;
 void      redraw(Widget, XtPointer,
           XmDrawingAreaCallbackStruct *);
 void      input_cb(Widget, XtPointer,
           XmDrawingAreaCallbackStruct *);
 void      destroy(Widget, XtPointer,
           XmDrawingAreaCallbackStruct *);
 void      set_colormap(int);

/* Select appropiate colormap.                          */
 if (mode & MY_COLORMAP)
  colormap = myColormap;
```

```
else
{
/* If standard colormap is used, try to          */
/* allocate the new gray levels used by the      */
/* current image.                                */
 set_colormap(num);
 colormap = theColormap;
}

/* Create shell to place the image.              */
 sprintf(name, "image-%d", num+1);
 top[num] = XtVaAppCreateShell("Image", "Image",
            topLevelShellWidgetClass, theDisplay,
            XmNmwmDecorations,
            MWM_DECOR_RESIZEH | MWM_DECOR_TITLE,
            XmNtitle,              name,
            XmNwidth,
            ximage[num]->width + 4,
            XmNheight,
            ximage[num]->height + 4,
            XmNcolormap,           colormap,
            NULL);
 sw       = XtVaCreateManagedWidget("scroll",
            xmScrolledWindowWidgetClass, top[num],
            XmNscrollingPolicy, XmAUTOMATIC,
            XmNscrollBarDisplayPolicy, XmAS_NEEDED,
            NULL);
```

```
area[num]= XtVaCreateManagedWidget("area",
           xmDrawingAreaWidgetClass, sw,
           XmNuserData,          num,
           XmNwidth,             ximage[num]->width,
           XmNheight,
           ximage[num]->height,
           NULL);
XtAddCallback(area[num],
              XmNexposeCallback, redraw, NULL);
XtAddCallback(area[num],
              XmNinputCallback, input_cb, NULL);
XtAddCallback(area[num],
              XmNdestroyCallback, destroy, NULL);

/* Allocate a pixmap to place the image.          */
pixmap[num] = XCreatePixmap(theDisplay,
              theRootWindow,  ximage[num]->width,
              ximage[num]->height, theDepth);

if (mode & MY_COLORMAP)
{
/* If using a private colormap, there is no      */
/* need to map the pixel values. Just place      */
/* the image in memory.                          */
 XPutImage(theDisplay, pixmap[num], theGC,
           ximage[num], 0, 0, 0, 0,
           ximage[num]->width,
           ximage[num]->height);
```

```
}
else
{
/* If using a standard colormap, the image     */
/* needs to be remapped to point to the        */
/* correct colormap position. Gray lavels are  */
/* not always allocated in order, also there   */
/* are some colormap cells being used by other */
/* programs in the display.                    */
 ix    = ximage[num]->width;
 iy    = ximage[num]->height;
 img   = (PIXEL *)ximage[num]->data;
/* Notice that we create a temporary image so  */
/* that we do not alter the original.          */
 temp = t = (PIXEL *)XtCalloc(ix*iy,
              sizeof(PIXEL));
 for (y=0; y < iy; y++)
  for (x=0; x < ix; x++)
*(t++) = (PIXEL)cmap[(PIXEL)(*(img++))];
 xtemp = XCreateImage(theDisplay, theVisual, 8,
              ZPixmap,
              0, (char *) temp, ix, iy, 8, 0);
 xtemp->byte_order       = MSBFirst;
 xtemp->bitmap_bit_order = MSBFirst;

 XPutImage(theDisplay, pixmap[num], theGC, xtemp,
              0, 0, 0, 0, ix, iy);
```

```
  XDestroyImage(xtemp);
 }

 XtRealizeWidget(top[num]);
/* One more image in the display.                    */
 image_count++;
}

void write_image(Widget w, XtPointer client_data,
             XmFileSelectionBoxCallbackStruct *cbs)
{
 char    *filename, *dire;
 imgfmt  *format;
 int     write_JPEG(char *, int);
 int     write_RPGM(char *);
 int     write_APGM(char *);
 int     write_PS(char *);
 int     write_TIFF(char *);

 if (!image_count)
 {
  wprint("No image present.\n");
  return;
 }

 XDefineCursor(theDisplay, XtWindow(top_level),
              watch);
 XFlush(theDisplay);
```

```
 XmStringGetLtoR(cbs->value,
                 XmSTRING_DEFAULT_CHARSET, &filename);

 XmStringGetLtoR(cbs->dir,
                 XmSTRING_DEFAULT_CHARSET, &dire);
 wprint("Writing file %s\n",
                 filename+strlen(dire));

/* Retreive the fileformat from the client data.*/
 format = (imgfmt *)client_data;

/* Choose the corresponding function to format. */
 switch(format->class)
 {
  case 0: write_RPGM(filename); break;
  case 1: write_APGM(filename); break;
  case 2: break; /* Not yet implemented. */
  case 3: write_JPEG(filename, format->qlty);break;
  case 4: write_PS(filename); break;
  case 5: write_TIFF(filename);
 }
 XUndefineCursor(theDisplay, XtWindow(top_level));
}

void set_colormap(int n)
{
 int i;
```

```
int count=0;      /* Number of gray levels used. */
int close=0;      /* Number of closely allocated */
XColor gray;

for (i=0; i < 256; i++)
{
 if (hist[n][i]) /* if the grey level is used...*/
 {
  count++;
  if (!allc[i])       /* if not yet allocated...*/
  {
   gray.red   = (unsigned short)(i * 257);
   gray.green = (unsigned short)(i * 257);
   gray.blue  = (unsigned short)(i * 257);
   gray.flags = DoRed | DoGreen | DoBlue;
/* Allocate color in standard colormap.       */
   if (!XAllocColor(theDisplay, theColormap,
                    &gray))
   {
/* If allocation failed. Allocate closest    */
/* gray level.                               */
    if (i % 3 == 1)
     cmap[i] = cmap[i-1];
    else
     cmap[i] = cmap[i+1];
    close++;
   }
```

```
    else
    {
  /* Success. Do not forget to keep the        */
  /* colormap position.                        */
     cmap[i] = gray.pixel;
     allc[i] = 1;
    }
   }
  }
 }
 wprint("%d gray levels. Allocated %d true,
        %d close.\n",
        count, count-close, close);
}

/* Initialize colormaps so there are always gray */
/* levels to work with.                          */
void init_colormap(void)
{
 int i, count=0, flag=0;
 XColor colors[256], gray;

/* Allocate private colormap. All gray levels    */
/* allocated in order starting from the first    */
/* colormap cell.                                */
 for (i=0; i < 256; i++)
 {
  colors[i].pixel = i;
```

```
 colors[i].red   = (unsigned short)(i * 257);
 colors[i].green = (unsigned short)(i * 257);
 colors[i].blue  = (unsigned short)(i * 257);
 colors[i].flags = DoRed | DoGreen | DoBlue;
}
myColormap = XCreateColormap(theDisplay,
             theRootWindow,
             theVisual, AllocAll);
XStoreColors(theDisplay, myColormap, colors, 256);

/* Allocate a minimum of 83 gray levels to work */
/* with.                                        */
for (i=0; i < 256; i += 3)
{
 gray.red   = (unsigned short)(i * 257);
 gray.green = (unsigned short)(i * 257);
 gray.blue  = (unsigned short)(i * 257);
 gray.flags = DoRed | DoGreen | DoBlue;
 if (!XAllocColor(theDisplay, theColormap, &gray))
  flag = 1;
 else
 {
/* Success. Do not forget to keep the colormap*/
/* position.                                  */
  count++;
  cmap[i] = gray.pixel;
  allc[i] = 1;
```

```
  }
 }

/* if less than 83 gray levels were allocated,  */
/* warn the user.                               */
 if (flag)
 {
  wprint("Colormap to crowded. Could only allocate
          %2d gray lev", count);
  wprint("els.\nRecomend to quit some applications
          to free colormap cell");
  wprint("s\nand reinit colormap, or to continue
          by using a private colo");
  wprint("rmap.\nFailure to do so may result in
          undesireble effects.\n");
 }
}

/* Redraw the image every time there is an      */
/* expose event.                                */
void redraw(Widget w, XtPointer client_data,
            XmDrawingAreaCallbackStruct *cbs)
{
 int n;

/* Retreive the image number from the drawing   */
/* widget.                                      */
 XtVaGetValues(w, XmNuserData, &n, NULL);
```

```
 XCopyArea(cbs->event->xexpose.display, pixmap[n],
           cbs->window, theGC,
           0, 0, ximage[n]->width,
           ximage[n]->height, 0, 0);
}

/* Process input events from the image area.      */
void input_cb(Widget w, XtPointer client_data,
              XmDrawingAreaCallbackStruct *cbs)
{
 int         n;
 static int x[NUM_OF_IMAGES], y[NUM_OF_IMAGES];
 void        sub_view(int, int, int, int, int);
 void        ascii_view(int, int, int, int, int);
 void        display_histogram(int);

/* Retreive the image number from the drawing    */
/* widget.                                       */
 XtVaGetValues(w, XmNuserData, &n, NULL);

/* Select the event type.                        */
 switch(cbs->event->type)
 {
  case ButtonPress:
/* Select the pressed button.                    */
  switch(cbs->event->xbutton.button)
  {
   case Button3:
```

```
/* Button 3 pressed. Destroy the image and    */
/* reset the corners.                         */
  XtDestroyWidget(gettopshell(w));
  x[n] = y[n] = 0;
  break;

  case Button2:
/* Button 2 pressed. Update lower corner.     */
  wprint("Image %d: subwindow lower corner %d
          %d\n", n+1, cbs->event->xbutton.x,
          cbs->event->xbutton.y);
/* Check if Shift was pressed to select ascii */
/* view or sub image.                         */
  if (cbs->event->xbutton.state & ShiftMask)
   ascii_view(n, x[n], y[n],
   cbs->event->xbutton.x, cbs->event->xbutton.y);
  else
   sub_view(n, x[n], y[n],
   cbs->event->xbutton.x, cbs->event->xbutton.y);
  x[n] = y [n] = 0;
  break;

  case Button1:
/*  But 1 pressed. If shift update upper      */
/*  corner or display histogram.              */
  if (cbs->event->xbutton.state & ShiftMask)
   display_histogram(n);
  else
```

```
    {
     x[n] = cbs->event->xbutton.x;
     y[n] = cbs->event->xbutton.y;
     wprint("Image %d: subwindow upper corner
             %d %d\n", n+1, x[n], y[n]);
    }
    break;
   }
  }
}

/* Image has been destroyed free resources and   */
/* update variables.                             */
void destroy(Widget w, XtPointer client_data,
             XmDrawingAreaCallbackStruct *cbs)
{
 int       c=0, i, n;
 unsigned long pixels[256];

/* Retreive the image number from the drawing    */
/* widget.                                       */
 XtVaGetValues(w, XmNuserData, &n, NULL);

/* If a histogram is present, destroy it too.    */
 if (top_hist[n])
  XtDestroyWidget(top_hist[n]);

 XFreePixmap(theDisplay, pixmap[n]);
 XDestroyImage(ximage[n]);
```

```
 ximage[n] = NULL;

/*  Get a new current image unless image limit  */
/*  has been exceeded.                          */
 if (current == n && !client_data)
  for (current=0; (current < NUM_OF_IMAGES) &&
                  (!ximage[current]); current++);

/* Update the number a given pixel is used. If  */
/* it is allocated and it isn't longer in use   */
/* and it isn't part of the initial 83, prepare */
/* to deallocate it from the standard colormap. */
/* No need to worry if using the private        */
/* colormap since it doesn't set the variable   */
/* allc[] and therefore no pixel will be        */
/* deallocated.                                 */
 for (i=0; i < 256; i++)
 {
  use[i] -= hist[n][i];
  if (allc[i] && !use[i] && (i % 3))
  {
   pixels[c++] = cmap[i];
   allc[i] = 0;
  }
 }

/* Free selected colormap cells.                */
 XFreeColors(theDisplay, theColormap,
```

```
                pixels, c, 0);
 XtFree(hist[n]);
 image_count--;/* One image less in the display. */
}

int get_current(void)
{
 int    i;

/* Find an empty ximage structure.                    */
 for (i=0; (i < NUM_OF_IMAGES) &&
           (ximage[i]); i++);

/* If search is succesful. Allocated histogram  */
/* memory and return value.                           */
 if (i < NUM_OF_IMAGES)
 {
/* Success. Allocate memory for the histogram  */
/* and return the value.                              */
  hist[i] = (unsigned int *)XtCalloc(256,
                sizeof(unsigned int));
  return i;
 }
 else
 {
/* Failure. Maximum number of images on         */
/* display. Need to destroy one to make room    */
```

```
    /* for the new one.                                    */
     XtRemoveCallback(area[current],
                      XmNdestroyCallback, destroy, NULL);
     XtDestroyWidget(gettopshell(top[current]));
     destroy(area[current], &i, NULL);

    /* Allocate memory for the histogram and               */
    /* return the value.                                   */
     hist[current] = (unsigned int *)XtCalloc(256,
                     sizeof(unsigned int));
     return current;
    }
}
```

gauss.c

```
#include "ipt.h"
#include <stdlib.h>
#include <sys/time.h>

void gauss(Widget w, XtPointer client_data,
           XmAnyCallbackStruct *cbs)
{
 int       x, y, ix, iy;
 float     sum;
 gaussian *distrib;
 PIXEL     *image, *result, *img, *res;
 float     number_generator(float, float);

 if (!image_count)
 {
  wprint("No image present.\n");
  return;
 }

 XDefineCursor(theDisplay, XtWindow(top_level),
               watch);
 XFlush(theDisplay);

/* Retreive the mean and standard deviation.    */
 distrib = (gaussian *)client_data;

 ix      = ximage[current]->width;
 iy      = ximage[current]->height;
```

```
 image   = img = (PIXEL *)ximage[current]->data;

 result  = res = (PIXEL *)XtCalloc(ix*iy,
                    sizeof(PIXEL));

 current = get_current();

 srand(time(NULL));   /* Initialize random seed. */

/*  Add gaussian noise to the pixels. Make sure */
/* that no pixel is higher.                     */
 for (y=0; y < iy; y++)
  for (x=0; x < ix; x++)
  {
   sum = number_generator(distrib->std,
                          distrib->mean);
   sum += *(img++);
   if (sum > 255)
    sum = 255;
   if (sum < 0)
    sum = 0;
 *res = (PIXEL)sum;
   use[*res]++;
   hist[current][*(res++)]++;
  }

 ximage[current] = XCreateImage(theDisplay,
                   theVisual, 8,
                   ZPixmap, 0, (char *)result,
```

```
                    ix, iy, 8, 0);
 ximage[current]->byte_order       = MSBFirst;
 ximage[current]->bitmap_bit_order = MSBFirst;

 display_image(current);
 XUndefineCursor(theDisplay, XtWindow(top_level));
}

float number_generator(float std, float mean)
{
 int   i;
 float val = 32767.0, a=0;

 for (i=0; i<12; i++)
  a += ((077777 & rand()) / val);
 a = (a - 6.0) * std + mean;
 return a;
}
```

REFERENCES

1. Barkakati, Nabajyoti. 1991. *X Window System Programming.* Carmel, Indiana: SAMS.

2. Gonzales, Rafael C., and Paul Wintz. 1977. *Digital Image Processing.* Boston: Addison-Wesley.

3. Heller, Dan. 1991. *Motif Programming Manual.* Sebastopol, California: O'Reilly & Associates, Inc.

4. Pratt, William K. 1991. *Digital Image Processing.* New York: Wiley.

5. Rosenfeld, Azriel, Avanish C. Kak. 1982. *Digital Picture Processing.* San Diego: Academic Press.

6. Shih, Frank Y., and Christopher C. Pu. 1993. "Threshold Decomposition of Soft Morphological Filters." *Proc. of IEEE Conference on Computer Vision and Pattern Recognition.* 672-673.