IMPLEMENTATION AND PERFORMANCE EVALUATION OF INTEL VTUNE IMAGE PROCESSING FUNCTIONS IN THE MATLAB ENVIRONMENT

Phaisit Chewputtanagul, David J. Jackson, Kenneth G. Ricks
Electrical and Computer Engineering
The University of Alabama
Tuscaloosa, AL 35487-0286 USA
jjackson@eng.ua.edu

Abstract

Many current general purpose processors use extensions to the instruction set architecture to enhance the performance of digital image processing and multimedia applications. In this paper, a MATLAB Application Program Interface (API) is constructed for the performance evaluation of Intel image processing functions. The Intel image processing functions are constructed for high performance operations which have been optimized for Intel architecture processors. Several functions are developed and categorized within the following groups: arithmetic and logical comparison operations, morphological operations, geometric operations, filtering operations, point operations and linear transform operations.

We present a comparison of image processing functions between MATLAB M-files, built-in functions and functions based on the Intel image processing library.

Keywords: MATLAB, digital image processing.

1. INTRODUCTION

Image processing has, since the late 1970's, been a field of great interest to many researchers. This is mainly due to the need of humans to improve information for visual interpretation and the need for accurate machine interpretation of image content. The advancements achieved in the various components of the computer such as disk drives, processor, memory, and software have each contributed to this growth.

Historically, engineers and scientists searching for image processing software solutions have had to choose between C/C++ libraries or graphical user interface (GUI) based packages. Due to the fixed functionality and cross-platform incompatibility of these alternatives, many technical professionals have been faced with developing their own algorithms and implementations from scratch. MATLAB [1] has extensive built-in image processing and analysis functionality together with an open and extensible environment.

MATLAB is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language. MATLAB includes functions for data analysis and visualization, numeric and symbolic computation, engineering and scientific graphics modeling, simulation and prototyping, programming and application development, and GUI design.

MATLAB is used in a variety of application areas including signal and image processing [2], control system design, financial engineering, and medical research. Although MATLAB is a complete, self-contained environment for programming and manipulating data, it is often useful to interact with data and programs external to the MATLAB environment. MATLAB provides an Application Program Interface (API) to support these external interfaces. This API has been used throughout this research.

MATLAB has a useful collection of elementary built-in functions for image processing. Nevertheless, MATLAB execution is slow in terms of clock cycles/pixel for typical image processing functions. In contrast, the Intel image processing library [3, 4] provides a set of low-level image manipulation functions in standard dynamic linked library (DLL) and static library forms. The functions are optimized for Intel architecture processors and are particularly effective at taking advantage of MMX technology [5]. MMX is a Pentium microprocessor technology that enhances the execution of multimedia applications.

There are two primary causes of slow speed for MATLAB functions. The first problem results from algebraic loops. The solutions to algebraic loops are iterative and code interpretation is performed at every time step. The second problem is that M-files are evaluated at every time step. By converting the M-file to a MEX-file (MATLAB Executable) the performance can be greatly improved. Our study focuses on the construction of C MEX-files incorporating the Intel image processing library functions to improve performance of image processing functions within the MATLAB environment.

2. INTEL IMAGE PROCESSING LIBRARY

The Intel image processing library provides software developers with image primitives that are commonly required by desktop publishing, machine vision, digital video, image compression and image printing applications. The library has been optimized for various Intel Architecture processors, including Pentium, Pentium with MMX™ technology, Pentium Pro, Pentium II, Pentium III and Pentium IV processors. A separate DLL is available for each of these processors.

The library functions provide filtering, thresholding, arithmetic, logical, and morphological operations. In addition, a number of functions provide linear geometric and linear image transforms. The linear image transforms include the fast Fourier transform (FFT) and the discrete cosine transform (DCT). The library uses a flexible image format, supporting channels of 1-, 8-, 16-, and 32-bit integer pixels, as well as 32-bit floating-point pixels, with an arbitrary number of channels per image. Conversion to and from the Windows* DIB (device-independent bitmap) image format is supported, as are conversions between color and gray scale images.

A summary of the library's major features and benefits is shown in Table 1. Table 2 lists the functions available in the Intel image processing library.

Table 1. Summary of the Intel image processing library features and benefits.

Feature	Benefit
Provides common low-level image processing routines	Helps reduce time to market
Provides functions optimized for a specific Intel processor	Allows obtaining optimal product performance with a specific processor
Detects a specific processor and loads processor-specific DLL	Allows use of a single scalable executable file for a number of processors
Provides flexible image format	Integration of the library with existing designs
Supports image tiling	Allows handling large images and monitoring application memory

Table 2. Functions available in the Intel image processing library.

Arithmetic and Logical Comparison Operations	Add, Subtract, Multiply, Shift, Square, Scale, And, Or, Xor, Alpha Composite, IsGreater, IsEqual, IsLess
Filtering Operations	Blur, Prewitt, Sobel, Laplacian, 2D Convolution, Median, Max, and Min Filters
Transforms	DCT, FFT
Morphological Operations	Erode, Dilate, Open, Close
Color Space Conversion	DIB (.bmp), Bitonal/Gray, Gray/Color, HLS, HSV, CIELab (XYZ), CIELUV, YUV, YCrCb, color twist, bit depth conversion
Point Operations	Threshold, Contrast Stretch, Compute and Equalize Histogram, Color Twist
Image Statistics	Moments, Norms, MSE
Geometric Operations	Zoom, Decimate, Rotate, Mirror, Shear, Affine, Perspective, Remap, Decimate-and-filter

3. C-MEX FILES

C subroutines can be called from MATLAB as if they were built-in functions. MATLAB callable C programs are referred to as MEX-files. MEX-files, for the Windows environment, are DLLs that the MATLAB interpreter can automatically load and execute.

MEX-files allow applications such as large pre-existing C programs to be called from MATLAB without having to be rewritten as M-files. In addition, bottleneck computations that do not run fast enough in the interpreted MATLAB environment can be recoded in C for efficiency. C MEX-files are created using an API provided by MATLAB.

The two components of a MEX-file include a computational routine and gateway routine. The MEX-file must contain the *mex.h* header so that the entry point and interface routines are declared properly. Figure 1 illustrates the inputs and outputs of MEX-files.

Inputs   nrhs : number of right-hand side arguments   prhs : pointer to right-hand side arguments
Outputs   nlhs : number of left-hand side arguments   plhs : pointer to left-hand side arguments

Figure 1. Inputs and outputs of MEX-files.

The variables *nlhs* and *nrhs* specify the number of left-hand and right-hand side arguments that are used when the MEX-file is called. For example, if the MEX-file was called in MATLAB using

[a,b] = mymexfunction(d, e, f);

nlhs equals 2 (for a and b), and *nrhs* equals 3 (for d, e, and f). The variable *prhs* is an array of pointers to the input arrays. Thus, in the above example, *prhs[0]* is a pointer to the array d, and *prhs[1]* is a pointer to the array e, and *plhs* is an uninitialized array. If *plhs* is greater than zero, the MEX-Files should fill the *plhs* array with pointers to arrays containing valid data, or generate an error message stating that too many outputs have been requested. This is further illustrated in Figure 2.

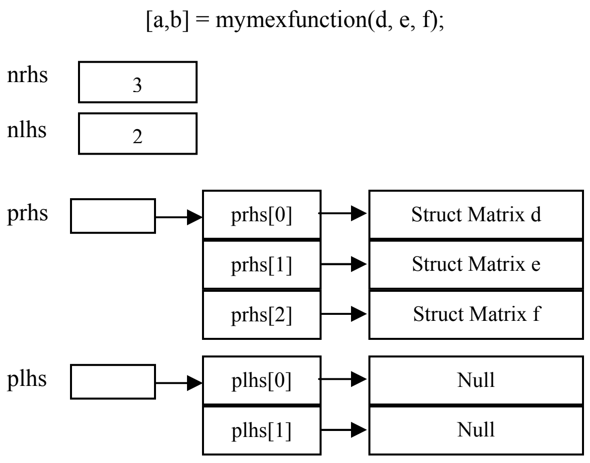

Figure 2. Illustration of the function of *plhs* and *prhs* arrays.

The name of the gateway routine must always be *mexFunction* and must contain these parameters:

```
void mexFunction(   int nlhs, mxArray *plhs[],
                    int nrhs, const mxArray *prhs[])
        {
                / * more C code */
        }
```

The parameters **plhs[]* and **prhs[]* are vectors that contain pointers to the left- and right-hand arguments of the MEX-files. Both are declared as type "*mxArray*", which implies that these variables are MATLAB arrays.

4. IMAGE PROCESSING OPERATIONS

Our study deals with several image processing operations which can be categorized in six parts: arithmetic and logical comparison operations, morphological operations, geometric operations, filtering operations, point operations, and linear transforms.

4.1 Arithmetic and Logical Operations

These operations modify pixel values using arithmetic or logical operations. All these functions use a single input image to create an output image. Table 3 lists the functions that perform arithmetic and logical operations.

Table 3. Arithmetic and logical operations.

Function	Description
AddS	adds a constant to pixel values of the source image
SubtractS	subtracts a constant from pixel values of the source image
MultiplyS	multiplies pixel values of the source image by an integer constant
MultiplySScale	multiplies pixel values by a constant and scales the products
Not	computes a bitwise NOT of pixel values
AndS	computes a bitwise AND of each pixel's value and a constant
OrS	computes a bitwise OR of each pixel's value and a constant
XorS	computes a bitwise XOR of each pixel's value and a constant
Abs	returns absolute pixel values of the source image
Square	squares the pixel values of the source image
RshiftS	shifts pixel values' bits to the left
LshiftS	divides pixel values by a constant power of 2 by shifting bits to the right

4.2 Morphological Operations

The morphological operations include simple erosion and dilation of an image. A specified number of erosions and dilations are performed as part of image opening or closing operations to eliminate or fill small and thin holes in objects, break objects at thin points or connect nearby objects, and generally smooth the boundaries of objects without significantly changing their area. Table 4 lists the functions that perform morphological operations.

4.3. Geometric Operations

These operations perform geometric transforms to resize the image, change the image orientation, or warp the image. Table 5 lists supported geometric transform functions.

Table 4. Morphological operations.

Function	Description
Erode	sets each output pixel to the minimum of the corresponding input pixel and its 8 neighbors
Dilate	sets each output pixel to the maximum of the corresponding input pixel and its 8 neighbors
Open	performs a number of erosions followed by the same number of dilations of an image
Close	performs a number of dilations followed by the same number of erosions of an image

Table 5. Geometric operations.

Function	Description
Rotate	rotates the image
Zoom	magnifies (zooms) the image
Decimate	shrinks (decimates) the image
Resize	resizes the image
Mirror	finds a mirror image
Shear	performs a shear of the source image

4.4. Filtering Operations

These operations perform linear and non-linear filtering operations. Most linear filtering is performed through convolution, either with user-defined convolution kernels or with the provided fixed filter kernels. Table 6 lists the filtering functions.

Table 6. Filtering operations.

Function	Description
Blur	applies a simple neighborhood averaging filter to blur the image
MinFilter	applies a minimum filter to an IPL image
Max Filter	applies a maximum filter to an IPL image
Median Filter	applies a median filter to an IPL image
FixedFilter	applies a commonly used (predefined) filter to an image
Convolve2D	convolves an IPL image with one or more convolution kernels

4.5. Point Operations

These operations perform on an image on a pixel-by-pixel basis and include threshold and histogram functions. Table 7 lists functions for point operations.

Table 7. Point operations.

Function	Description
ComputeHisto	computes the image intensity histogram
HistoEqualize	equalizes the image intensity histogram
ContrastStretch	stretches the contrast of an image using an intensity transformation
Threshold	performs a simple thresholding of an image

4.6. Linear Transforms

These operations implement the forward and inverse Fast Fourier Transform (FFT) and Discrete Cosine Transform (DCT). Table 8 lists the functions for performing linear image transform operations.

Table 8. Linear transforms.

Function	Description
RealFft2D	computes the forward or inverse 2D real fast Fourier transform of an image
DCT2D	computes the forward or inverse 2D discrete cosine transform of an image

The functions are written, as library routines, using Microsoft Visual C++ 6.0 and the Intel Image Processing Library V2.1. The designs are written as C MEX-files to be executed in the MATLAB environment. C MEX-file is built by using the MEX script to compile C source code with additional calls to API routines.

5. APPLICATION AND RESULTS

5.1 GUI

A GUI is developed to display all six operation classes. The primary purpose of the GUI is to allow observation of the execution time (in clock cycles/pixel) for each function and the results of applying the function to an input image. It is written and designed within MATLAB as shown in Figure 3.

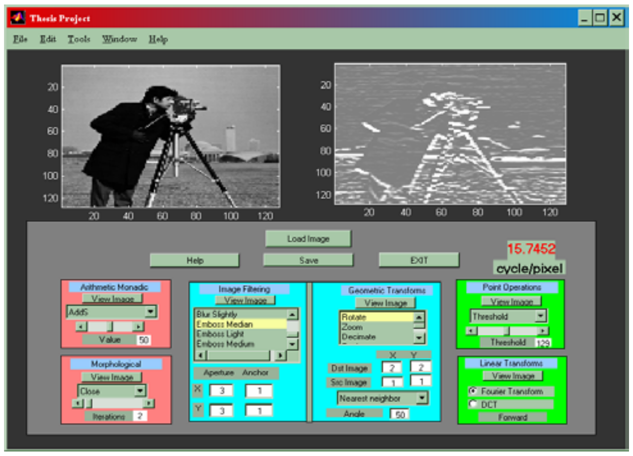

Figure 3. GUI layout (performing emboss median filtering).

5.2 Results

The Intel image processing library has been a source of dramatic improvements for image processing functions. Specifically, when a function is called consecutive times, the function will normally perform faster than the first time. For example, the contrast stretching function uses 913.89 clock cycles/pixel for the first pass processing time. The function spends only 4.3 clock cycles/pixel in the subsequent runs.

The execution time performance is limited by the memory bandwidth [6], regardless of the processor speed. On the first execution, the processor has to load and initialize the DLL and fetch the image data from main memory to cache before applying the Intel image processing functions to the raw image data. On the other hand, subsequent function executions can run the function immediately since the image data already resides in the cache. Table 9 illustrates the performance comparison for the first pass and second pass execution times.

Table 9. Performance comparison for first time and second time execution.

Function	**Clock cycle/pixel for first run (P III 550 MHz)**	**Clock cycle/pixel for second run (P III 550 MHz)**	**Overall Speedup**
- Erode	1242	19.5	63.69
- Blur	992	44	22.55
- Zoom	1127	14.4	78.26
- Contrast Stretching	913.89	4.3	212.53
- Fourier Transform	1049	64	16.2

The Intel image processing functions perform faster than the M-files or built-in MATLAB functions based on size of the image. Figure 4 presents results of the analysis between the Intel image processing functions and the native MATLAB implementation. The contrast stretching function was used to perform this analysis. The sample values (r1, s1) and (r2, s2) were created as (90, 50) and (160, 130), respectively. The "cputime" built-in MATLAB function was used to measure time (in seconds) for two functions, an M-file function and the Intel image processing library function. Figure 5 and Figure 6 illustrate a similar analysis for a threshold and Laplacian function, respectively.

6. CONCLUSIONS

In this paper, we evaluate the performance of the MATLAB functions and the Intel image processing functions within the MATLAB environment. We use several image processing functions to test overall speedup. The operations chosen for implementing are various arithmetic monadic, morphological operations, filtering operations, geometric operations, point operations, and linear transforms.

Based on our observations, we found significant overall speedup when the Intel image processing function is called consecutive times. In addition, the functions also execute in less time according to the size of the image.

Image Size	**MATLAB functions (in ms)**	**IPL functions (in ms)**	**Overall Speedup**
32*32	31.9	1.1	29.00
64*64	86.3	2.2	39.23
128*128	312	5.0	62.40
256*256	1784	15.4	115.84
512*512	11449	61.5	186.16

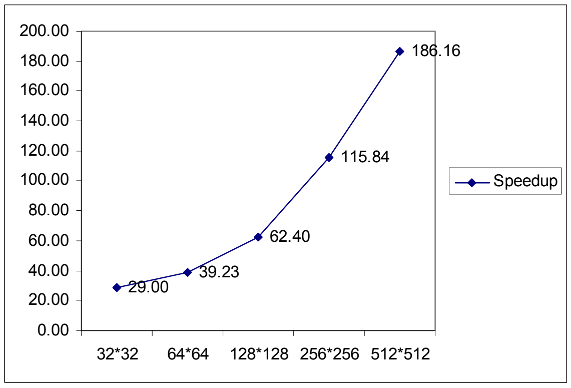

Figure 4. Performance comparison analysis between functions (contrast stretch).

Image Size	MATLAB functions (in ms)	IPL functions (in ms)	Overall Speedup
32*32	160	1.1	145.45
64*64	332	1.6	207.50
128*128	990	4.3	230.23
256*256	4230	14.9	283.89
512*512	21470	58.8	365.14

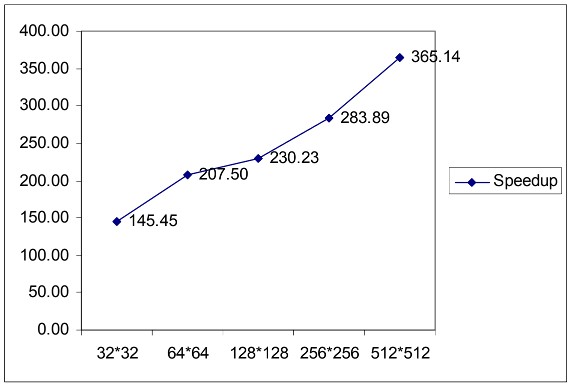

Figure 5. Performance comparison analysis between functions (threshold).

Image Size	MATLAB functions (in ms)	IPL functions (in ms)	Overall Speedup
32*32	72	1.1	65.45
64*64	220	1.6	137.50
128*128	874	3.3	264.85
256*256	3691	9.3	396.88
512*512	22140	35.7	620.17

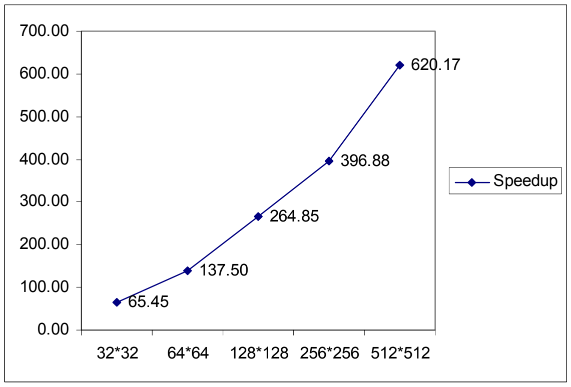

Figure 6. Performance comparison analysis between functions (Laplacian).

REFERENCES

[1] The Math Works Inc, *Application Program Interface Guide*, Version 5.0, Natick, MA.

[2] Robert C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Second edition, Addison-Wesley, 1992.

[3] Intel Corporation*, Intel Image Processing Library Reference Manual*, Order Number 663791-003, USA.

[4] Intel Corporation, *Intel Image Processing LibraryV2.1*,URL:http://developer.intel.com/vtune/perflibst/ipl/index.htm.

[5] Bhargava, R.; John, L.K.; Evans, B.L.; Radhakrishnan, R, *"Evaluating MMX Technology Using DSP and Multimedia Applications"*, Microarchitecture, 1998. MICRO-31. Proceedings. 31st Annual ACM/IEEE International Symposium on , 1998

[6] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1990.