4.18

Software for Image and Video Processing

K. Clint Slatton
The University of Florida

Brian L. Evans
The University of Texas at Austin

1 Introduction

Digital systems that process image data generally involve a mixture of software and hardware. For example, Digital Video Disc (DVD) players employ audio and video processors to decode the compressed audio and visual data, respectively, in real time. These processors are themselves a mixture of embedded software and hardware. The DVD format is based on the MPEG-2 video compression and AC-3 audio compression standards, which took several years to finalize [refer to chapter 6.4]. Before these standards were established, several years of research went into developing the necessary algorithms for audio and video compression. This chapter describes some of the software that is available for developing image and video processing algorithms.

Once an algorithm has been developed and is ready for operational use, it is often implemented in one of the standard compiled languages such as C, C++, or Fortran for greater efficiency. Coding in these languages, however, can be time consuming because the programmer must iteratively debug compile-time and run-time errors. This approach also requires extensive knowledge of the programming language and the operating system of the computer platform on which the program is to be compiled and run. As a result, development time can be lengthy. To guarantee portability, the source code must be compiled and validated under different operating systems and compilers, which further delays development time. In addition, output from programs written in these standard compiled languages must often be exported to a third-party product for visualization.

Many available software packages can help designers shorten the time required to produce an operational image or video processing prototype. Algorithm development environments (Section 2) can reduce development time by eliminating the compilation step, providing many high-level routines, and guaranteeing portability. Compiled libraries (Section 3) offer high-level routines to reduce the development time of compiled programs. Source code (Section 4) is also available for imaging applications and may be assembled into compiled libraries. Visualization environments (Section 5) are especially useful when manipulating and interpreting large data sets. A wide variety of other software packages (Section 6) can also assist in the development of imaging applications.

2 Algorithm Development Environments

Algorithm development environments strive to provide the user with an interface that is much closer to mathematical notation and lexicon than general-purpose programming languages, such as C, C++, or Fortran. The idea is that a user should be able to write the desired computational instructions in a native language that requires relatively little time to master. Also, graphical visualization of the computations should be fully integrated so that the user does not have to leave the environment to observe the output. This section examines four widely used commercial packages: MATLAB, Interactive Data Language (IDL), LabVIEW, and VisiQuest (formally Khoros). For comparison of the styles of specifying algorithms in these environments, Figs. 1–4 show examples of computing the same image processing operation using MATLAB, IDL, LabVIEW, and VisiQuest, respectively.

2.1 MATLAB

MATLAB software is produced by The MathWorks, Inc., and has its origins in the command-line interface to the LINPACK and EISPACK matrix libraries developed by Cleve Moler in the late 1970s [1]. MATLAB interprets commands, which shortens programming time by eliminating compilation. The MATLAB programming language is a vectorized language, which means that it can perform many operations on numbers grouped as vectors or matrices without explicit loop statements. Vectorized code is more compact, more efficient, and parallelizable. The current major version is version 7, released during the summer of 2004.

Versions 1–4 of MATLAB assumed that every variable was a matrix. The matrix could be a real, complex, or string data type. Real and complex numbers were stored in double-precision floating-point format. A scalar would have been represented as a 1×1 matrix of the appropriate data type. A vector is a matrix with either one row or one column. Version 5 added signed and unsigned byte data types, which dramatically reduced storage in representing images. MATLAB provides other numeric data types, such as signed and unsigned 16-bit, 32-bit, and 64-bit integers and 32-bit single-precision floating-point numbers, as well as the ability to define mixed-type data structures, such as cell arrays and structures. Numeric arrays of arbitrary dimension are also supported. Version 6 added object-oriented programming constructs such as operator overloading.

The MATLAB algorithm development environment interprets programs written in the MATLAB programming language, but a compiler for the MATLAB language is available as an add-on to the basic package. When developing algorithms, it is generally much faster to interpret code than to compile code because the developer can immediately test changes. In this sense, MATLAB can be used to rapidly prototype an algorithm. Once the algorithm is stable, it can be compiled for faster execution, which is especially important for handling large data sets. The MATLAB compiler MATCOM converts native MATLAB code into C++ code, compiles the C++ code, and links it with MATLAB libraries. The compiled code executes up to ten times faster than the interpreted code when run on the same machine [2]. The more vectorized the original MATLAB program is, the smaller the speedup in the compiled version. Highly optimized vectorized code may not experience any speedup at all.

The MATLAB algorithm development environment provides a Java-based graphical user interface (GUI) interface, an interpreter for the MATLAB programming language, an extensive set of common numerical and string manipulation functions, 2D and 3D plotting functions, and the ability to build customized GUIs. The standard GUI interface provides a command line sub-window and list of variables in memory. The layout can be customized. User-defined MATLAB functions can be added by creating a file of the same name as the new function that has a ".m" extension and that specifies the number of arguments and returned values using the function keyword. Alternatively, a ".m" file can contain a script of commands or standalone program. For faster computation, users may dynamically link C routines as MATLAB functions through the MEX utility. As an alternative to the command-line interface, the MATLAB environment offers a "notebook" interface that integrates text and graphics into a single document.

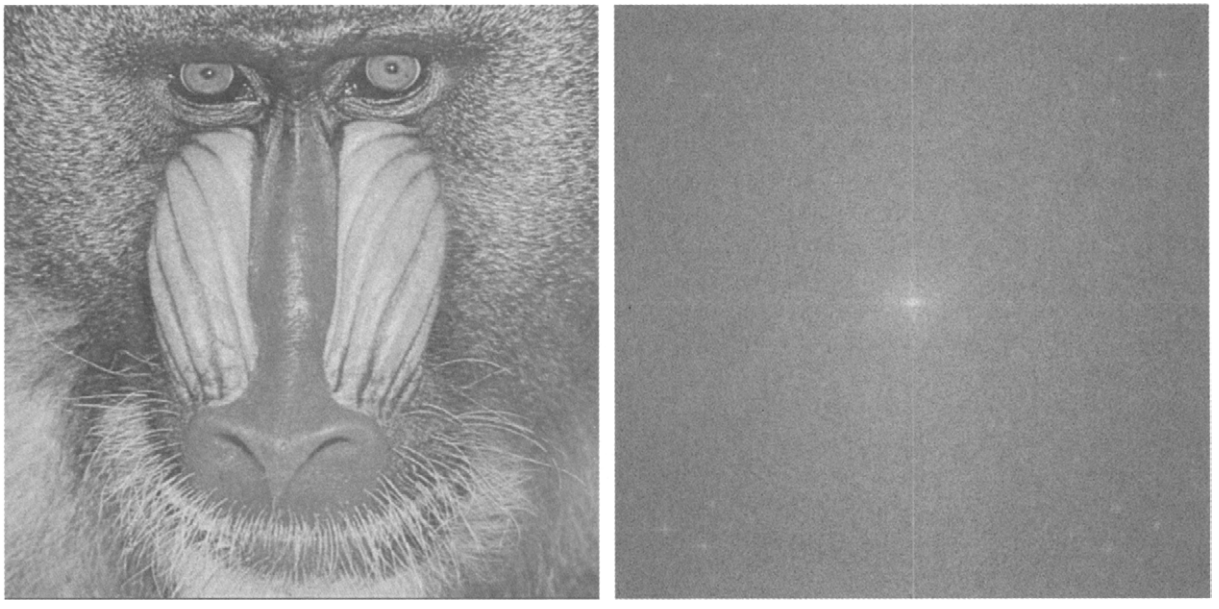
MATLAB toolboxes are available as add-ons to the basic package and greatly extend its capabilities by providing application-specific function libraries [1,3]. The *Signal Processing Toolbox* provides signal generation; finite impulse response (FIR) and infinite impulse response (IIR) filter design; linear system modeling; 1D fast Fourier transforms (FFTs), discrete cosine transforms (DCTs), and Hilbert transforms; 2D discrete Fourier transforms; and windows, spectral analysis, and time-frequency analysis. With Version 6, extremely efficient FFT algorithms [4] were introduced.

The *Image Processing Toolbox* represents an image as a matrix. It provides image file input/output for TIFF, JPEG, and other standard formats. Morphological operations, DCTs, FIR filter design in two dimensions, and color space manipulation and conversion, are also provided. The Canny edge detector [refer to chapter 4.10] is also available through the *Image Processing Toolbox.* The *Wavelet Toolbox* implements several wavelet families, 1D and 2D wavelet transforms, and coding of wavelet coefficients. Additional toolboxes with functionality useful in imaging systems are available in control system design, neural networks, optimization, splines, and symbolic mathematics. The *Image Acquisition Toolbox* provides functionality to acquire video from PC-compatible frame-grabber cards.

MATLAB's strength in developing signal and image processing algorithms lies in its ease of use, powerful functionality,

and data visualization capabilities. Its programming syntax has similarities to C and Fortran. Because the MATLAB programming language is imperative, specifying algorithms in the MATLAB language biases the implementation towards software on a sequential machine. Using the *SIMULINK* add-on, designers can visually specify a system as a block diagram to expose the parallelism in the system. Each block is implemented as MATLAB or C code. *SIMULINK* is well suited for simulating and designing continuous, discrete, and hybrid discrete/continuous control systems [5]. *SIMULINK* has advanced ordinary differential equation solvers and supports discrete-event modeling. Because of the run-time scheduling overhead, simulations of digital signal, image, and video processing systems in *SIMULINK* are extremely slow when compared to a simulation using the MATLAB programming language.

MATLAB runs on Windows, Macintosh, Linux, and Unix operating systems including HP-UX, IBM AIX, SGI, and Sun Solaris. The MathWorks Web site (http://www.mathworks.com) contains freely distributable MATLAB add-ons. The MATLAB newsgroup is comp.soft-sys.matlab. Due to the large number of MATLAB users, several books are available that address signal and image processing with specific emphasis on MATLAB examples and problem sets [6, 7]. An example of a MATLAB command sequence to compute the 2D FFT of an image is shown in Fig. 1.



(a) (b)

```
file_id = fopen('mandrill.raw','r');
fsize = [512,512];
[I1,count] = fread(file_id, fsize, 'unsigned char');
I1=I1';
figure, image(I1); axis off, axis square, colormap(gray(256))
map = 0:1/255:1;
map = [map', map', map'];
imwrite(I1, map,'mandrilltiff.tif', 'tiff')
I2=fft2(I1);
I2=abs(I2);
I2=log10(I2+1);
range=max(max(I2))-min(min(I2));
I2=(255/range)*(I2-min(min(I2)));
I2=fftshift(I2);
figure, image(I2); axis off, axis square, colormap(gray(256))
imwrite(I2, map, 'mandrillFFTtiff.tif', 'tiff')
```

(c)

FIGURE 1 MATLAB example: (a) image, (b) FFT, and (c) code to display images, compute the FFT, and write out the TIFF images. Note images are displayed on the screen inside a figure window that containd pull down menus that allow point and click annotation, and interactive zoom, and save and print operations.

2.2 IDL

The Interactive Data Language (IDL) by Research Systems, Inc., is based on the APL computer language [8]. IDL provides a computer language with built-in data visualization routines and pre-defined mathematical functions. Of the algorithm development environments discussed in this chapter, IDL most closely resembles a low-level language such as C. IDL commands can be issued sequentially in an interactive mode, but, unlike MATLAB programs, IDL programs must be recompiled before being executed each time a change is made to the code. So the advantage of IDL is not ease of algorithm development so much as providing a tremendously powerful integrated data visualization package. IDL is probably the best environment for flexible visualization of very large data sets. The current major version is Version 6, released in 2003.

Arrays are treated as a particular data type so that they may be operated on as single entities, which reduces the need to loop through the array elements [8]. The basic IDL package consists of a command-line interface, low-level numerical and string manipulation operators (similar to C), high-level implicit functions such as frequency-domain operations, and many data display functions. The development environment on Windows includes an integrated drag-and-drop *GUIBuilder* for rapidly designing interfaces. The *GUIBuilder* generates IDL code that runs on any supported platform. IDL instructions and functions are put into ".pro" files. Although IDL syntax may not be as familiar to C and Fortran programmers as MATLAB's syntax, it offers streamlined file access and scalar variables do not need to be explicitly declared. IDL's new Intelligent Tools, or *iTools*, functionality provides a flexible set of interactive utilities to easily open, analyze and visualize data and produce graphics.

The new version of IDL supports additional file formats, including medical imaging (DICOM) and AutoCAD (DXF), as well as hierarchical scientific data formats including HDF, HDF-EOS and netCDF. GeoTIFF and ESRI shapefiles are also supported. IDL supports aggregate data structures in addition to scalars and arrays. Data types supported include 8-bit, 16-bit, 32-bit, and 64-bit integers, 32-bit and 64-bit floating-point numbers, and string data types [8]. Many standard image formats, such as JPEG and TIFF are supported. Formatted I/O allows access to any user-defined ASCII or binary format. IDL supports arrays having up to 8 dimensions. Many important image processing functions are provided, such as 2D FFTs, wavelets, and median filters. IDL I/O supports MPEG video coding, but does not provide an explicit DCT function.

IDL provides dynamic linking to external C and Fortran functions, which is analogous to the MEX utility in MATLAB. IDL, however, does not have an automated method for converting code into another language. For users that work with those languages and wish to access IDL's capabilities, it is possible to call IDL as a subroutine library or graphics engine from their programs.

Research Systems Inc. offers several complementary software packages written in IDL. Some of these can stand alone, while others are add-ons. The stand alone Envi package (see Section 5) and the add on *Wavelet Toolkit* package extend IDL's signal and image processing functionality, while other packages, such as *IDL Dataminer* and *ION*, provide capabilities such as database management and data sharing over the Internet.

IDL runs on Windows, Macintosh, Linux and Unix operating systems including HP-UX, IBM-AIX, SGI-IRIX, and Sun Solaris. The Research Systems Inc. Web site (http://www.rsinc.com) contains IDL libraries written by third parties, some of which are freely distributable. The IDL newsgroup is `comp.lang.idl`. An example of an IDL command sequence to compute the 2D FFT of an image is shown in Fig. 2.

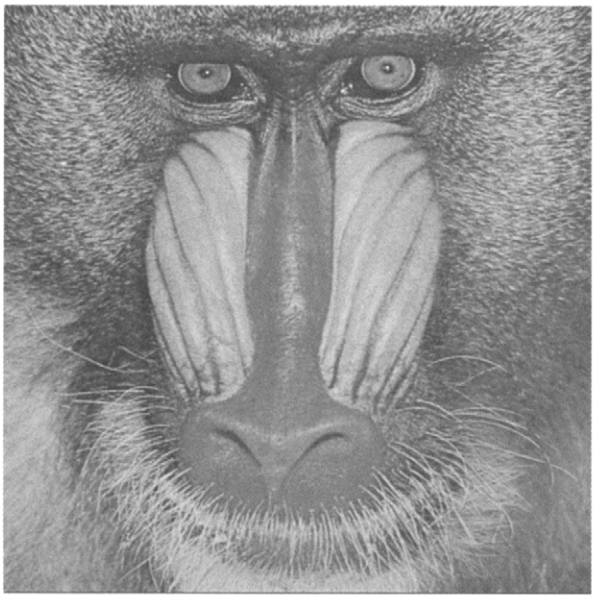
2.3 LabVIEW

LabVIEW, produced by National Instruments, is based on visual programming using block diagrams [9], unlike the default text-based interfaces for MATLAB and IDL. LabVIEW was originally developed for simulating electronic test and measurement equipment. For example, it has specialized I/O and data-handling routines for many serial-port standards and data acquisition systems. The current major version is Version 7, which was released in May 2003.

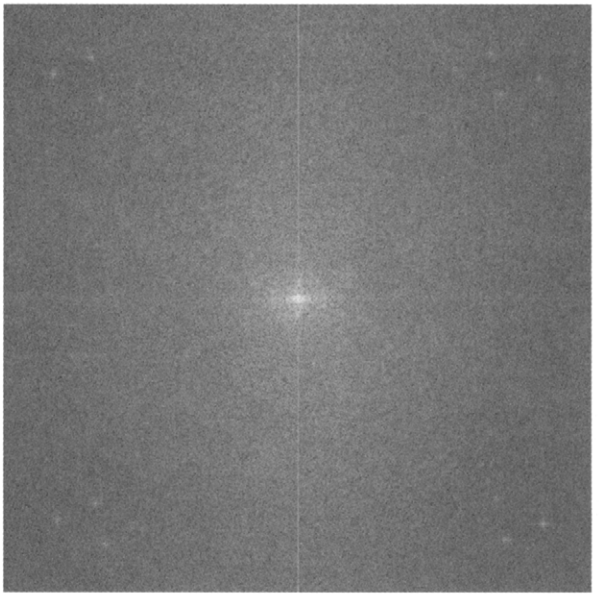
LabVIEW is primarily an interactive environment. The basic interface is called a virtual instrument (VI). A VI is analogous to a function in a conventional programming language. Rather than being defined by lines of text as a MATLAB program, a VI consists of a graphical user interface with a dataflow diagram representing the source code and icon connections that allow the VI to be referenced by other VIs. This programming structure naturally lends itself to hierarchical and modular computing.

In a VI, the dataflow diagram follows the G dataflow language. Dataflow languages naturally capture the functional parallelism in digital signal, image, and video processing systems [10]. The G dataflow language provides one representation that is simultaneously useful for simulation on desktop computers and synthesis onto embedded systems. On a desktop computer, G is by default compiled onto a multithreaded implementation and executed by the operating system. The same G program can be retargeted to embedded software on PDAs and reconfigurable hardware implemented by field programmable gate arrays.

A VI dataflow diagram consists of nodes and wires. Each wire communicates one data value in one direction. The datatype on a wire cannot change during program execution. Supported numeric data types include 8-bit, 16-bit, 32-bit,



(a)



(b)

```
file1 = 'mandrill'
lun = 1;
openr, lun, file1
pic = bytarr(512,512)
readu, lun, pic
close, lun
picr = rotate(pic,7)
tiff_write,'mandrilltif', pic
window,0,xsize=512,ysize=512, title='mandrill 512x512'
tvscl,picr,0,0
picrf = fft(picr,-1)
picrfd = abs(picrf)
picrfd = alog10(picrfd+1.0)
picrfd = sqrt(sqrt(sqrt(picrfd)))
range = max(picrfd)-min(picrfd)
picrfd = ((255/range)*(picrfd-min(picrfd)))
picrfd = shift(picrfd,256,256)
tiff_write,'mandrillFFTtif', picrfd
window,1,xsize=512,ysize=512, title='fft 512x512'
tv, picrfd, 0, 0
return
end
```

(c)

FIGURE 2 IDL example: (a) image, (b) FFT, and (c) code to display images, compute the FFT, and write out the TIFF images.

and 64-bit integers, and 32-bit and 64-bit floating-point numbers [9]. Strings, arrays, matrices, and other data structures are also supported. Nodes compute outputs from inputs. A node is enabled for execution when one token is available on each input (homogeneous). A node will not produce data until all internals of the node have completed. Only one node can write a value onto a particular wire, but many nodes can read from a wire.

LabVIEW provides several pre-defined nodes. For example, boxed structures provide structured programming constructs. The case box is similar to the switch statement in C. The iteration box realizes for and while loops. The sequence box forces a sequential ordering according to how nodes are grouped. The event box reacts to system events (e.g., mouse clicks and keyboard input). The formula box allows the developer to enter multiple lines of ANSI C code to compute

a formula. Shift registers may be added at I/O terminals of boxed structures to realize feedback.

LabVIEW provides several ways other than wires to communicate data: local variables, global variables, and first-in first-out (FIFO) queues. Local variables are visible only inside a VI. Since LabVIEW programs are multithreaded, reads from and writes to local and global variables are protected through mutual exclusion and always complete. First-in first-out (FIFO) queues may be utilized to guarantee real-time input/output performance.

LabVIEW has extensions in the form of toolkits. Toolkits primarily provide application-specific demonstrations and libraries of VIs. Currently shipping toolkits include *Modulation, Signal Processing,* and *DSP Test Integration Toolkit.* The *Modulation Toolkit* includes analog and digital building blocks for communication systems. The *Signal Processing Toolkit* contains VIs for wavelet and filter bank design, joint time-frequency analysis, digital filter design, and spectral analysis [9]. The *DSP Test Integration Toolkit* provides an interface to the Texas Instruments digital signal processor development environment and boards for automation of real-time testing of embedded software implementations.

Some toolkits provide new graphical views more natural for certain applications or subsystems. The views are ultimately converted to G before they are executed, which allows the views to be mixed with other G diagrams. The *State Diagram Toolkit* provides finite state machine specifications in the form of bubble (state) and arc (transition conditions) diagrams. Some image and video compression standards have involved finite state machines mixed with the dataflow processing (e.g., MPEG-2 compression of video frames). The Simulation Interface Toolkit provides differential equation solvers and converters from MATLAB's *SIMULINK* diagrams to Simulation Interface views. The Simulation View has natural applications in control system simulation as well as partial differential equation approaches to image processing (e.g., optical flow calculations).

The *Vision Development Module* supports image acquisition, browsing, and processing. Gray scale, color, and binary images are supported. Image formats include bitmap, TIFF, JPEG, and PNP. Image sequences can be read and written in AVI format. Image processing operations include histogram calculations and histogram equalization; edge detection and filtering; geometric transformations; fast Fourier transform; resampling; morphological operations; pattern and shape matching; and blob analysis.

The *Vision Development Module* contains formats for analog video standards like PAL and NTSC. The module does not provide certain important functions such as the 2D DCT, but the 2D DCT can be computed from the 2D FFT. The module provides an interactive environment for developing prototype image processing applications without programming.

LabVIEW's image handling capabilities are limited compared to those of MATLAB and IDL. LabVIEW does not directly access raw binary image files. The raw binary files must be read into LabVIEW as 2D arrays and then converted into standard formats such as BMP, TIFF, JPEG, or PNG before LabVIEW can access them. Once image data has been converted into a standard format, many operations, such as logarithms, cannot be implemented directly. Image data must first be converted into an array structure, then the operation is performed, and finally the array is converted back to a standard image format. Also, operations on complex-valued data are limited. To take the absolute value of a complex-valued image, the user must explicitly multiply the data with its complex conjugate and then take the square root of the product.

Of the four algorithm development environments discussed in this section, LabVIEW would be the best suited for integration with hardware, especially for data acquisition. LabVIEW VIs can be compiled and downloaded into embedded real-time data acquisition systems. For example, National Instruments offers a Compact Vision System (*CVS*) embedded high-performance processor. *CVS* is a compact machine vision/image processing system that runs the *LabVIEW RT* (Real-Time) module. Image processing functions (VIs) can be developed using the *Vision Development Module* and *LabVIEW RT* and then run on *CVS* for real-time imaging applications.

LabVIEW is available for Windows, Macintosh, and UNIX operating systems (e.g., HP-UX and Sun Solaris). However, many of the add-on packages such as *IMAQ Vision* and *HiQ* are not available on UNIX platforms. *Application Builder* is a LabVIEW extension that provides the capability to create shared libraries (DLLs) from VIs. The National Instruments Web site (http://www.natinst.com) contains freely distributable add-ons for LabVIEW and other National Instruments software packages. The LabVIEW newsgroup is `comp.lang.labview`. An example of a LabVIEW command sequence to compute the 2D FFT of an image is shown in Fig. 3.

2.4 VisiQuest

VisiQuest by the AccuSoft Corporation was offered as Khoros by Khoral Research, Inc. prior to May 2004.[1] VisiQuest is another visual programming environment for modeling and simulation [11]. The block diagrams use a mixture of dataflow and control flow. VisiQuest supports 8-bit, 16-bit, and 32-bit integer and 32-bit and 64-bit float data types. VisiQuest is written in C, and supports calls to external C and C++ code. A variety of toolboxes are available for VisiQuest that provide capabilities in I/O, data processing,

[1]The main changes in the Accusoft version pertain to the open source function library. The library has been migrated to LAPACK for enhanced linear algebra support. No functionality has been removed.

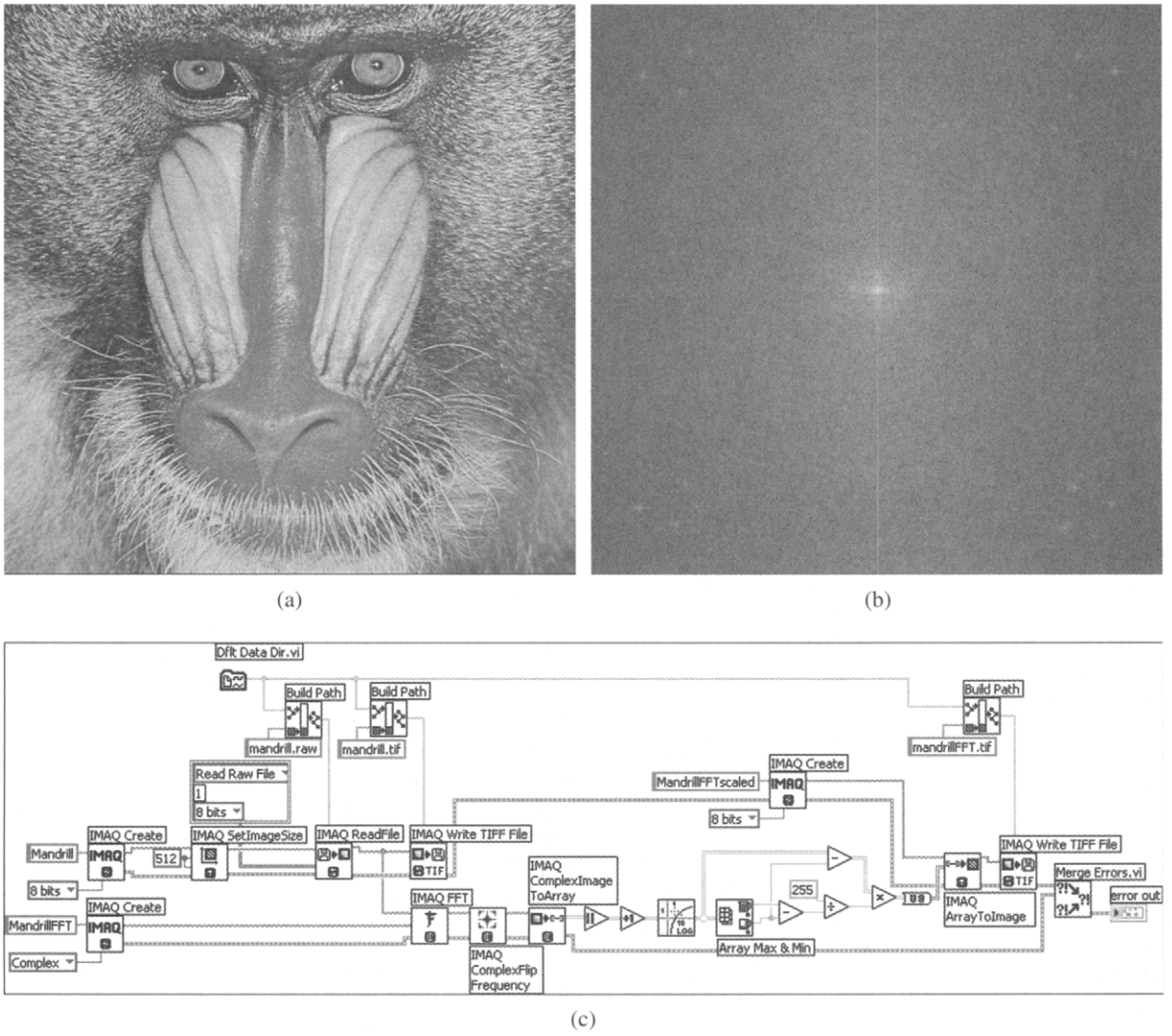


FIGURE 3 LabVIEW example: (a) image, (b) FFT, and (c) code to display images, compute the FFT, and write out the TIFF images.

data visualization, image processing, matrix operations, and software rendering. The current major version is version 3.5, released in 2004.

VisiQuest libraries are effectively linked to the graphical coding workspace through a flow control tool[2]. When the flow control tool is run, a workspace window appears with several action buttons and pull-down menus along its periphery. The action buttons allow the user to run and reset the program. The pull-down menus access mathematical and I/O functions, called "subforms". Once the user selects a subform and specifies the input parameters, it is converted into an icon, referred to as a "glyph", and appears in the workspace. A particular glyph will perform a self-contained task such as generating an image or opening an existing file containing image data. Another glyph may perform a function such as a 2D FFT. Custom glyphs can be written in C or C++ (as standalone programs, routines or libraries), Java (as a library, applet, application only, or application and applet), or Fortran, as a script in csh, sh, ksh or Perl. Arrow buttons on the glyphs represent input and output connections. To perform an operation such as an FFT on an image, the user connects the output port of the image-accessing glyph to the input port of the FFT operator glyph. This is the primary manner in which larger algorithms are constructed.

The *Datamanip Toolbox* provides data I/O, data generation, trigonometric and nonlinear functions, bitwise and complex math, linear transforms (including FFTs), histogram and morphological operators, noise generation and processing, data

[2]The Khoros libraries are still linked to the graphical workspace by the *Cantata* flow control tool in VisiQuest. However, the name *Cantata* has been dropped, so the name VisiQuest refers to the Visual Programming Environment as well as the overall product. All other product components have kept their names.

clustering (data classification), and convolution. *Datamanip* also provides general multi-dimensional (up to five dimensions) data manipulation operators. *Datamanip* requires that the *Bootstrap* and *Dataserv* toolboxes also be loaded. The *Image Toolbox* provides median filtering, 2D frequency domain filtering, edge detectors, and geometric warping, but no DCT. Many of the matrix operations that are useful in image processing are only available in the *Matrix Toolbox*. The *Geometry Toolbox* includes a 3D rendering engine based on geometric primitives. It is no longer the main plotting toolbox for 2D and 3D plots, which are handled now by a separate toolbox *Envision*. VisiQuest comes bundled with most of the toolboxes relevant to signal and image processing.

The *Xprism Pro* package runs independently of VisiQuest, but is meant to complement the VisiQuest product by extending 2D and 3D visualization capabilities beyond *Envision*. *XPrism Pro* uses dynamic rendering so that large data sets can be viewed at variable resolution. Most other environments require large data sets to be explicitly downsampled to enable rapid plotting. Other add-on toolboxes offer wavelets and formats for accessing geographic information system (GIS) data. The strength of VisiQuest is that the user can develop complete algorithms very rapidly in the visual programming environment, which is significantly simpler than that of LabVIEW. The weakness is that this environment biases designs towards execution in software.

VisiQuest allows extensive integration with MATLAB through its *Mfile Toolbox*, making MATLAB functions and programs available to VisiQuest. This toolbox is available on most, but not all, of the platforms on which VisiQuest is supported. The MATLAB programs can be treated as source code inside VisiQuest objects. The user is required to have the MATCOM compiler for converting MATLAB code to C++ code. It also supports single and double-precision float calculations, but not all MATLAB functionality is supported.

VisiQuest runs on Windows (via a Unix emulator[3]), Linux, and Unix (SGI, and Sun Solaris) operating systems. A native Windows version will be released in fall 2004. The AccuSoft Web site is (http://www.accusoft.com). A newsgroup for VisiQuest has not yet been established, but the Khoros newsgroup is `comp.soft-sys.khoros`. An example of a VisiQuest command sequence to compute the 2D FFT of an image is shown in Fig. 4.

3 Compiled Libraries

Whether users are working in an algorithm development environment or writing their own code, it is sometimes important to access mathematical functions that are written in low-level code for efficiency. Many libraries containing mathematical functions are available for this purpose. Often, a particular library will not be available in all languages or run on all operating systems. In general, the source code is not provided. Object files are supplied, which must be linked with the user's program during compilation. When the source code is not available, the burden is on the documentation to inform the user about the speed and accuracy of each function.

3.1 Intel

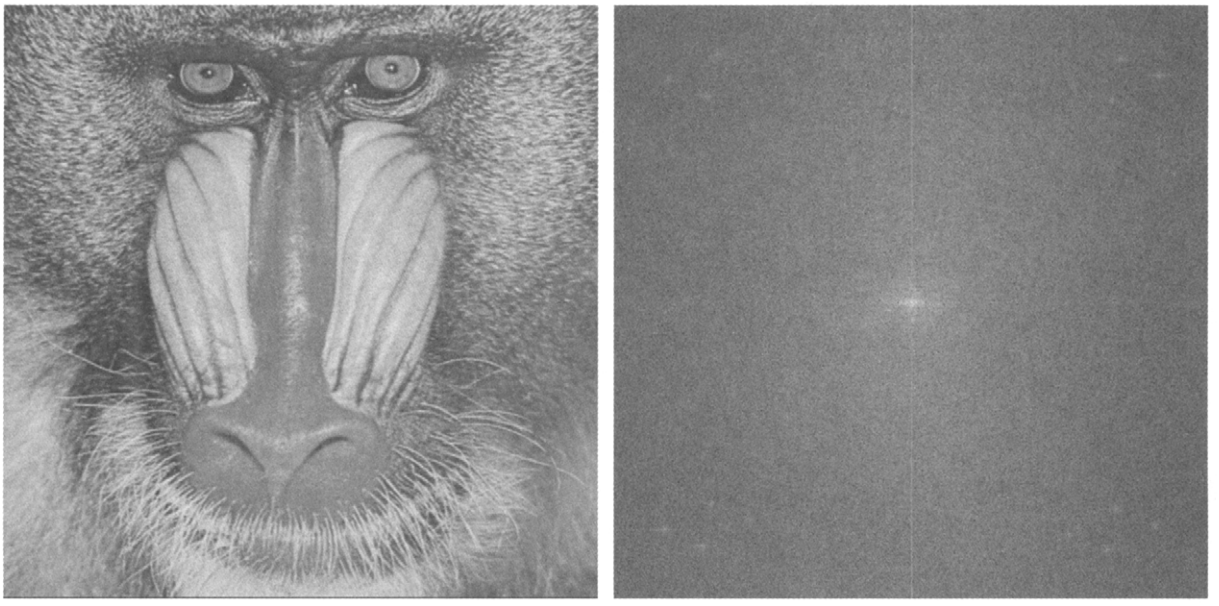
Intel offers free-trial libraries (http://developer.intel.com/software/products/perflib/index.htm) for signal processing, image processing, pattern recognition, general mathematics, and JPEG image coding functions. These functions have been compiled and optimized for a variety of Intel processors. The libraries require specific operating systems (Microsoft Windows 2000 and XP and RedHat Linux) and C/C++ compilers (Intel, Microsoft, or GNU). Signal processing functions include windows, FIR filters, IIR filters, FFTs, correlation, wavelets, and convolution. Image processing functions include morphological, thresholding, and filtering operations as well as 2D FFTs and DCTs.

When the Intel library routines run on a Pentium processor with MMX, many of the integer and fixed-point routines will use MMX instructions [12]. MMX instructions compute integer and fixed-point arithmetic by applying the same operation on eight 8-bit words or four 16-bit words in parallel. In MMX, eight 8-bit additions, four 16-bit additions, or four 16-bit multiplications may be performed in parallel. Switching back and forth to the MMX instruction set incurs a 30-cycle penalty. The use of MMX generally reduces the accuracy of answers, primarily because Pentium processors do not have extended precision accumulation. Furthermore, many of the library functions make hidden function calls, which reduces efficiency. When using the Intel libraries on Pentium processors with MMX, the speedup for signal and image processing applications varies between 1.5 and 2.0, whereas the speedup for graphics applications varies from 4 to 6 [13].

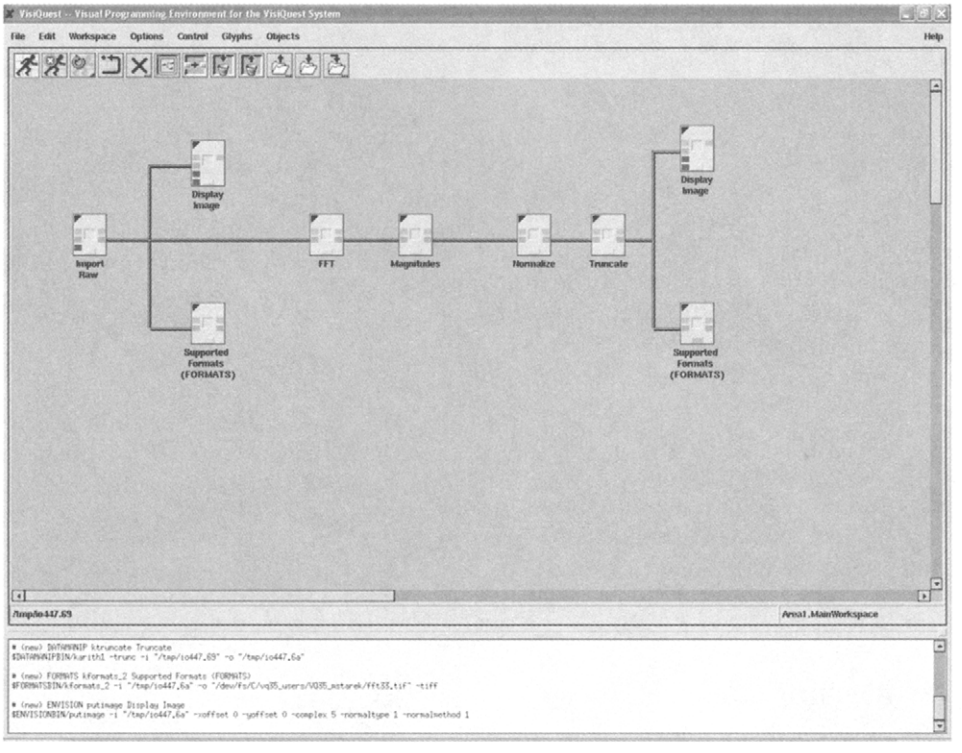
3.2 IMSL

Other math libraries are available that are not specialized for signal and image processing applications, but contain useful functions such as FFTs and median filtering. The most prevalent is the family of IMSL libraries provided by Visual Numerics Inc. (http://www.vni.com/). These libraries support Fortran, C, and Java languages. These libraries are very general. As a result, over 65 computer platforms are supported.

[3]Windows Services for Unix (SFU) 3.5 allows UNIX and Linux packages to run on Windows machines.



(a) (b)



(c)

FIGURE 4 VisiQuest example. (a) Image, (b) FFT, and (c) code to display images, compute the FFT, and write out the TIFF images.

4 Source Code

The source code for mathematical functions and image processing applications are available. This section describes two sets of available source code besides the source code that comes with the algorithm development environments listed in Section 2.

4.1 Numerical Recipes

Numerical Recipes by the Numerical Recipes Software Company (http://www.nr.com) provides source code in Fortran, C, and C++ languages for a wide variety of mathematical functions. As long as users have the appropriate compiler on their machine, these programs can be run on any computer platform. It is the user's responsibility to write the proper I/O commands so that their program can access the desired data. The tradeoff for this generality is the lack of optimization for any particular machine and the resulting lack of efficiency. The algorithms are not tailored for signal and image processing applications, but some common functions supported are 1D and 2D FFTs, wavelets, DCTs, Huffman encoding, and numerical linear algebra routines.

4.2 Encoding Standards

Information regarding the International Standards Organization (ISO) image coding standard developed by the Joint Photographic Experts Group (JPEG) is available at the Web site (http://www.jpeg.org/). Links to source code for the JPEG and JPEG 2000 encoding and decoding algorithms for C, C++, and Java can be found at that Web site. Information regarding the ISO encoding standards for audio/video developed by the Moving Picture Experts Group (MPEG) is available at the Web site (http://www.mpeg.org/). Links are available to the source code for the encoding and decoding algorithms. These programs can be used in conjunction with the algorithm development packages mentioned previously, or with low-level languages.

5 Specialized Processing and Visualization Environments

In addition to the general purpose algorithm development environments discussed above, many packages exist that are highly specialized for processing and visualizing large data sets. Some of these support user-written programs in limited native languages, but most of their functionality consists of pre-defined operations. The user can specify some parameters for these functions, but typically cannot access the source code. Generally, these packages are specialized for certain applications, such as remote sensing, seismic analysis, and medical imaging. We examine packages that are specialized for remote sensing applications as examples.

Remote sensing data are typically comprised of electromagnetic (sometimes acoustic) energy that has been modulated through interaction with objects. The data are often collected by a sensor mounted on a moving platform, such as an airplane or satellite. The motivation for collecting remotely sensed data is to acquire information over large areas not accessible via *in situ* methods. This method of acquiring data results in very large data sets. When imagery is collected at more than one wavelength, there may be several channels of data per imaged scene. Remote sensing software packages must often handle data sets several gigabytes in size. Although a multi-channel image constitutes a multidimensional data set, these packages usually only display the data as images. These packages generally have very limited 2D and 3D graphics capabilities. However, they do contain many specialized display and I/O routines for common remote sensing formats that other types of software do not have. They have many of the common image processing functions such as 2D FFTs, median filtering, and edge detection. They are not very useful for generalized data analysis or algorithm development, but can be ideal for processing data for interpretation without requiring the user to learn any programming languages or mathematical algorithms.

In addition to some of the common image processing functions, these packages offer functions particularly useful for remote sensing. In remote sensing, images of a given area are often acquired at different times, from different locations, and by different sensors. To facilitate an integrated analysis of the scene, the data sets must be co-registered so that a particular sample (pixel) will correspond to the same physical location in all of the channels. This is accomplished by choosing control points in the different images that correspond to the same physical locations. Then 2D polynomial warping functions or spline functions are created to resample the child images to the parent image. These packages contain the functions for co-registering so that the user does not need to be familiar with the underlying algorithms.

Another major class of functions that these packages contain is classification or pattern recognition. These algorithms can be either statistics-based, neural network–based, or fuzzy logic–based. Classifying remote sensing imagery into homogeneous groups is very important for quantitatively assessing land cover types and monitoring environmental changes, such as deforestation and erosion.

When users have large remotely sensed data sets in sensor-specific formats, and need to perform advanced operations on it, working with these packages will be quicker and easier than working with the algorithm development packages. Several remote sensing packages are available. We will discuss two of the most widely used and powerful packages: Geomatica and Envi. Other popular packages include ERDAS, ERMapper, PV-Wave, and ESRI.

5.1 Geomatica

Geomatica software is a geomatics software package developed by the PCI Geomatics company. It supports imagery, geographical information systems (GIS) data, and many ortho-projection tools [14]. Geomatica supports many geographic and topographic formats such as universal transverse mercator, and can project the image data onto these non-uniform grids so that they match true physical locations. The current major version is version 9.

Both command line and graphical interfaces are used depending on the operations to be performed. Low-level I/O routines make it easy to import and export data between Geomatica and other software packages in either ASCII or binary format. Geomatica provides a limited native language so that some user-defined operations can be performed without having to leave the Geomatica environment. Geomatica functions can be accessed by programs in other languages, such as C, via linking commands.

Most common image formats, such as JPEG and TIFF, are supported, as well as formats for particular sensors. Image files can have up to 1024 channels. Data represented by the image pixels is referred to as raster data. Raster data types supported include 8-bit and 16-bit (signed and unsigned) integers and 32-bit floating-point numbers. In addition to raster data, Geomatica also supports vector data. Geomatica vectors are collections of ordered pairs (vertices) corresponding to locations on the image. The vectors define piecewise linear paths that can be used to delineate exact boundaries among regions in the image. These lines are independent of the pixel size because they are defined by the mathematical lines between vertices. Vectors can be used to draw precise elevation contours and road networks on top of the imagery.

Geomatica supports specialized 2D and 3D data visualization. Geomatica also supports "data fly throughs" allowing the user to scan over the data from different vantage points. Geomatica has a complete set of co-registering and mosaicking functions, and standard image filtering and FFT routines. Geomatica also includes its own drivers for accessing magnetic tape drives for reading data. Some applications for which Geomatica is well suited include watershed hydrological analysis, flight simulation, land cover classification.

Geomatica is available on Windows (NT, 2000, XP), Linux, and Unix operating systems including, SGI-IRIX and Sun Solaris. The Web site (http://www.pcigeomatics.com/) contains demonstration and image-handling freeware, as well as a subscriber discussion list `discuss-request@pcigeomatics.com`.

5.2 Envi

The Environment for Visualizing Images (Envi) by Research Systems, Inc., is written in IDL. However, it is not necessary to acquire IDL separately to run Envi because a basic IDL engine comes bundled with Envi. Envi has a menu-driven graphical user interface. Although batch operations are possible, it is best suited for interactive work.

Envi supports many of the same features and capabilities as Geomatica. Geomatica has extensive photogrammetric capability and hydrological analysis options, while Envi has very user-friendly access to its functions and extensive sensor-specific formats. Envi can also be easily integrated with external IDL code. Envi is accessible through the same Web site as IDL.

6 Other Software

Many other software tools are used in image and video processing. For image display, editing, and conversion, X windows tools xv and ImageMagick are often used. The xv program by John Bradley at the University of Pennsylvania (http://www.trilon.com/xv/) is shareware. ImageMagick by John Cristy at E.I. du Pont de Nemours and Company Inc. (http://www.imagemagick.org/) is freely distributable. ImageMagick can also compose images, and create and animate video sequences. Both tools run on a variety of Windows, Linux, and Unix operating systems.

Symbolic mathematics environments are useful for deriving algebraic relationships and computing transformations algebraically (such as Fourier, Laplace, and z transforms). These environments include Mathematica [15] from Wolfram Research Inc. (http://www.wolfram.com) and Maple [16] from Waterloo Maple Software (http://www.maplesoft.com). Mathematica has the following application packs related to signal and image processing: *Digital Image Processing*, *Signals and Systems*, *Wavelet Explorer*, *Global Optimization*, *Time Series*, and *Dynamic Visualizer*. Maple offers a variety of notebooks on engineering and scientific applications, but none of the Maple notebooks relate to signal or image processing. Maple is accessible in MATLAB through its symbolic toolbox. Mathematica and Maple run on Windows, Macintosh, and Unix operating systems. The newsgroup for symbolic mathematics environments is `sci.math.symbolic`. The Mathematica newsgroup is `comp.soft-sys.math.mathematica`. The Maple newsgroup is `comp.soft-sys.math.maple`.

System-level design tools, such as SPW by Cadence (http://www.cadence.com), COSSAP by Synopsys (http://www.synopsys.com), ADS by Agilent Technologies (http://eesof.tm.agilent.com/), and Ptolemy by the University of California at Berkeley (http://ptolemy.eecs.berkeley.edu), are excellent at simulating and synthesizing 1D signal processing systems. Using these tools, designers can specify a system using a mixture of graphical block diagrams and textual representations. The specification may be efficiently simulated or synthesized into software, hardware, or a mixture of hardware and software. These system-level design tools provide

many basic image and video processing blocks for simulation. For example, Ptolemy provides image file I/O, median filtering, 2D FIR filtering, 2D FFTs, 2D DCTs, motion vector computation, and matrix operations. These system-level design tools also provide an interface to MATLAB in which a block in a block diagram can represent a MATLAB function or script. However, these tools currently have limited but evolving support for synthesizing image and video processing systems into hardware and/or software.

Modern discrete sensing modalities, such as scanning laser ranging, have made large data sets of irregularly spaced 3D points more common. These data can only be analyzed using traditional image processing packages by collapsing vertical columns of points into a single value, e.g., the mean value of all points in that column. However, some algorithm development packages, such as MATLAB and IDL, can work with the 3D point data directly, and some new packages have been developed specifically for analyzing 3D point data. QT Viewer is a 3D model generation and manipulation package developed by the Johns Hopkins University Applied Physics Lab (APL) to facilitate real-time manipulation of large amounts of complex 3D data. The Web site is http://www.jhuapl.edu/ott/. Another package, PolyWorks, was developed by InnovMetric (http://www.innovmetric.com/) to register 3D point data obtained from multiple 3D scans.

7 Conclusion

For image and video processing, we have examined algorithm development environments, function libraries, source code repositories, and specialized data processing packages. Algorithm development environments are useful when a user needs flexible and powerful coding capabilities for rapid prototyping of algorithms. Each of the four algorithm environments discussed provides much of the functionality needed for image processing and some of the functionality for video processing. When a user wants to code an algorithm in a compiled language for speed, then function libraries become extremely useful. A wide variety of source code upon which to draw exists as part of algorithm development environments and source code repositories. If there is no need to understand the underlying algorithms, but there is a need to perform specialized analysis of data, then the data interpretation and visualization packages should be used. We also surveyed a variety of other tools for specific tasks. Electronic design automation tools for image and video processing systems are evolving.

Acknowledgements

The authors wish to thank Michael Starek of the University of Florida and Ian Wong and John Schutz of The University of Texas at Austin for running the 2D FFT examples in Figs. 1–4.

References

[1] *The MATLAB 6 User's Guide*, The MathWorks Inc., Natick, MA.
[2] "MATLAB Compiler Speeds Up Development," *MATLAB News & Notes*, Winter 1996.
[3] Duane C. Hanselman and Bruce Littlefield, *Mastering MATLAB 6: A Comprehensive Tutorial and Reference*, ISBN 0-13-019468-9, Prentice Hall, 2001.
[4] http://www.fftw.org/
[5] *The SIMULINK User's Manual*, The MathWorks Inc., Natick, MA.
[6] Samuel Stearns, *Digital Signal Processing with Examples in MATLAB*, CRC Press, 2002.
[7] Rafael Gonzalez, Richard E. Woods, and Steven L. Eddins, *Digital Image Processing Using MATLAB*, Prentice Hall, 2003.
[8] *The IDL User's Manual*, Research Systems, Inc., Boulder, CO.
[9] *The LabVIEW User's Manual*, National Instruments, Austin, TX.
[10] Hugo A. Andrade and Scott Kovner, "Software synthesis from dataflow models for embedded software design in the G programming language and the LabVIEW development environment," *Proceedings IEEE Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 1998, 1705–1710.
[11] *The Khoros User's Manual*, Khoral Research, Inc., Albuquerque, NM.
[12] *Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture*, Intel Corp., http://developer.intel.com/design/PentiumII/manuals/243190.htm.
[13] Ravi Bhargava, Ramesh Radhakrishnan, Brian L. Evans and Lizy K. John, "Evaluating MMX technology using DSP and multimedia applications," *Proceedings IEEE International Symposium on Microarchitecture*, Nov. 30-Dec. 2, 1998, Dallas, TX, pp. 37–46.
[14] *The PCI User's Manual*, PCI, Inc., Ontario, Canada.
[15] Stephen Wolfram, *The Mathematica Book*, 3rd ed., Wolfram Media Inc., Champaign, IL, 1996.
[16] K. M. Heal, M. Hansen and K. Rickard, *Maple V Learning Guide for Release 5*, Springer Verlag, 1997, ISBN 0-387-98397-X.