CODEN: LUTFD2/(TFRT-5416)/1–32/(1989)

Window Based Handling of Image Hardware

Toni Ericsson

Department of Automatic Control
Lund Institute of Technology
October 1989

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Master Thesis

Date of issue
October 1989

Document Number
CODEN: LUTFD2/(TFRT-5416)/1-32/(1989)

Author(s)
Toni Ericsson

Supervisor
Lars Nielsen

Sponsoring organisation
STU

Title and subtitle
Window based handling of image hardware

Abstract

Window based interactive software systems can be used to create very user friendly interfaces e.g to image processing libraries, and hence to image processing hardware. A system has been implemented, where standard "off the shelf" products that are commercially available from different companies have been used. The use of readily available components greatly simplifies the implementation but even more the documentation. This is especially true on the software side.

Key words
Image processing, interaction, hardware interface.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
32

Recipient's notes

Security classification

1. Introduction

The purpose of this thesis has been to look into different possibilities of designing a window based environment for image processing software and hardware, in this case on a personal computer. A current trend in image technology is to provide plug-in boards for image handling together with software libraries for image processing.

Another line of development is window based interactive software systems (Newman and Sproull, 1978). Such software can be used to create very user friendly interfaces e.g to image processing libraries, and hence to image processing hardware.

A system has been implemented, where standard "off the shelf" products that are commercially available from different companies have been used. The focus has mainly been on general concepts for interactive image processing, and the system ideas can be viewed as an extension of earlier systems in the same flavor (Nielsen, 1985,1988).

This report starts with an overall description of the systems hardware and software components. Then follows a description of the resulting user program. Under the chapter Implementation an explanation is given how the program was implemented using the different tools, which is then followed by a discussion over the choice of user interface and the internal structure. Finally conclusions are drawn from the above mentioned experiences.

2. System description

The general idea is to implement an interactive system that provide for the user to manipulate existing image processing hardware based on a regular computer with a monitor displaying some sort of windowing system and with the aid of a pointing device.The actual tools for this work have been the Matrox PIP-1024B image processing board and the accompanying software library PIP-EZ as the image processing tools, and the Microsoft Windows operating environment as tool for the window environment. Thus the principal task has been to make a window based environment for PIP-EZ, since PIP-EZ itself is more of a low level set of routines designed specifically for the PIP-1024B board. This is a general problem since the hardware for image processing that is available today greatly differs in capacity and
speed. Many considerations have to be made concerning how to represent the capabilities of the hardware in a way that the user can easily understand without having to have a detailed knowledge about the actual system.

A IBM-PC/AT computer with a Microsoft mouse as pointing device was chosen as the host for this system since it is a single user computer and supported by a wide range of development tools. It is also the probably most inexpensive way of building an image processing system because of the competition on the market.

2.1 The Matrox PIP-1024B Board

The PIP is a plug in video frame grabber-digitizer board, available for different types of computers. It digitizes a video signal coming from e.g a video camera or video recorder in real-time at a resolution of 256 by 256 or 512 by 512 pixels, and with 8 bits per pixel. The board itself can be viewed as consisting of 3 sections, namely the input section, the grabbing section and the output section.

The input section consists of inputs for 3 video channels and the A/D converter. One of the input video signals is selected by the software and the PIP will lock on to the signals sync to drive the video functions of the board. Alternately the board can generate its own sync signal, which can be of different types. After the signal has passed through the input selection the sync signal is separated and passed on to the output section. The signal is then digitized by the A/D converter to a value in the range 0 to 255, where 0 represents black and 255 represents white. How the signal isdigitized is affected by 2 parameters given to the A/D converter, namely gain and offset. The offset parameter adds a DC voltage to the signal thus making the input darker or brighter. This has the same effect as the brightness potentiometer on a regular TV set. The gain parameter sets the range to which the A/D converter
digitizes the signal. A small range means that the A/D converter will be more sensitive to changes in the signal, whereas a big range might result in that the signal will not be digitized to the full resolution of the A/D converter. This parameter has the same effect as the contrast potentiometer on a regular TV set.

The grabbing section consists largely of the input lookup table and the video memory. After the signal has been digitized at some point an 8 bit value is ready to be stored representing the brightness of some small area of the picture. Before this value is stored it is passed through the input lookup table. This is table consists of
256 values with indices 0 to 255. These values are directly programmable by the software. The value from the A/D converter then serves as an index to this table and

the value at that index is then stored in the video memory. Thus it is possible to alter any digitized value to any other value thereby altering the picture for example its negative (as a photonegative).

The video memory is accessible by the PIPs CRTC as well as from the host computer via its bus. When the signal has been digitized the CRTC will generate the appropriate x- and y-coordinates in video memory where the value will be stored. The video memory of the PIP-1024B board has the capability to store 4 512 by 512
pictures and thus has a total capacity of 1024 by 1024 pixels. A picture can be grabbed to any arbitrary location in memory, and if it goes outside of the memory boundaries it will wrap around to the beginning. Thus it is possible to grab a picture which starts at x coordinate 768 and ends at x coordinate 255.

When a pixel value is written to the video memory, be that from framegrabbing or by a write instruction from the computer, it is possible to protect some or all bitplanes from being modified. This means that for example only the highest bit of the 8 bit pixel value is modified if the 7 lowest bits are write protected. Which bitplanes that are protected are specified by the writeprotect mask which is programmable by the software.

The output section consists of the keyer, the 3 output lookup tables (henceforth named LUTS) and the 3 D/A converters. When the output section is displaying either what is in the frame buffer or what is coming from one of the input channelsthis information is first passed through the keyer. The effects of the keyer can however only be seen when the frame buffer is being displayed. If the keyer is then turned on the PIP will at the lowest bit of each pixel in the frame buffer. Should that bit be set the pixel value in the frame buffer is displayed otherwise the pixel value from the input LUT is displayed. When the input is displayed it is actually the digitized pixel value from the input LUT that is displayed. After having passed through the keyer, the pixel value is routed through each of the 3 output LUTs. These LUTs are labeled red, green and blue, and consequently control their respective beam in the CRT. Since each pixel value serves as the same index to each LUT it is possible to remap the grayscale input to arbitrary pseudocolors. Each of output LUTs yield an 8 bit value thus making possible to choose 256 colors
from a palette of over 16 million.

The output of the output LUTs then goes directly to the D/A converters which yield the 3 analog signals for the monitor. A separated sync signal is also available from the output.

The PIP board is capable of grabbing and displaying video signals at a resolution of 256 by 256 pixels as well as at 512 by 512 pixels. Switching between these 2 modes can be done by the software. It is also possible for the PIP to operate on an American video signal (NTSC) as well as on European video (PAL). When operating with American video the resolution is somewhat less than previously indicated. This is however of little significance for the overall description.

2.2 The PIP-EZ software library

The PIP-EZ software is a library package that facilitates the control of the PIP board. This software consists of routines that provide PIP hardware control of functions such as frame buffer input/output, LUT manipulations, graphics and imaging functions. Working with PIP-EZ releaves the programmer of having to program the PIPs various I/O registers at assembler level. It further provides a basic model of the PIP board as an image processing system.

The languages supported by PIP-EZ are C, Fortran and Pascal. However it will only work with some of the compilers today available on the market. These are generally the compilers from Microsoft Corp. that are of later date. It is also possible to work with compilers for other languages that create .OBJ files as long as the calling conventions used by the Microsoft compilers are used.

Also provided with the PIP-EZ is an interpreter for executing the functions interactively. This interpreter has a fully command line driven interaction with the user, much like most shells for operating systems.

2.3 The IBM PC/AT

The IBM PC/AT is perhaps the most widely used personal computer today. Therefore the description of its features will only cover the most elementary parts and those things that are of relevance for developing windows based applications on it.

The microprocessor used is the Intel 80286, a 16 bit processor with a 24 bit address bus. Memory on the motherboard is limited to 640 K, but extensions are possible through paging. The processor runs at a speed of 6 MHz with 0 wait states. Other chips on the mother board supply functions such as timer functions, sound generation, interrupts and I/O. It is also possible to install a mathematical coprocessor (Intel 80287). There exists an external databus to which 7 open slots are connected. Some of these slots provide 16 bit data transfers some only 8 bit. The PIP board connects to one of the 8 bit slots.

The PC/AT used here has a 1.2 MByte floppy drive, a 20 MByte harddisk 512 K of RAM, 512 Kbyte as a ramdisk, a standard keyboard, an enhanced graphics adapter with an enhanced color monitor, and a Microsoft mouse. The enhanced graphics card is capable of displaying 16 colors at a resolution of 640 by 350 pixels, where the colors are selectable to some degree. One of the drawbacks of this card is that it does not supply any hardware overlaying of graphic objects as for example cursors or icons. This means that the background has to be saved where ever these objects are displayed, resulting in a somewhat slower performance.

Another main concern when programming the PC/AT is the segmented architecture of the processor. Addresses in these processor are not represented by a single 32 bit register value, but by 2 16 bit register values. This means that memory is divided into segments of which each can be at the most 64 Kbyte big. The two registers representing a physical address overlap by 8 bits thus yielding a 24 bit address. It is possible to execute instructions and reference data where only the lower 16 bits of the address is updated, as well as executing instructions etc. where both 16 bit registers are updated. The instructions where only the lower 16 bits are updated are generally faster, but have a more limited address space. The same considerations apply when using pointers to data. This poses special problems which will be discussed in conjunction with implementing the system in MSWindows.

2.4 The operating system

MS-DOS, alternatively PC-DOS, is the operating system for the IBM PC family of computers. Basically MS-DOS is a single user and one machine operating system since it does not provide any true multitasking or protection for shared resources. It arranges all files in a tree-like hierarchical structure of directories and subdirectories. It further more provides the user with functions for managing files, controlling output to the

screen and receiving input. A command line shell provides the user interaction and all functions are accessible from programs through software interrupts. MS-DOS utilizes the ROM BIOS in the computer for some of its functions.

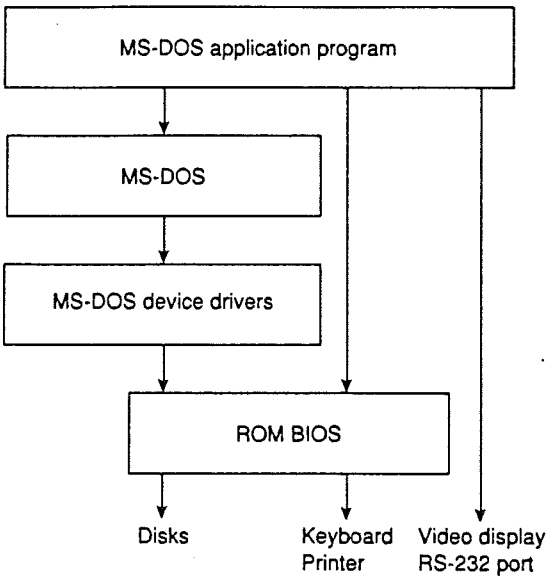


Figure 1. The structure of MS-DOS.

Because of its simplicity and due to the state of art of personal computers at the time when it was designed, MS-DOS has some serious shortcomings. One is that it can not directly address more than 640 Kbyte of RAM. Another is that the harddisks (or rather the harddisk partitions) used by MS-DOS can not be greater than 32 MByte. Both these limitations are of importance since images contain large amounts of data.

2.5 Microsoft Windows

Microsoft Windows, or just Windows, resides on top of MS-DOS and provides the user with three main capabilities : a graphics-oriented user interface, a multitasking capability, and hardware independence. When running Windows the user has a uniform environment for all applications he or she is running since the programs all have a very similar layout, practically regardless of what type of program it is. This environment is very similar to the one that became so popular with the introduction of the Apple MacIntosh. Very often the user has a pointing device, a mouse, with which he can supply input to the application program. In Windows this is typically done by selecting items in pulldown menus, pressing buttons in dialogboxes and drawing figures on the screen.

Of the three capabilities offered by Windows, the graphics-oriented user interface is the most striking and certainly the most important for the average user. The Windows user interface uses pictures to symbolize operating system commands, actions, and devices. Menus used with pointing devices and single keystrokes have replaced lengthy text commands. Pictures and graphics also enhance text messages. In Windows, text itself is just graphical information in that it can be sized, changed in shape, and moved around the page.

A typical Windows application has a caption bar at the top normally displaying the name of the application. Beneath this caption bar it has a menubar which contains several pulldown-menus. The user can select items in these menus by pointing to them with the mouse and pressing the appropriate mousebutton, or by pressing special keys on the keyboard.

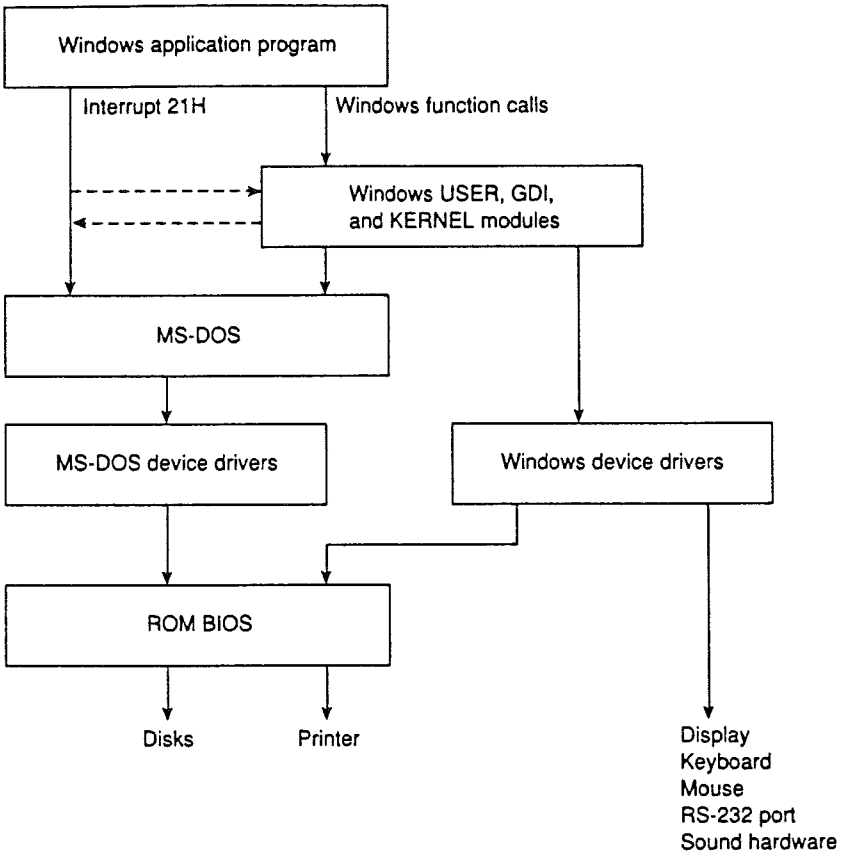


Figure 2. How Windows fits in with MS-DOS.

The rest of the display surface of an application normally consists of one or more child windows. These windows often supply the user with some information and receive user input independently of each other. The main display surface of the application is itself a child window since it shares the screen with other applications. The display surface of all applications running at the same time can be arbitrarily changed. If an application has a display surface that is too small for it in order to display all information, most applications will supply the main window with scrollbars such that the user can scroll and pan until the desired information is visible.

Some menuitems in the pulldownmenus trigger themselves new childwindows on the screen which often lay on top of windows under. These windows are often called dialogboxes, since they provide the user with a dialog for entering more specific data. A paint program may for example have a menuitem with which the user can choose a penstyle. Then a dialogbox might appear that provides the user with different options and receives the users selection.

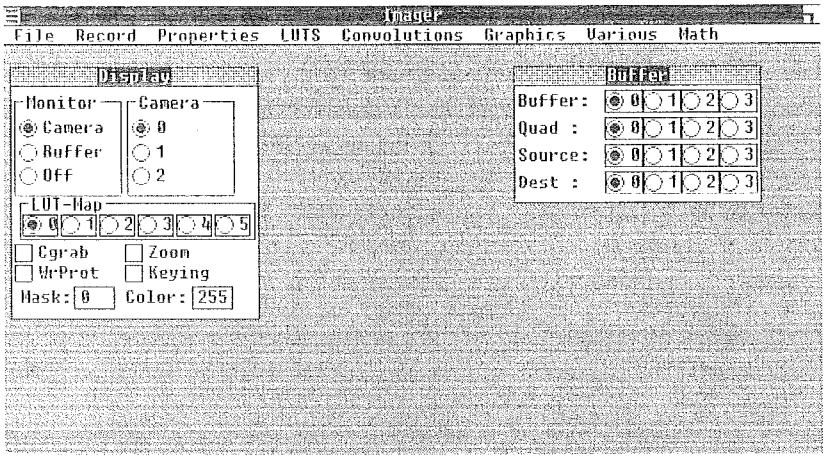


Figure 3. The starting display of Imager.

Those applications that have been loaded but yet not displayed are represented as small icons at the bottom of the screen. The user may activate an iconized application simply by clicking on the icon with the mouse. All applications that are written for Windows will run on any computer system that has Windows installed. This means that a Windows application can run on any display adaptor for which there is a Windows driver, regardless of this adaptors resolution and color capabilities. This is one example of Windows device independence.

2.6 Considerations

The PIP board is only one of many image processing boards that exist today. Similarly other window systems exist. Despite these differences there is a common platform for both image processing hardware and window systems. The question then arises how one can describe these system such that it is possible make a user interface for image processing that can be ported to different hardware and window systems without too much effort. It is also of interest to make an abstract description of image processing properties and capabilities, since this description will serve as a base when one decides what should be presented in the user interface.

The scope of this thesis has been to implement such an interface for existing hardware, thus penetrating the problems involved in building such systems. This involves analyzing the hardware and the window system to find an acceptable design for such a system with regards to the available tools. To build a system that can handle any hardware and that implements all image processing procedures used today is beyond the scope of this thesis. However an the experiences from this work can serve as guidelines for how this work could be continued.

3. The resulting user program

The resulting program, named Imager, for the PIP 1024B board implements the most fundamental routines and other routines to be run under Windows.

Imager has, as most other Windows applications, a caption bar and a menubar. The main window of Imager is always empty and consists only of a gray background. This background is then overlapped by numerous child windows, where each has a very specific task. When Imager starts two child windows are displayed at once. These two windows contain status of the board, and other parameters which are essential to many of the other functions provided in Imager. Such basic information is for example from where the input is currently being taken, what is displayed on the monitor, which LUT map is being used etc. etc.

Since some of the functions in Imager are invoked by menuitems and this quite frequently, and, since these functions also require parameters, this approach releaves the user of constantly having to supply these parameters by hand although they remainconstant very often.

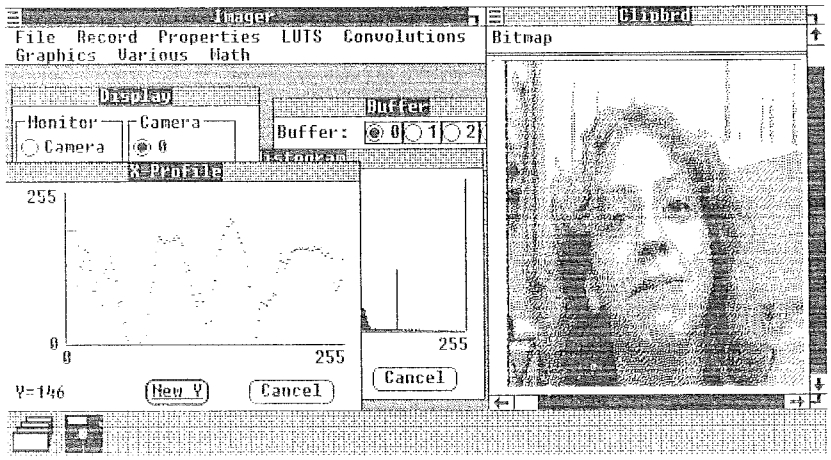


Figure 4. Imager passing data to other applications.

Many of the different menuitems trigger new child windows, which then run concurrently with the other window already present. Other menuitems trigger dialogboxes that set some internal parameter, or prompt for example a filename before saving data on disk. Imager also provides functions for printing the images on paper and passing pictures to other Windows applications.For a more detailed description on how Imager works and what capabilities it has, refer to appendix A, which is the manual for Imager.

4. Implementation

The Imager application has been written in the C programming language, and with a variety of other tools. To facilitate for modifications of Imager, the source code has been divided into several files, where each file contains functions that are closely related and that perform some defined task, such as file I/O, printing, or handling the input for some child window.

Before continuing to the layout of Imager, a description of the tools used to build Imager is necessary since they somewhat set limitations to what can be done and what not. The PIP board, PIP-EZ and Windows have already been described, and will therefore not be commented further.

4.1 The C compiler

The choice of C as the programming language for this project has two main reasons. First of all, the documentation for programming under Windows gives practically all examples in C code. Also the functions that Windows supplies for various tasks are all documented in C as far as parameter and return types are concerned as well as the syntax. Secondly those compilers used for compiling code for Windows must be able to generate special instructions for certain functions, which is selectable in the Microsoft C compiler version 4.00. Even though other compilers are available it seemed a good choice since Microsoft has designed Windows. Also this compiler is one of those supported by PIP-EZ.

The compiler is able to handle standard C, as defined by Kernighan and Ritchie, as well as a few extra facilities, most of them machine dependent. An extensive runtime library is included with the compiler as well as a linker and a make utility. When making Windows applications it is however not possible to use this standard linker for reasons that shall later be described.

4.2 The Windows Development Kit

In order to produce applications for Windows special tools are needed. Microsoft provides a special kit for this. It contains, among other things an 80 kbyte large header file named "windows.h", which contains the definitions for datatypes, file structures and many other things used by Windows. This file should be included in every source code file when making references to these structures.

The kit contains the library files for the Windows specific functions. These files are not ordinary library files that contain compiled code, but references to the functions that are already present in Windows. This means that not all function calls are resolved at link time. Instead these function calls have to be resolved at runtime, that is, when the application is loaded. This is the reason why the ordinary linker won't work for Windows applications. To link a Windows application one has to use a new linker called "Link4", which is supplied with the development kit. This linker sets up a table in the executable file that lists those functions for which the addresses can not be resolved at link time. When Windows then loads the program it inserts the addresses of these functions.

Further, the kit supplies some useful utilities for building Windows applications. Such a utility is the icon editor which lets the programmer design the icons for the application. Another is the font editor with which one can build characters with different typesettings.

The maybe most useful utility in the development kit is the dialogbox editor. With this editor it is possible to design dialogboxes and child windows on the screen instead of programming these in terms of x- and y-coordinates. Designing the windows this way also releaves the programmer of much boring code when developing the program.

Child windows, fonts, and icons etc., are all viewed by Windows as resources that the program can use. They all go into what is called a resource script file that is compiled with a special resource compiler, also supplied with the kit. This leads us to the topic of how a Windows application is put together.

4.3 Building a windows application

The steps involved in building a Windows application are somewhat different from building a normal application under MS-DOS. The C compiler compiles the source code into the object files. Resources, such as icons, cursors, fonts, and dialogboxes, are listed in the resource script file. The resource compiler uses this file to create the resource file.

The Windows linker, Link4, then combines one or more object files from the compiler into an executable file according to the module definitions file. This file is an ASCII text file that specifies the size of the program's heap and stack, the names of the functions that are exported (so that Windows can call them), whether the program's code is shareable (so that several instances of the program running at the same time use the same code), and whether the Windows can move the code and data around to make room for other programs. The executable file produced by Link4 has everything but the resources in it.

The last step is to use the resource compiler to add the resource file to the executable file from Link4. The result is a Windows format executable file. Windows format executable files are intended to be started from within Windows. If one starts a Windows program from MS-DOS, it displays the message "This program requires Microsoft Windows".

It is also possible to automate the process of compilation by building a make file for the program-maintainer utility, Make, that describes all the necessary actions.

4.4 Implementing Imager

The implementation of Imager has gone step by step, building code for various tasks one at the time and then, at the end, binding them together. The first concern was to divide the job into clearly defined subtasks. These can be divided into three major categories. One concern is the user interface in Windows, that is how this should look. Another concern is the functions making a model of the PIP board and how to supply input to the PIP monitor. The final concern is handling file I/O and which files should be created and maintained by Imager.

Routines were divided into three areas for the user interface. A choice was made to implement a child window for every task that was reasonably independent, and that

needed to provide the user with some sort of immediate feedback. Thus a separate source code file was made for the functions handling every such window. The benefits of this is that every window is separated from the others and if one window is to be modified only one file will have to be recompiled. Those actions that didn't need any immediate feedback were simply put as menuitems. A separate file was made to handle the menu selections. Two files were created for handling initialization of Imager. Imager can start up in a different mode by making changes to only these two files, and the code of these files is only needed when Imager starts. Having this code in separate files means that it can later be discarded from memory.

The files that contains the routines for performing operations on the PIP board were each specialized to deal with specific functions. They contain such functions as coordinate transformations, drawing graphics and providing mouse input with cursors. This includes a simple abstract model of the PIP as an image processing system. If this model is to be extended and new facilities added, only these files have to be altered. A choice had to be made on what data Imager should store on disk files for use in later sessions. These data are fully handled by the files that contain the code for file I/O. The procedures for prompting the user for information regarding these files are put in the source code files. A reason for doing this is that file data structures and user feedback is closely related. If a structure is changed then only the dialogbox for giving the user the information needs to be changed.

5. Discussion

5.1 Choice of user interface

The main objective of the user interface is to make the handling of hardware as easy and selfexplanatory to the user as is possible. An important factor to consider is how input can be supplied by the user. On regular computer terminals the keyboard is almost always the sole source of input. Even though a user interface can be built quite sophisticated this way it often displays severe shortcomings. Simulating commonly used tools such as pens and buttons often turns out to be awkward to do by keyboard. Pointing devices such as a mouse or a lightpen provides this capability in a much more natural way.

Since pointing devices are suitable for handling image data, it is natural to look for a type of interface in which these can be used. Graphical window systems supply good support for e.g a mouse which could make them practical for image processing applications. A windowing system is good also for other reasons. Because it is graphically oriented it can make use of pictures for selections.A picture is said to say more than a thousand words, and by chosing an operation by means of a picture the user can avoid having to read long and boring texts. This could speed up the learning process. Also, menus and buttons, things that are often familiar to the user from other environments such as car panels, vending machines etc., can be incorporated with ease. Adding to this that image processing is graphical by itself, building an image processing system based on a window system seemed the natural choice.

5.2 View of the image processing hardware

To start with, it was established that the hardware to be used for handling images would consist of some kind of plug-in board. Such frame grabber boards mainly consist of raster memory and A/D converters. To be able to bind the user interface to different kind of hardware it was considered that an abstract model of thehardware, especially raster memory had to be made, otherwise the result would be a heavily hardware dependent program.

The available raster memory is often capable of storing more than one image. A definition has to be made e.g were an image can be stored without destroying other images. Otherwise lost data could easily occur. Hence the available image space was divided into rectangular working areas, here each area is represented by a number. Though this limits the freedom of the user it also makes mistakes less probable, and the meaning of a working area identifier can be altered on other hardware with a minimum of change to the program code.

Issues such as color representation and monitor display were also taken under consideration. Since a pixel value can represent different colors with the use of color lookup tables no color representation was made for an image. It is the responsibility of the user to use the lookup tablet himself in a way he finds appropriate. The remapping of colors is hence a property of the presentation section of the hardware and not the image itself.

It was considered if the images should be part of, and displayed in the windowing system. Since images contain lots of data, it was found that the repainting of windows

would suffer too much in terms of speed for this to be an acceptable solution. As new hardware emerges it could be possible to implement this feature, giving the user more of a desktop environment. This is however not the case today, and hence a two monitor solution was judged to be the best choice.

5.3 Internal structure

Much of concern regarding the internal structure of a window based image processing system is what the system should know about the windowing system and the hardware. Since the aim was to build an image processing system the system was given the knowledge that there are such functions such as edge operators etc. It was also given the knowledge that there are subwindows, menus etc, which are properties of the window system. Given the goal that the system should be easily portable to other image processing hardware, the system was more tightly bound to the chosen window system than to the hardware.

The image processing functions were put into special, clearly defined files, which could be modified if new hardware was selected. The extent of these files would depend on how much support is given by existing software libraries for the hardware, and could therefore benefit from such support.

5.4 Experiences of our system

The fact that this program was made on an IBM PC/AT with the MS-DOS operating system has put limitations on Imager. Image processing is memory expensive, and memory on the IBM compatible machines is not well organized for large requirements of memory. A few operations may hit the limit of accessible memory. To somewhat avoid this problem operations have to be limited. Another problem deals with Windows itself. Windows runs many applications at the same time within the limited memory available. Even though Windows tries to solve this problem by moving the programs around in memory and sometimes even swapping them out on disk this is still a problem. The functionality of Imager is thus dependent on what other programs are running at the same time. The Windows applications take all actions based on messages. When Windows moves things around lots of messages are generated. This means that there can be considerable time between an operation request and its execution. For a more detailed description on how a Windows application works, refer to appendix B. Some problems deal with poor documentation and possible bugs in Windows. A few functions don't function the way they should. This has been solved by using other functions than those intended for some special task. However this has been at the expense of memory.

6. Conclusions

General aspects of interactive image processing have been studied. This includes a graphics visual interface and modeling of existing hardware. The combination of window based software with image plug-in boards makes it possible to design a user friendly environment for image processing. Commands can be grouped graphically for the interaction, and there are several flexible ways to extend the system.

A successful implementation has been made based on commercial hardware and software. The amount of work has been reasonable, and the basis laid can be extended as new hardware and software systems emerge. The result has been satisfactory and has resulted in a program which has been sold commercially.

7. References

NIELSEN, L. (1985): "Simplifications in visual servoing," Ph.D. thesis CODEN: LUTFD2/TFRT-1027, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

NIELSEN, L. (1988): "Automated Guidance of Vehicles using Vision and Projective Invariant Marking ," *Automatica*, **24**, No.2, 135-148.

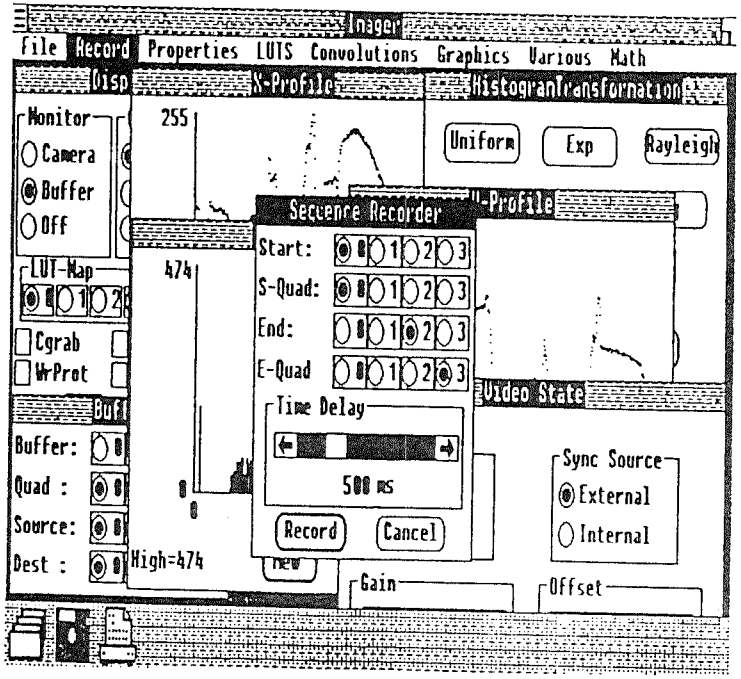
NEWMAN W.M, SPROULL R.F (1978): *Principles of Interactive Computer Graphics*, McGraw-Hill.

MATROX Electronic Systems Ltd. (1988): *PIP Hardware Manual*, MATROX Ltd, Dorval, Quebec.

MATROX Electronic Systems Ltd. (1988): *PIP-EZ User Manual*, MATROX Ltd, Dorval, Quebec.

MS-IMAGER

INTERACTIVE IMAGE-PROCESSING SOFTWARE FOR MATROX PIP-1024



MICRO SCANDIA AB
BOX 325
S-201 23 MALMÖ
THP:+46-40-358650
FAX:+46-40-922301

1. Introduction

Imager is a fully windows-driven interactive program for doing imageprocessing on the Matrox PIP 1024B imageprocessing -board installed in IBM-PC/PC-XT/PC-AT or true compatible. It reqiures MS-DOS 3.2, Microsoft Windows and a Microsoft Mouse. A mouse compatible with the Microsoft Mouse or that has a driver for MS-Windows should also do. The current version will run properly with MS-Windows versions 1.02 - 1.04.

It is recommended that your computer be equipped with at least 512K of memory.

The functions included in Imager follow the PIP-EZ library, which is delivered with the PIP board, to some extent. Some of the functions of PIP-EZ are unmodified, some have been restricted and new facilities have been added. To what extent modifications have been made will follow from the text describing the particular function.

Before reading this manual you should have read through the PIP-manual and be familiar with the concepts regarding the hardware and the PIP-EZ software library. You should also be familiar with MS-Windows.

2. Contents of the disk

The Imager-disk should contain the following files :

```
Imager    .exe  :  The Imager program .
Imager*   .ini  :  Initialization files for Imager
*         .*lu  :  LookUp table files described later
Test      .win  :  Stored test picture of part of
                   the screen
Test      .wid  :  Info about Test.win
```

3. Installing Imager

Before installing Imager you should have installed MS-Windows on your computer, since Imager won't run without it.

To install Imager follow these steps :

1) Copy Imager.exe and Imager.ini to a directory of your choice.

2) Copy the *.*lu files to a directory of your choice, preferably the same as in step 1.

3) Copy the Test.* files to a directory of your choice, preferably the same as in step 1.

4. Imager vs PIP-EZ

The main difference between Imager and the PIP-EZ regarding the configuration of the PIP-board is the way the framebuffer is configured. Even though the PIP-1024 has a frambuffer that is 1024*1024 pixels big, Imager divides this workspace into either 4 512*512 buffers or 16 256*256 buffers (henceforth named quadrants). The user thus can't scroll or pan over the whole 1024*1024 area. This limitation has been made so that the user always has clearly defined workspaces and quickly can change between them.

5. Basic Definitions

Before proceeding to the program itself a few definitions have to be made. These are definitions that will occur frequently in the following text.

Monitor

When the word monitor is used it is always meant the monitor where the output video of the PIP- board is displayed.

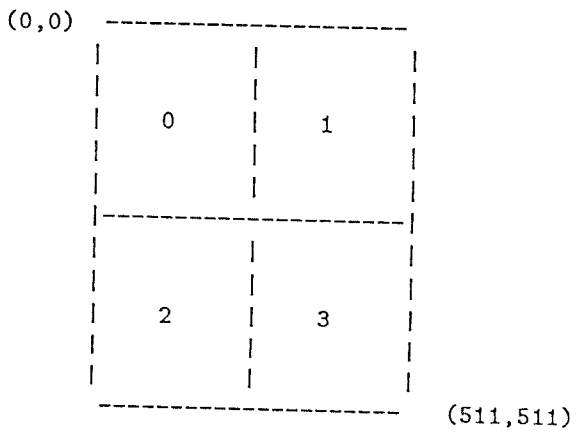
Buffer

A buffer is a 512*512 pixels big area of the frame- buffer. When the PIP-board is not in zoom-mode you can view one buffer at the time on your monitor. Four such buffers exist and are labeled 0-3. This is configuration is the same as the PIP-EZ mode 0.

Quadrant

A quadrant is 256*256 pixels big area of the frame- buffer. Sixteen quadrants exist when zoom-mode is selected. In zoom mode a quadrant will occupy the the whole screen of the monitor. The labeling of the different quadrants involves a quadrant index 0-3 as well as a buffer index 0-3 because each buffer is divided into 4 quadrants. The labeling of the quadrants will thus be 0:0, 0:1, .. ,1:0, 1:1 ... 3:3, etc.

The location of each quadrant within its buffer is as follows :



Coordinate system

The upper left corner of a buffer/quadrant will always have the coordinates (0,0) and the lower right corner will be (511,511) or (255,255) respectively.

Rectangle

A rectangle is a rectangular area of a buffer or quadrant usually used to define the extents or boundaries for some operation.

Window

A window is similar to a rectangle but is more of what is known as an AOI (Area of Interest), which sets the boundaries for imageprocessing operations. This is synonymous with the PIP-EZ definition of a window.

Set-Window and Set-Rectangle

The user will often be prompted to define a rectangle or window. When so, the following happens. The arrow-cursor disappears from the computer-monitor and the corners of a rectangle will appear on the monitor. By moving the mouse the user can move the rectangle on the monitor. The rectangle will be confined to the currently displayed buffer/quadrant. By pressing and/or releasing the left button of the mouse the user toggles between moving the whole rectangle or only the lower right corner thus resizing the rectangle. When resizing the rectangle the corners will be shown with double lines. The horizontal lines will be shown doubled if the user is allowed to resize the y-extents of the rectangle. The vertical lines will be shown doubled if the user is allowed to resize the x-extents of the rectangle. Pressing the right mousebutton will set the rectangle, carry out any requested operation and return the user to the computer-monitor.

6. Starting Imager

To start Imager you doubleclick with the left mousebutton on Imager.exe from the MSDOS-Executive.

When Imager has been loaded it searches the current directory for a

file named "Imager.ini". This file contains initialization data for Imager. The format of this file is :

```
baseadress(hex) video sync camera monitor zoom
```

The line with these values should be the first line of this textfile. Since Imager only scans this first line, you are free to include comments in the file on all lines after. The values the respective paramaters, and their meanings are as follows:

Baseadress

This is the address to which the board is strapped. If you have not changed the configuration of the straps from when the board was shipped then this value should be 26c . This value should always be given in hexadecimal notation.

Video

This parameter initiates the board to either american or european video. A value of 0 should be given for american video and any other value for european video.

Sync

A value of 0 gives internal synchronization all other values give external synchronization.

Camera

This parameter selects one of the three cameras (input channels). Values from 0 to 2 are expected.

Monitor

This parameter specifies what is to be shown on the monitor. Values from 0 to 2 are expected, where 0=Camera , 1=Buffer and 2=Off (see The "Dipslay" window below).

Zoom

Setting this parameter to 0 will make Imager start in 512*512 mode, and any other value will start Imager in 256*256 mode (zoom-mode).

For example a file with the parameters set as :

```
26c 0 1 0 0 1
```

will initiate Imager to

```
Baseaddress 26c (hex)
American Video
External Sync
Camera 0
Monitor will display Camera
Zoom mode (256*256)
```

If Imager doesn't find the Imager.ini file or if the file is incorrect, Imager will start up with a default state which is,

```
Baseaddress 26c (hex)
European Video
External Sync
Camera 0
Monitor will display Camera
Full mode (512*512)
```

At first a menubar and two childwindows are displayed in the main Imager-window. The two child windows named "Display" and "Buffer" will be visible during the whole session with Imager. The contents of these windows are important to many other operations, and therefore need explanation before proceeding to the other functions in Imager.

7. The "Display" window

This window consists of the following items:

Monitor

The alternatives in this box determine what is to be shown on the monitor.

```
Camera  : The input from one of the input video channels
          is displayed.
Buffer  : The framebuffer is displayed.
Off     : The monitor is turned off.
```


Camera

One of the three video input channels is selected.

LUT-Map

One of the six lookuptablemaps is selected (two are reserved for internal use).

Cgrab

Checking this box turns continous framegrabbing on.

Zoom

Checking this box sets the PIP-board in zoom (256*256) mode.

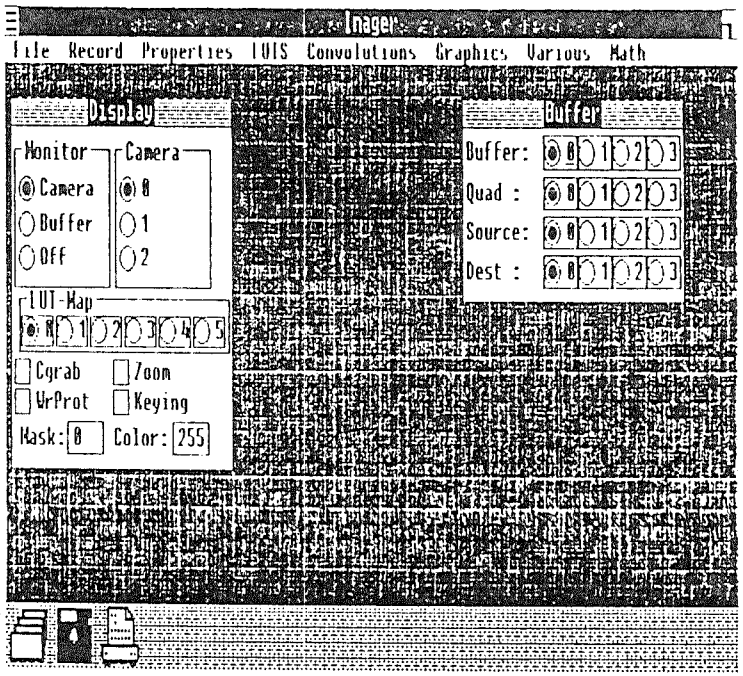


Figure 1. The starting display of Imager

WrProt

Checking this box protects certain bitplanes from being written to. Which bitplanes are set by "Mask" (see below).

Keying

Checking this box turns the videokeying on.

Mask

This is the value that specifies which bitplanes should be protected from being written in to. To make a mask effective "WrProt" must be turned off, the new value entered and then turning "WrProt" on again. If the new value is not within the limits 0-255 the mask will be reset to the previous mask.

Color

This is the drawingindex for the pen when drawing graphics into the framebuffer. It must be in the range 0 to 255.

8. The "Buffer" window

Buffer

Checking one of these radiobuttons will display that buffer on the monitor, provided that the "Buffer" button in the "Display"-window has been selected. The chosen buffer will then be the "current buffer".

Quad

When in zoom-mode these radiobuttons select the quadrant to be displayed, which will be the "current quadrant". This will of course also depend on the "current buffer".

Source

These radiobuttons specify which buffer should be the source for an operation of type dest=f(source), where dest is the destination buffer in which the result is stored. If in zoom-mode, the source will be the "current quadrant" in the source buffer and destination will be the "current quadrant" in the dest buffer. Hence it is only possible to do an operation from a quadrant in one buffer to the same quadrant in an other buffer when in zoom-mode.

Dest

These radiobuttons specify which buffer is to be the destination buffer.

9. Windows in Imager

A window is an AOI that Imager uses when performing certain operations. This is the same as the window in PIP-EZ. We will call this window the "current window". The window will be different depending on if you are in zoom-mode or not. Hence Imager holds different windows for each mode. Another window is the "split/zoom window". This window is only interesting when you are not in zoom-mode, and only for a few operations.

10. Features

The functions included in Imager are triggered from various pulldown menues where some items trigger and display new windows. The rest of this manual will examine these menues one by one, and if the menuitem triggers a new window the contents of that window will be described.

The menues are:

```
File
Record
Properties
LUTS
```

```
Convolutions
Graphics
Various
Math
```

File

The file menu consists of the following items:

```
SaveFrame...
LoadFrame...
SaveWindow...
LoadWindow...
```

SaveFrame...

A dialogbox is shown that prompts you for a name. The name should

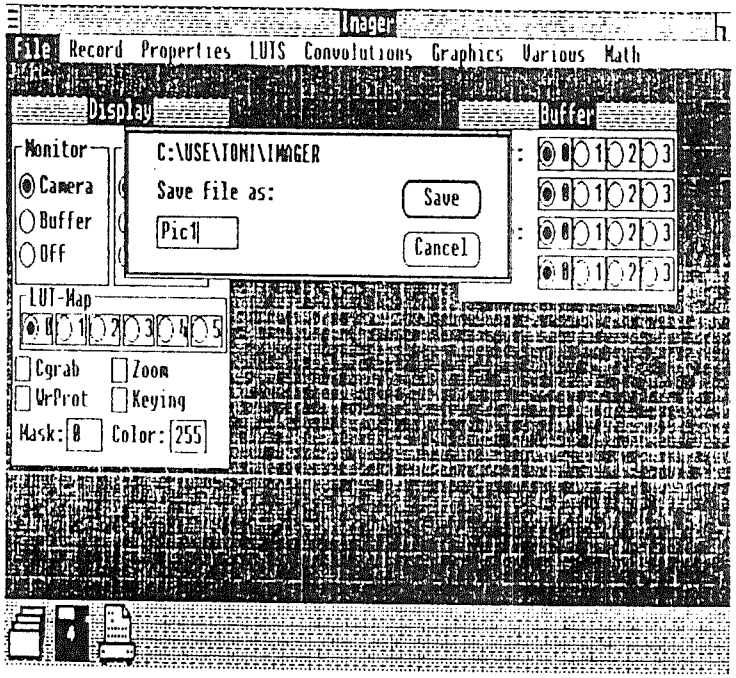


Figure 2. Dialogbox for saving a frame.

be a regular filename without extension, since Imager adds the extension ".FRM". Imager then stores the current buffer in this file. The file is put in the directory where you started Imager from.

LoadFrame...

A dialogbox with a listbox is displayed that lists the files that have the extension ".FRM", and subdirectories. To load a frame (or buffer) select the filename and press the Ok button, or doubleclick on the filename. You may also load frames from other subdirectories. To do so select a subdirectory by doubleclicking with the left mousebutton on it and the frame-files in that directory will be displayed.

Save Window...

This is the same as SaveFrame... except that only the part of the buffer/quad that is in the current window is saved. These files will have the extension ".WIN". A file with the same name but with the extension

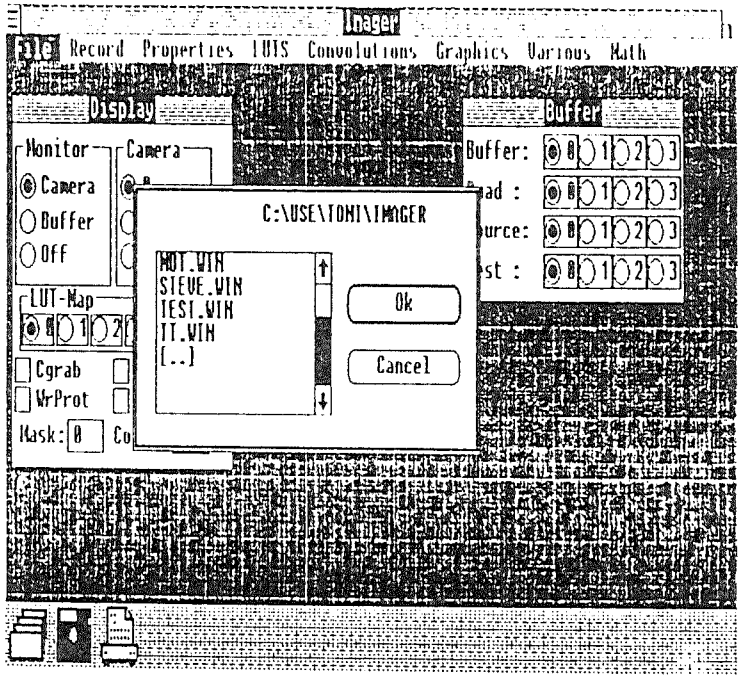


Figure 3. Dialogbox for loading a window.

".WID" is also produced. This file contains information about about the location of the window.

Note : This only works for buffer 0.

LoadWindow...

This is the same as LoadFrame... but for ".WIN" files. You must make sure that the ".WID" file is in the same directory as the ".WIN" file, otherwise Imager will probably not load the window correctly.

Note : This only works for buffer 0.

Record

This menu consists of the following items:

```
Snap              (^S)
Feedback Snap     (^F)
Zoom Snap         (^Z)
Sequence...
```

Snap

Selecting this menuitem takes a single snapshot into the current buffer or quadrant. This function may also be invoked by pressing "Control Z" on the keyboard.

Feedback Snap

Selecting this menuitem takes a feedback-snapshot into the current buffer/quad. This function may also be invoked by pressing "Control F" on the keyboard.

Zoom Snap

Selecting this menuitem takes a snapshot in zoom-mode into the current buffer/quad. This function may also be invoked by pressing "Control Z" on the keyboard. The snapshot is taken into the split/zoom window position in the current buffer.

SequenceRecorder...

This menuitem triggers the SequenceRecorder dialogbox. The SequenceRecorder takes a number of snapshots with a timedelay that you

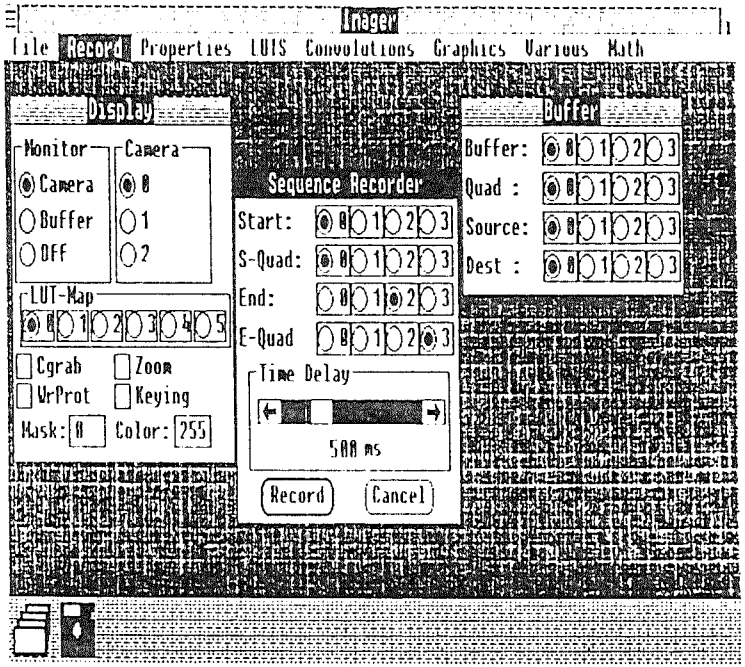


Figure 4. The SequenceRecorder dialogbox.

set with the Timedelay scrollbar. You set the startbuffer, startquadrant, endbuffer and endquadrant with the respective radiobuttons. The quadrant-radiobuttons are only effective if Imager is in zoom-mode. The buffer/quadrant that the snapshot is taken into will wrap around at 3 and continue at 0 if the start buffer/quadrant is greater than the endbuffer/quadrant. When in zoom-mode the quadrants are snapped first before continuing to the next buffer.

Properties

This menu consists of the following items:

```
X-Profile...
Y-Profile...
Histogram...
Plot Histogram
```

```
Transformation...
Set Full Window
Set Window
Show Window
Set Split/Zoom Window
Split
Zoom View
```

X-Profile...

A window appears that allow you to view the pixel values along a line y=const in the current buffer/quadrant. The values will be shown in an xy-diagram where the x-axis is the x-position of the pixel and the y-axis is the pixel- value. You select a new y coordinate by pressing the "New Y" button. A small cross appears on the monitor that you can move with the mouse. You select by pressing the right mousebutton, and the diagram

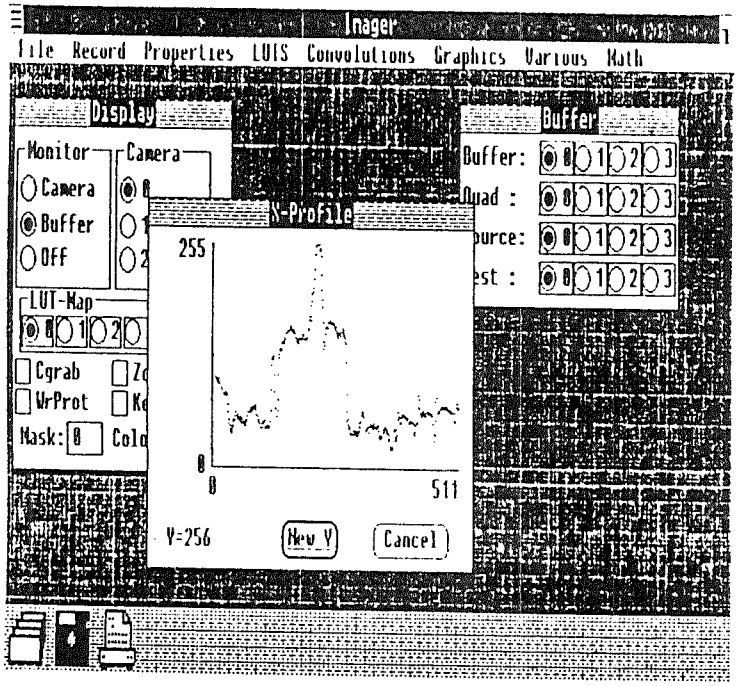


Figure 5. The X-Profile window.

is drawn. You can remove this window from the screen by pressing the "Cancel" button

Y-Profile...

A window appears that allow you to view the pixel values along a line x=const in the current buffer/quadrant. The values will be shown in an xy-diagram where the x-axis is the y-position of the pixel and the y-axis is the pixel- value. You select a new x coordinate by pressing the "New X" button. A small cross appears on the monitor that you can move with the mouse. You select by pressing the right mousebutton, and the diagram is drawn. You can remove this window from the screen by pressing the "Cancel" button.

Histogram...

This menuitem displays a window that lets you make a histogram of the current window in the current buffer/quadrant. To make a histogram

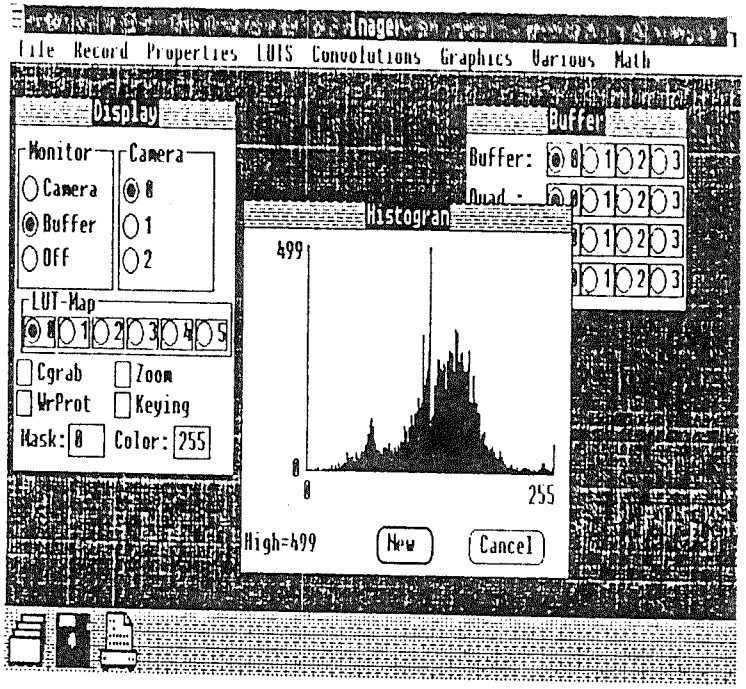


Figure 6. The Histogram window.

press the "New" button and the histogram will be drawn. The count of the most frequent pixel will be shown in "High=(count)". You may remove this window from the screen by pressing the "Cancel" button.

Plot Histogram...

When selecting this item you will be prompted to set a rectangle on the monitor, where you can only resize the y-extents. A histogram will then be made as in Histogram... but this time the histogram will be drawn within your selected rectangle on the monitor.

Transformation...

A window is displayed that contain buttons to perform various histogram transformations. These and the variables that have been set as input parameters are the same as for the PIP-EZ modhist function. The histogram is taken from the current window in the source-buffer and the resulting transformed picture is written into the current window in the dest-buffer.

Set Full Window

Selecting this item will set the window to its maximum in the current mode. This is what is visible on the monitor. Setting the window when in zoom-mode will not affect the window that is in non zoom-mode, and vice versa.

Set Window

You will be prompted to set a rectangle on the monitor. This set rectangle will be the new window. Setting the window when in zoom-mode will not affect the window that is in non zoom-mode, and vice versa.

Show Window

A rectangle will be drawn around the current window. If you do this operation once more without changing the window inbetween, the picture will be restored. If, however, you change the window inbetween, the pixels under the rectangle can not be restored.

Set Split/Zoom Window

Here you set the split/zoom window, a 256 by 256 window , that is used in a few operations. This window can not be resized.

Split

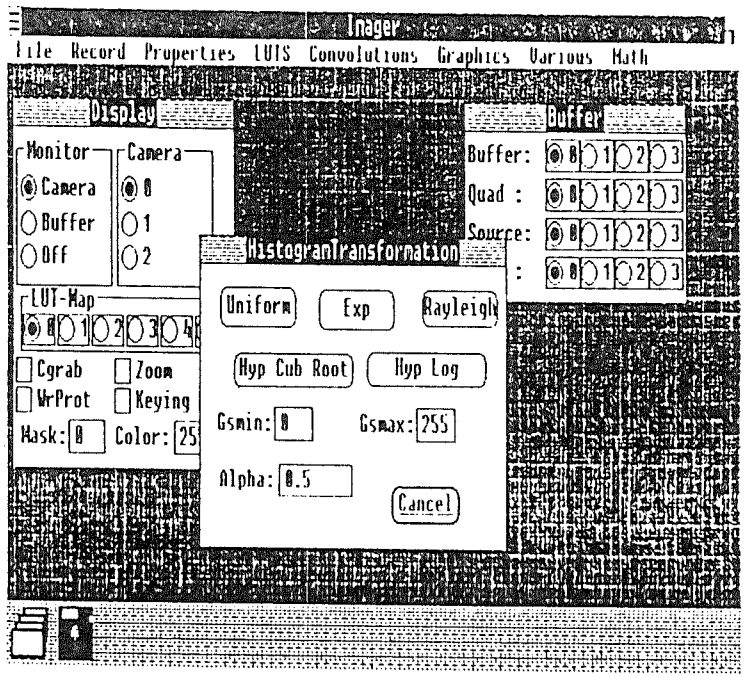


Figure 7. The HistogramTransformation window.

The area in within the split/zoom window in the source-buffer will be copied into all 4 quadrants in the dest-buffer.

Zoom View

The area in within the split/zoom window in the current buffer will be zoomed in to the whole monitor. You resume by pressing the right mousebutton.

LUTS

This menu consists of the following items:

```
Load LUT...
Define & Save LUT...
```

Load LUT...

The same dialogbox as in LoadFrame... and LoadWindow... will be displayed, and the files containing LookUp tables are listed. These files have the extensions ".ILU", ".OLU" and ".FLU" depending on if the file contains an input LUT, the 3 output LUTS or input and output LUTS respectively. These files are binary files of the following format:

```
".ILU"  : A 256 byte file with an input lookup table.
".OLU"  : The 256*3 bytes of the output LUTS stored in
          the order R-G-B.
".FLU"  : First the 256 bytes of the input LUT then
          the 256*3 bytes of the output LUTS in the same
          order as above.
```

You may create your own LUT files if you follow these rules and give them the right extension. When you have selected a LUT-file it will be loaded into the current LUT-Map that is set in the "Display" window.

Define & Save LUT...

A dialogbox will be displayed allowing you to make a LUT of your own choice. There are four scrollbars named I, R, G and B that let you set the values for the respective LUT and index. The index that you want to edit is set with the "LUT Index" scrollbar. If you check the "Auto" checkbox then every time you change the LUT index the LUTS at the index will automatically be given the values of the IRGB scroll- bars. This depends on which mode in the "Mode" groupbox (radiobuttons) you have chosen. If you have chosen "Input" only the input LUT is affected, "RGB" only the output LUTS are affected and "Full" then all LUTS are affected. By also checking the "Interval" checkbox all LUT indices from the old index to the new index are set to the IRGB values. You can't choose "Interval" unless "Auto" is chosen. If the "View" checkbox is checked then every time you select a new index the IRGB scrollbars are updated to show the present values at that index.

The "Monitor" radiobuttons let you choose between showing input videosignal or the framebuffer on the monitor. This is helpful when designing the input LUT.

The "Save" button saves the current edited LUT as a file of the formats described above. What format and what is saved will of course

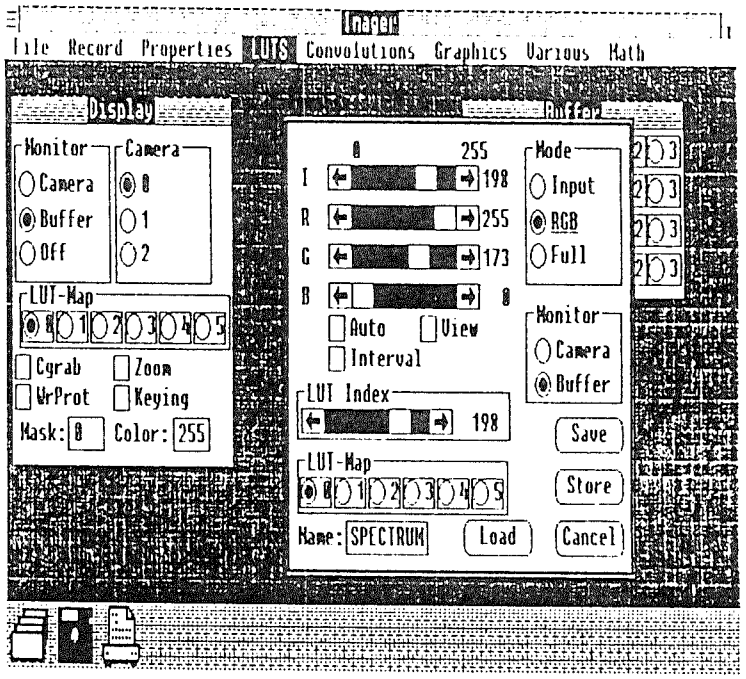


Figure 8. The Define & Save LUT dialogbox.

depend on what "Mode"-radiobutton that is selected. You must first enter a name in the name editbox at the lower right in the dialogbox.

The "Store" button stores your LUT in the LUT-Map that is selected with the "LUT-Map" radiobuttons.

The "Load" button triggers the "Load Lut..." dialogbox and lets you load a LUT-file for editing.

By pressing the "Cancel" button you close this dialogbox.

Convolutions

This menu consists of the following items:

```
Average
Median
Sharpening 1
```

```
Sharpening 2
Horisontal Edge
Vertical Edge
Prewitt
Sobel
Laplace 1
Laplace 2
Dilatation
Erosion
Copy
```

All these operations,except Copy are convolutions with a 3*3 kernel. The input is taken from the current window in the source-buffer and the result is written into the current window in the dest-buffer. They are also exactly the same as those in PIP-EZ.

The Copy item merely copies the window as is without altering the pixels.

Graphics

This menu consists of the following items:

```
Ellipse
Fill Ellipse
Rectangle
Fill Rectangle
Line
Grid
Small Text
Big Text
Red   (color mode)
Green (color mode)
Blue  (color mode)
White (B/W mode)
Black (B/W mode)
```

Ellipse

you want the text to be drawn and press the right mousebutton. The text will be drawn with the current color (index). The size of the characters will be 8 by 8 pixels.

Big Text

Same as Small Text but the size of the characters will be 16 by 16 pixels.

Red (color mode)
Green (color mode)
Blue (color mode)
White (B/W mode)
Black (B/W mode)

All these items affect the current LUT-Map in the following way :

The three highest indices of the input LUT are set to 252. The three highest indices of the output LUTS are set to are set to red, green and blue colors. The rest of the LUT indices are set to normal greyscale values.

The color (index) is the set to the color specified in the menuitem.

This is often useful when you want to draw graphics in color on a black and white picture.

Various

This menu consists of the following items:

```
VideoState...
Print...
ClipBoard
Draw Spectrum H
Draw Spectrum V
```

VideoState...

Window is displayed allowing you to change videoformat, syncronizing source, gain and offset. The "Autoset" button will set optimal gain and offset the same way as the PIP-EZ autoset routine.

By pressing the "Ok" button the window is destroyed and the new values are kept.

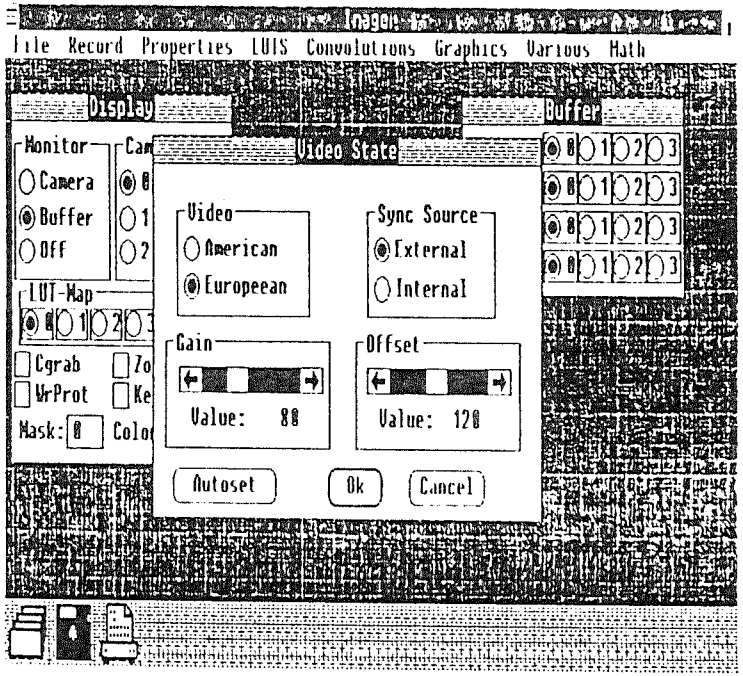


Figure 9. The VideoState window.

By pressing the "Cancel" button the window is destroyed and the old state is reinstated.

Print...

This allows you to print a halftoned image of the current window in the current buffer/quadrant. First a dialogbox will ask you for a name, and you must check the "Histogram" checkbox if you want a histogram of the image to be printed as well. Press the "Print" button to continue. Now a time-consuming operation starts. During this process a dialogbox will appear so that you can cancel the operation at any time. When this process is over the spooler will print the image (See example at the end of this manual).

It is essential that you have a printer installed that has a spooler driver and raster capabilities otherwise this process will fail. (A plotter won't do !)

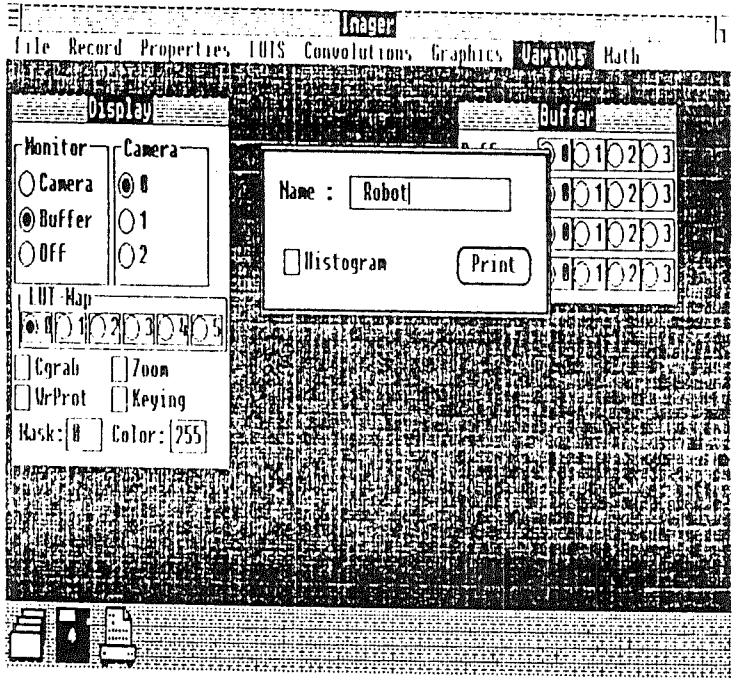


Figure 10. The Print dialogbox.

Clipboard

Selecting this item will make a halftoned binary image of the current window in the current buffer/quadrant and transfer it to the clipboard. The halftoned image can then be copied into other applications such as Paint or Write, or simply be viewed by the Clipboardviewer.

Draw Spectrum II

Selecting this item will prompt you to set a rectangle. A horizontal spectrum will then be drawn in this rectangle with color 0 at the left and color 255 at the right in the current buffer/quadrant.

Draw Spectrum V

Selecting this item will prompt you to set a rectangle. A vertical spectrum will then be drawn in this rectangle with color 0 at the bottom and color 255 at the top in the current buffer/quadrant.

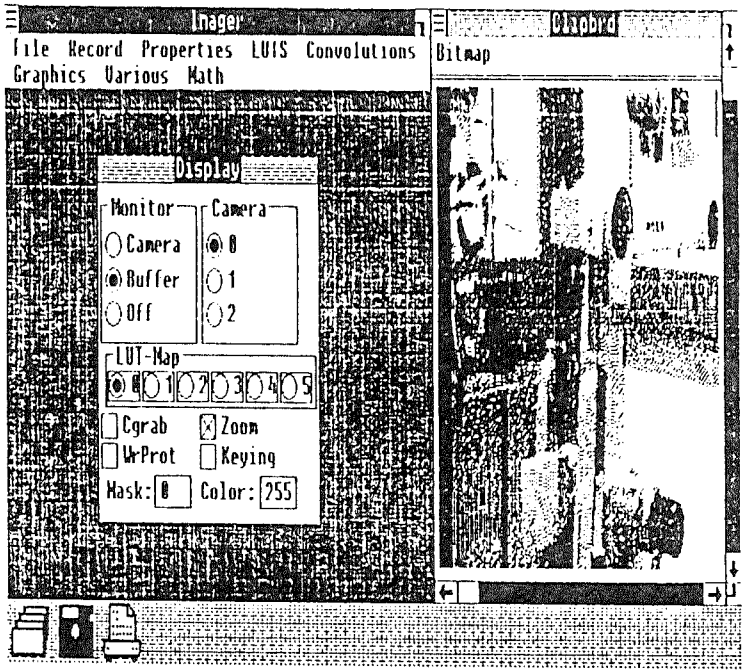


Figure 11. A bitmap transfered to the clipboard.

Math

This menu consists of the following items:

```
Add
Sub
And
Or
Xor
Not
```

All of these item perform a mathematical or logical function on the image contained in the current window in the source-buffer and sometimes destination buffer, placing the resultant image in the destination-buffer. All operations are done pixelwise, and are defined as follows :

```
Add  :  dest=(dest+source)/2
Sub  :  dest=abs(dest-source)
And  :  dest=(dest and source)      ( bitwise )
Or   :  dest=(dest or source)       ( bitwise )
Xor  :  dest=(dest xor source)      ( bitwise )
Not  :  dest=not(source)            ( bitwise inverted)
```

11. Concluding Remarks

Since all applications that run simultaneously under Windows fight for the same memory not all operations in Imager will succeed at all times. Sometimes Imager might not even start. To make sure that Imager can run as well as possible, run as few other applications as possible concurrently with Imager.

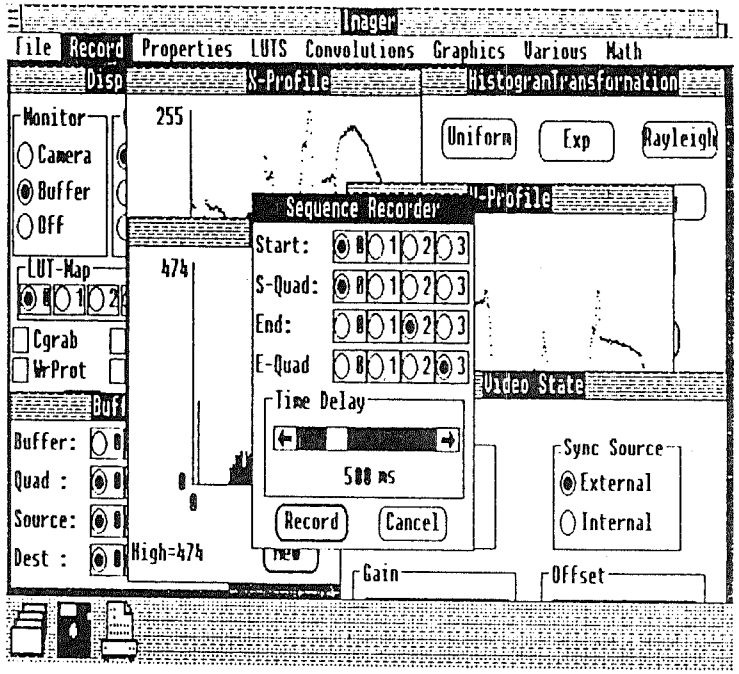


Figure 12. Imager with most of its windows.

Note : MS-DOS, MS-Windows, MS-Mouse and Microsoft are trademarks of Microsoft Corporation.

IBM, IBM-PC,-PC/XT and IBM PC/AT are trademarks of International Business Machines Inc.

PIP-1024, PIP-EZ and Matrox are trademarks of Matrox Electronic Systems Ltd.

Appendix B

Windows applications

A windows application sees Windows as a layer between the operating system and the drivers that handle the device dependence.

The application is a task that sits in loop, polling the message system for messages and responding to them. Messages can be generated by the clock, mouse, keyboard, or by other tasks. A task may or may not have a window, but one that does must also have a window procedure that interprets messages sent to the window. Messages can be sent synchronously or asynchronously.

Every task must have task-entry point, called `Winmain`. This procedure initiates the program and then starts polling messages from its message queue. Depending on what happens in the Windows system, Windows will send different messages to the application. If, for example, a part of the applications window has been overwritten by some other window, Windows will send a `WM_PAINT` message to the application, telling it to repaint its window.

If a an application should choose not to process a certain message it must pass this to the `DefWindowProc()`. This procedure is part of Windows and Windows will then take some default action for the message.

An applications main window procedure can choose process all messages for its child windows, or the application can supply these windows with their own window procedures, in which case Windows will send all messages concerning these windows directly to their respective window procedures.

This later step has been taken in Imager in order to achieve greater modularity. Hence the Imager window procedure processes only a few messages itself.

```
LONG FAR PASCAL ImagerWndProc( hWnd, message, wParam,lParam)
 HWND hWnd;
 unsigned message;
 WORD wParam;
 LONG lParam;
{

 switch (message)
 {
  case WM_SYSCOMMAND:
   switch (wParam)
   {
    case IDSABOUT:
     DialogBox( hInst, MAKEINTRESOURCE(ABOUTBOX), hWnd,
                lpprocAbout );
    break;

    default:
     return DefWindowProc(hWnd, message, wParam, lParam );
```

```
    break;
   }
  break;

  case WM_COMMAND:
   MenuDispatch( hWnd, wParam, lParam);
  break;

  case WM_DESTROY:
   PostQuitMessage( 0 );
  break;

  default:
   return DefWindowProc(hWnd, message, wParam, lParam );
  break;
 }
 return(0L);
}
```

In windows everything, code as well as data, is an object in global memory. Windows moves these objects around when it needs more space. Therefore one has to be extremely careful with pointers to allocated memory objects. A pointers that once pointed to a valid object may now point at some garbage if Windows has moved this object. This is why Windows supplies handles for reference to every object. The handle can then be used to retrieve the data in the object to a fixed memory location.