Image WorkFrame

ver 1.2

Craig Muller
1991-1994

cmuller@ccu.umanitoba.ca
Computer Vision Laboratory
Dept. of Industrial Engineering
University of Manitoba
Winnipeg, Manitoba.
CANADA. R3T 2N2

Introduction to Image WorkFrame

Image Workframe (IWF) is specifically designed to provide an easy interface to the Windows programming environment. It has grown out of the Computer Vision projects which have been under way for a number of years.

Most of the projects were aimed at running on the PC platform and did not require any real-time or multitasking. They did however require a lot of memory space and a Graphical User Interface if they were to be interactive. Because of its low cost and huge support base Windows was seen as a ideal environment for this software. Windows was chosen as a preferred environment for developing this toolkit because it is, by design, graphically based with levels of abstraction for all the hardware devices. This makes it more difficult to build a small program but when the software becomes more complex, such as in image processing, the task is much easier to implement. IWF allows relative novice programmers to develop image processing tools and perform experimentation without the drudgery of learning all the details of windows programming an without needing to develop low-level routines first. The target for this software effort is to help engineers, scientists and students develop code quickly an easility. A secondary goal of this effort is to build a highly structured software tool which is based on a modular design. Each software project can be fit into place and modified without changing the primary code for the program.

One of the most important problems to overcome the tendency to re-invent the wheel. This occur quite often as researchers re-write low level routine which should generally be available to them. This library tries to alleviate this problem by providing a set of low level routines as well as some high level ones. The low level routines are stored in the kernel library. The high level routines are found in the control library and provide the basic support for the IWF software.

This software in its present form is distibuted free of charge and can be copied as many times as you like. I would classify it as Freeware and hope that people try it out for their experimentation. In return I hope to get constructive feedback from those that use as to how it can be improved.

Their were originally many tools attached to the IWF before it was packaged but some of these tools have been removed for two reasons. Some tools were not complete and will be added in later updates of the WorkFrame. Others because they were either too specific or propriatory were left out. This does not hamper the primary function of the IWF which is to provide programmers with easy access to image data to develop their own custom tools without the need to deal with display, storage and retreival.

Hardware Requirements

Image Workframe will run on any machine which is equipped with MS Window 3.1, but for optimum performance I recommend a minimum 80386 or higher processor with 800x600

256color display system using small fonts. I have found that IWF works best on an 80486 system with 1024x768 256color system with small fonts.

I. THE IMAGE DESKTOP

Image Workframe (IWF) has been specifically designed to perform image processing on a PC using the Windows operating environment. **IWF** provides a programming framework which creates and image desktop. The desktop is a multi-image interface (MII) which can accomodate up to 10 images simultaneously (The upper limit is not really known, 10 was chosen as a reasonable upper limit). Each image is contained in its own window and exhibits much of the behavior of a standard window. The desktop is controlled exclusively by IWF and only specific access is provided for user modules. Access to the image data is obtain from the handles to the image windows themselves. You can add you own project code to IWF as a module. The modules are invoked and controlled by a popup window or dialog which can have its own menu and/or child windows. IWF supports multiple simultaneous modules.

A. Loading/Saving Images

IWF can load images onto the desktop using a number of ile formats. Currently the file format are still limited, but 256 color PCX compatibility is included since it is probably one of the most widely used image formats. If you are using other formats then they will need to be converted using some of the shareware file conversion available.

B. Image Manipulation

The Image menus provides support for geometric operation on images. Geometric manipulation involves the the mapping of pixels into a new location. In order to move the pixels without destroying other pixels which need to be moved, a secondary image called a target or destination image is created. Using a source and destination image and image can be modified and still maintain the original image. This way, if the changes are not satifactory, the original image can be restored. More basic forms of geometric operation involve scaling, mirror, flip, and rotation of the pixel data.

1. Scaling

Image scaling scales the image horizontally and vertically by sampling the original and mapping the sampled value into the destination image. By scaling at different rates for horizontal and vertical, the aspect ration of and image can be changed. This is also useful for aspect ration correction.

2. Mirror

Image mirror operations simply flips the pixel data horizontally.

3. Flip

Image flip operation flips the pixel data vertically

4. Rotation

Image rotation rotates the pixel data about the center of the image.

C. Region Based Tools

Image Workframe has a number of build in tools for working with images. By holding the control key down and using the left button on the mouse, you can interactively set the region of the image for processing. The tools are region based, which means that they act on the processing rectangle which has been defined by the click and drag of the mouse, rather than the entire image as the image. Since the tools are implemented as modeless dialogs. they can be left on the screen and can exist simultaneously. The Edit/Undo command allows you to undo the last operation performed.

1. Palette Manager Tool

Most gray scale images are represented by a set of numbers ranging between 0 and 255 corresponding to an 8 bit / pixel image. Sometimes it is necessary to modify what each of these numbers represents. In a normal view of an image these numbers represent brighness values from 0 meaning black and 255 meaning white. Using a table numbered from 0 to 255, it is possible to transform the meaning of the pixel value into any other value immediately. This table is called a Look Up Table and usually contains 3 numbers per index, one for each the red, green, and blue color values. Using the table it is possible to change pixels from one gray to another or from gray to a color. The process of converting a gray pixel to a color is callled pseudo-coloring. Pseudo coloring is commonly used to bring out features in an image which are normally difficult to see using only gray scales. Manipulation of the display Look Up Table is done using an interactive tool in Image Workframe called Palette Manager.

Palette Manager is a tool I have implemented and included as part of IWF and is implemented as a modeless dialog module. The control bar at the top of the client area controls the indexes for the black and white level. By clicking the mouse on this bar you can move the indexes up and down the scale. It follows all the conventions of the user modules and receives the mouse messages from the active image as described about. It is a very useful tool for manipulation gray palettes and shows what can be done from popup dialog under IWF.

The palette manager has evolved from many applications where the image brightness needed to be balanced. What is unique about this tool is it is completely interactive. Most tools for palette manipulation are destructive in nature, once the palette has been modified its original information is destroyed. IWF use a dual table image structure which keeps a source palette and a display palette. The source palette is then mapped into the display palette interactively by the Palette Manager. If you don't like what you see it is easy to reset the display palette. This makes any changes made to the image non-destructive. Palette manager actually uses a series of tables as mapping functions. These function have specific purposes which are:

A) Palette Table: This is the source palette for the image
B) Usage Table: Defines what range of indexes to use from the image source palette.

C) Gamma Table: Applys a gamma correction factor to an image.
D) Gray Table: Applys a gray mapping function which can be equalization or stretching.
E) Lookup Table: This is the final table which is created by A to D and is used for display.

These tables provide a very powerful way the modify the image by simply pointing and clicking. The Usage table is controlled by setting the B and W edit fields. The are not limited to number between 0 and 255. Larger number can be used to create a palette which wraps more than once. It is also possible to set B greater than W. This will result in the table being used in a reverse order. (Useful for creating negatives). The Gamma table is controlled by the edit field G. By default the camera and monitor are assumed to have a gamma of 0.5. The correction factor is computed by taking the camera gamma and dividing it by the display gamma. The edit field controls the image appearance based on the camera. A value of 0.5 creates a linear table resulting in no change, a value of 0.3 produces an image which expands the darker ranges, a value of 0.7 produces an image which compresses the darker ranges. The Gray table is controlled by the buttons on the bottom of the Histogram and by the bar on the top of the histogram. Clicking on the top bar will set the upper and lower limits for the palette mapping. Bringing the two limits closer together creates a Lookup Table for contrast stretching. The Usage and Gamma tables will automatically map themselves into the Gray range. If equalization is required this can be done by pressing the 'equal' button. This does a histogram equalization over the full range of brightness.

Custom palettes can be loaded into the Palette Table at any time. This is a destructive process because is replaces the original image palette. The Lookup Table can be mapped into the pixel data using the 'apply' button. The result will be a new histogram distribution with the gray palette set to linear. This process is also destructive and should only be used if you need to generate an image with tha changes mapped directly to the pixels.

2. Convolution Tool

Digital filtering techniques are powerful ways of enhancing image an bringing out features in an image which would not normally be visible. There are image filters for use with a whole array of different applications, all designed to do something different with the image information. Convolution of an image is mathematically quite a complex process, however when an image is digitized it is made into discrete elements. As a result the convolution of discrete image simplifies to a process of calculating the new pixel based on its neighbors.

The convolution tool in IWF perform a discrete integer convolution on the area of the image define by the processing rectangle. The processing rectangle is set by holding the control key down and using the left mouse button to click and drag a rectangle. In the display dialog for the too are edit fields showing the coefficients for convolution. Each of the neighboring pixels contributes a percentage of its own value to the calculation of the new pixel. The process is called weighting and the kernel defining how the neighboring pixels are weighted is called the convolution kernel. Each element of the convolution kernel is called a convolution coefficient and is basically a multiplier for that pixel value. The total of all the neighbors is summed and then normalized to produce a new image. Each kernel will have a divisor to normalize the image.

This is shown in the edit field at the bottom.

On the right side of the dialog is a set of radio button which allow you to select a specialized process. This can be Standard filtering, Prewiit filtering, Sobel filtering, or Median filtering. Standard filtering uses the kernel displayed in the edit fields. Both Prewitt and Sobel filters use two kernels to compute the center pixel. Median filtering is not a convolution but rather selects the mean value in the set of pixels and uses that for the center pixel.

3. Profiler Tool

Palette manipulation is a useful way to analyze and enhance images as a whole. When it is necessary to look at specific areas in the image then a more useful approach is to study slices of image brightness. This information can reveal a lot of information about object found in an image. The slices can be viewed in a two dimensional for, with the vertical axis representing brightness and/or gradient and the horizontal axis representing position.

The Profiler is a simple but useful tool built into Image Workframe for examining slices of an image. It allows you to look at the brightness or gradient of an image along a line. The line is controlled by the mouse pointer. To active the line make sure the profile window is the active module, then click anywhere on the target image. As you move the cursor around you will see a horizontal or vertical line being drawn on the screen. Depending on where you move the cursor the line will toggle from horizontal to vertical. To mark the end point simply click again. The module will then draw a brightness and/or a gradient line on the image.

4. Render Tool

The render tool is a little like a profile tool in 2 dimensions. What it does is plot the image brightness as the third axis in a surface contour. The surface contour detail can be adjusted to show greater or lesser detail.

Status Bar

Part of the IWF desktop includes a status bar at the bottom edge. This bar provides the user with basic system and image information.

The first part of the status line includes a pictorial representation of the current logical palette realized to the system. It represents all the possiblities of colors and grays which can be displayed on the image desktop. By default if the active image is flagged as grayscale (image->color==FALSE) the IWF will map a gray and color default palette to the system. if the active image is color (image->color==TRUE) then IWF will map the image palette into the system. You may notice that the default palette only has 64 grays instead of 256. This is done for two reasons, one is your eye cannot tell the difference, the other is most 256 color video displays only have 6 bit output DACs (eg. 8514 mode) and as a result they can only produce a maximum of 64

grays (2^6=64). As a result, more than 64 grays will be redundant.

The second part of the status bar gives the dimensions of the active image.
The third part of the status bar gives the current x and y
positions of the mouse cursor when it is overtop the active image. At those positions the pixel data number and the corresponding red green and blue palette entries are given. In addition a set of three meter bars shows the relative amounts of red green and blue for that pixel.

The forth part of the status bar is a comment line for comments contains in the image file.

User File Types and 16 bit images

A user defined file type is provided for reading in different types of raw, uncompressed image files. You can select width, height, extension, header size, and bits per pixel. These fields are stored in the file **iwf.ini** and can be edited using notepad. When reading in 16 bit image files they will be reduced to 8 bit by shifting the bit planes down by the value given in the shift bit field. Depending on the dynamic range of the image you may want to change this number to maximize the number of quantization levels in the display image. Overflow will be wrapped around.

The shift bits field also affects how IWF will read astronomy file types FTS and SBI. Future versions of IWF are planned to have full 16 bit grayscale processing capabilities to target astronomy and medical imaging.

II. Programming with IWF

Libraries

There are three main libraries which make up IWF. They are compiled for **medium memory model.** Your compiler must be set up to compile for medium model.

control.lib - High level control code for IWF
kernel.lib - Low level processing code.
module.lib - Special user modules included with IWF.

The following section describes the library procedures provided ny IWF for working with images ans windows. They are part of the kernel library in IWF. These procedures are the only ones need to do just about anything with the images on the desktop.

Special Window Procedures: (macros in iwf.h)

```
GetImage()     - Gets the current image attached to an image window
```

```
SetImage()     - Sets a new image to the image window
GetScale()     - Gets the current scaling factor for the image window
SetScale()     - Sets the image scaling factor for the window
RefreshImage() - Refreshes the image window are a change has been made.
```

The special window procedures are used for connecting images with windows. The image windows are created by the IWF but any image can be attached to it. All you need to do is get or set the image to the window and the rest of the display process and messaging is handled by IWF. The image is not automatically updated by IWF until you call for a refresh. This way the program avoids any unnecessary repainting which can make a program appear sluggish and clumsy.

Once you have the image structure then it is possible to access and manipulate the data for the image. The structure IMAGE is defined in the the header file kernel.h. Setting the scaling and calling *RefreshImage()* for the image window will cause the window to automatically scale the display to that size (given in %). This does not scale the data, only the way it is displayed on the screen. The file imageio.c was included so that you can look at the syntax and see how to access the image data from these functions. It does not need to be included in the project because it is already in the library kernel.lib. The image structure contains all relevant information about the image being displayed and maintains a handle to a data block containing the pixel information. This is the link to all the image data. **imageio.c** demostrates how the information can be accessed using function calls or directly using the handle (I don't recommend using the latter since there usually is little or nothing to be gained by direct access except in intensive looping operations). In looping intensive processes which work one pixel at a time the best way to speed it up is to extract a piece of the image to a local buffer first and then process it. If this is still too slow than direct access may be necessary.

Image Procedures

The following routines allow you to manipulate the image data and create new ones. The syntax for each one is outline in imageio.c. and geom.c

Creation Procedures:

```
CreateImage()    - Creates an image which can be used for storing pixel data.
DestroyImage()   - Frees an image used for storing pixel data.
ClearImage()     - Clears an image to zero.
DuplicateImage() - Creates a new image which is a duplicate structure
```

Input Output Procedures:

```
CopyImageData()  - Copies an image pixel data to a destination image.
CopyImagePal()   - Copies an image palette to a destination image
GetImagePixel()  - Gets a pixel from an image at x,y and stores it in dn.
SetImagePixel()  - Sets a pixel in an image to a value.
GetImageScan()   - Gets a raster line from an image at line y and stores it at dn.
GetImageRow()    - Gets a row of pixels from an image and stores it at dn.
PutImageRow()    - Puts a row of pixels from a buffer into an image
SetImageRow()    - Sets a row of pixels to a value
GetImageCol()    - Gets a column of pixels from an image and stores it at dn.
```

```
PutImageCol()    - Puts a column of pixels from a buffer into an image
SetImageCol()    - Sets a column of pixels to a value
GetImageRect()   - Cuts a rectangle from an image and stores it in the buffer.
PutImageRect()   - Pastes a rectangle from the buffer to an image.
SetImageRect()   - Sets a rectangular region of pixels to a value
```

Geometric Procedures:

```
ReduceImage()    - Creates a new image which is
a reduction of the original.
ScaleImage()     - Creates a new image which scales the original into it.
MirrorImage()    - Creates a new image which is a mirror of the original
FlipImage()      - Creates a new image which is a flip of the original
RotateImage()    - Creates a new image which is a rotation of the original
```

Processing Rectangle:

The image structure has built into it a RECT structure for defining what part of the image is to be processed. By holding the control key down and using the left button on the mouse, you can interactively set the region of the image for processing. This is built in so that a user module does not need to provide an interface to select a region. All the user module needs to do is use the coordinates which are set in the image structure.

```
image->rc.left
image->rc.top
image->rc.right
image->rc.bottom
```

By default these coordinates always start up set to the entire image region. They can be set at any time when the image is the active image, or by the user module itself. Most of the built in function are designed to operate on the entire image, but a few use these coordinates to determine what to process.

Messaging

In addition to getting and manipulating images, IWF provides messaging to the user module so that it is possible to interact with the image using the mouse. The desktop uses the idea of an active image to determine how messages are sent to the user modules, otherwise it would be very difficult to coordinate all the simultaneous images. The active image window is shown by the active title bar and is set by double clicking in the client area of the image window you wish to become active. The active image window passes mouse click and mouse move messages along to the active user module. The handle of the image window sending the message is passed in the *wParam* parameter of the message. All the user module needs to do to determine if the message is native to the window or from and image is call *IsWindow(wParam)* and it will return true if *wParam* is from a valid window (i.e. and image window) otherwise it is native to the user window.

Mouse messages sent by the active image to the user modules are:

WM_LBUTTONDOWN
WM_MBUTTONDOWN
WM_RBUTTONDOWN
WM_LBUTTONUP
WM_MBUTTONUP
WM_RBUTTONUP
WM_MOUSEMOVE

Double click messages are not passed on and are reserved for changing the state of the image windows. The **lParam** parameter will contain the logical coordinates of the image where the mouse cursor is pointing. Even if the image display is zoomed the mouse will point to the correct co-ordinates. In addition **wParam** will contain the window handle of the active window rather than the special key state of the keyboard. The special key state of the keyboard is reserved for the image window to allow for re-direction of the mouse messages. **wParam** can be used to determine where the mouse message is originating. We have found that this set of messages combined with your own set of controls is abundant for manipulating the image from the user modules.

IWF maintains global variables which are accessible from any point. They are:

hWndSrc - Handle of the active image window.
hWndDst - Handle of the destination image window
hWndMod - Handle of the active user module window to send messages

These can be used to access the active image data without using the wParam from mouse messages. Within the IMAGE structure is a RECT structure which defines the active area of interest. To set this area hold the control key down and click and drag across the active image window.

You can get control of the mouse messages which are sent from the active image and have them sent to your application. This can be done by simply setting

```
hWndMod = hWndMyModule
```

which will re-direct the messages to your window. This can be done at any time. Only one module can be active at a time so the previous module will lose the mouse messages from its input queue. I recommend that each module have some way that the user can select it to be active. (for image windows I use MB_LBUTTONDBLCLK)

Adding a User Module

Building modules to work under IWF is relatively straightforward. Two examples have been include to demonstrate what is involved. One shows how to build a popup window which is owned by the IWF desktop window. The second shows how to build a modeless dialog for IWF.

Both have very similar window procedures since they are both quite similar from the viewpoint of the system. I personally prefer the modeless dialog method since controls are very easy to add using a resource toolkit. If you use a modeless dialog make sure that you do not call the default window procedure like it is done for the popup window (An easy mistake to make). If you select the dialog class to be **BorDlg** you no longer receive WM_PAINT messages in you dialog procedure. For this reason I did not use the subclass for my user modules such as the ***Palette Manager***. Your menu control is implemented into the popup window rather than the main menu. This allow the menu to be encapsulated within the user module and keeps the IWF menu separate. Remember, whenever you want the active image to update itself to reflect the changes made to the data, call InvalidateRect() to force a re-paint of the image.

When you are finished with you new user module the last thing to do is hook it onto the IWF so that you can launch it from the desktop. This is done my editing the file module.h and adding in the defines for each module such as

```
#define  MODULE1  MyCreationProc
#define  NAME1    "MyModule"
#define  DIALOG1  hWndMyDialog (only needed for modeless dialogs)
```

where MODULE1 defines the name of the window creation procedure, NAME1 is the name to put in the launch button on the desktop, and DIALOG1 is the handle to the window for modless dialogs. The last define is required only if the module is a modeless dialog so that the message dispatcher can send it to the dialog.

To bind you code to the IWF:

1. Create a project file called **iwf**
2. Add **control.lib, kernel.lib, modules.lib and bwcc.lib** are in the project
3. Add **iwf.c** (main c file), **iwf.rc** (main resource file) and **iwf.def** to the project
4. Any any of your code and/or resources to the project and resource file.
5. Add your module definitions to **module.h**.
6. Build the new project.

The base project file would look like

iwf.c
iwf.rc
iwf.def
modules\samp_dlg.c
modules\samp_win.c
modules.lib
control.lib

bwcc.lib
+ any additional user code and resources

If you want to build the sample modules add the resources using 'Add to project' in Resource workshop for each one you want to add to the main resource file. Make sure you have the **bwcc.dll** (Borland Windows Custom Controls) library in your default directory or thw Windows system directory or else the dialogs won't appear. Two example modules are provided to demonstrate how to construct a user window and a user modeless dialog under IWF and how to get messages from the image window.

Disclaimer

No warranty has been written or implied for the use of this software. I have done my best to produce code which is bug free, however since this is only a side line for me undoubtedly there will be a few things which slip my eye. I rely mostly on my students as beta testers, which is not always the best. I will try to address bugs as they come to me any update the code as soon as possible. I have crashed the code more times during its creation than I have had hot dinners, and so far, no nuclear meltdowns. Naturally I cannot be responsible for the behavior of a programming tool if things go awry, but I am quite certain it will be relatively harmless.

The best way of all to become familiar with using ***Image WorkFrame*** is to study the source code examples provided. In there is all the information to begin building you own projects. I invite anyone who builds something that they feel is useful to other people to e-mail me and it could be included in IWF giving credit to the author. Any suggestions for improvements or additions are also welcome. You can e-mail me at:

cmuller@ccu.umanitoba.ca

Since I am continually in the process of upgrading the documentation, if you e-mail me then I can send you any new additions to the documents right away. (You can never have enough documentation when it comes to free software).