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 and 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. Available source code (Section 4) for an entire 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 vernacular than general-purpose programming languages. The idea is that a user should be able to write out 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, IDL, LabVIEW, and Khoros. For comparison of the styles of specifying algorithms in these environments, Figures 1-4 show examples of computing the same image processing operation using MATLAB, IDL, LabVIEW, and Khoros, respectively.

2.1 MATLAB

MATLAB software is produced by The MathWorks, Inc., and has its origins in the command-line interface of 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,
meaning 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.

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 x 1 matrix of the appropriate data type. A vector is a matrix with either one row or one column. MATLAB 5 is also vectorized, but adds signed and unsigned byte data types, which dramatically reduce storage in representing images. Version 5 also introduces other data types, such as signed and unsigned 16-bit, 32-bit, and 64-bit integers and 32-bit single-precision floating-point numbers. MATLAB 5 provides the ability to define data structures other than matrices and supports arrays of arbitrary dimension.

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 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 is up to 10 times faster than the interpreted code when run on the same machine [2]. The more vectorized the 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 command-line interface, an interpreter for the MATLAB programming language, an extensive set of common numerical and string manipulation functions, 2-D and 3-D plotting functions, and the ability to build custom graphical user interfaces (GUIs). A user-defined MATLAB function can be added by
creating a file with a “.m” extension containing the interpreter commands. Alternatively, a “.m” file can serve as a 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; 1-D Fast Fourier Transforms (FFTs), Discrete Cosine Transforms (DCTs), and Hilbert Transforms; 2-D Discrete Fourier Transforms; statistical signal processing; and windows, spectral analysis, and time-frequency analysis. 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, and 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, 1-D and 2-D wavelet transforms, and coding of wavelet coefficients. Additional toolboxes with uses in imaging systems are available in control system design, neural networks, optimization, splines, and symbolic mathematics.

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 [4]. 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.


2.2 IDL

The Interactive Data Language (IDL) by Research Systems, Inc., is based on the APL computer language [5]. 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. Even in its interactive mode, IDL programs must be recompiled and 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.

Arrays are treated as a particular data type so that they may be operated on as single entities, reducing the need to loop through the array elements [5]. 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. IDL Insight provides a graphical user interface. IDL instructions and functions are put in “.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** 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 [5]. Image formats supported include JPEG, TIFF, and GIF. 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 2-D 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. Most often, the user that works with those languages and wishes to access **IDL**'s capabilities must write output files from their programs and then read those files into **IDL** for analysis or visualization.

Research Systems Inc. offers several complementary software packages written in **IDL**. Some of these can stand alone, while others are add-ons. Except for the **Envi** package, which is discussed in Section 5, these packages do not typically extend **IDL**'s signal and image processing functionality. Instead, they provide capabilities such as database management and data sharing over the Internet.

**IDL** runs on Windows, Macintosh, and Unix operating systems including DEC Ultrix, HP-UX, SGI, and Sun Solaris. The Research Systems Inc. Web site ([http://www.rsinc.com](http://www.rsinc.com)) contains **IDL** libraries written by third parties, some of which are freely distributable. The **IDL** newsgroup is comp.lang.idl.

### 2.3 LabVIEW

**LabVIEW**, produced by National Instruments, is based on visual programming using block diagrams [6], unlike the default text-based interfaces for **MATLAB** and **IDL**. **LabVIEW** was originally developed for simulating electronic test equipment, so many of its icon and name
conventions reflect that legacy. For example, it has many specialized I/O and data-handling routines for serial-port standards and hardware simulations.

**LabVIEW** block diagrams represent its own native language called G. G may be either interpreted or compiled. G is a dataflow language, which is a natural representation for data-intensive computation for digital signal, image, and video processing systems [7].

**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. Basic data structures available for use in the VIs are nodes, terminals, wires, and arrays [6]. **LabVIEW** supports 8-bit, 16-bit, 32-bit, and 64-bit integers, and 32-bit and 64-bit floating-point numbers [6].

**LabVIEW** has limited data visualization capabilities. The add-on package **HiQ** is required for 2-D and 3-D graphics. The **LabWindows/CVI toolkit** allows the user to generate C code from VIs, which would be linked to **LabVIEW** libraries. A user can add a code interface node function to describe the operation of a node in the C language. The **Analysis VI toolkit** contains VIs for signal processing, linear algebra, and statistics [6]. The toolkit supports signal generation, frequency-domain filtering (based on the FFT), windowing, and statistical signal processing, in one dimension. Other signal processing and image processing toolkits are available. The **Signal Processing Suite** enables time-frequency and wavelet analysis. The **IMAQ** image processing toolkit contains formats for analog video standards like PAL and NTSC. **IMAQ** also provides 2-D frequency-domain and morphological operations, but it does not provide other important functions such as the DCT.

**LabVIEW**'s image handling capabilities are limited compared to those of **MATLAB** and **IDL**. **LabVIEW** does not access raw binary image files. Images must be converted into standard
formats such as TIFF before **LabVIEW** can access them. Many operations, such as logarithms, are not compatible with image data directly. Image data must first be converted into an array structure, then the operation is performed, and finally the array is converted back to an image. 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 1-D data acquisition. **LabVIEW** VIs can be compiled and downloaded into embedded real-time data acquisition systems. Neither of these capabilities, however, are available for imaging systems.

**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** and **HiQ** are not available on UNIX platforms. The National Instruments Web site ([http://www.natinst.com](http://www.natinst.com)) contains freely distributable add-ons for **LabVIEW** and other National Instruments software packages. The **LabVIEW** newsgroup is comp.lang.labview.

### 2.4 Khoros

**Khoros** by Khoral Research, Inc., is another visual programming environment for modeling and simulation [8]. The block diagrams use a mixture of dataflow and control flow. **Khoros** supports 8-bit, 16-bit, and 32-bit integer and 32-bit and 64-bit float data types. **Khoros** is written in C, and supports calls to external C code. **Khoros** can also access external C++ code. A variety of toolboxes are available for **Khoros** that provide capabilities in I/O, data processing, data visualization, image processing, matrix operations, and software rendering.

**Khoros** libraries are effectively linked to the graphical coding workspace through a flow control tool called **Cantata**. When **Cantata** 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 2-D FFT. Glyphs can be written in C using Khoros templates. 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 clustering (data classification), and convolution. Datamanip requires that the Bootstrap and Dataserv toolboxes also be loaded. The Image Toolbox provides median filtering, 2-D 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, and the Geometry Toolbox is required to provide 2-D and 3-D plotting capabilities. The Khoros Pro Software Development Kit comes bundled with most of the toolboxes relevant to signal and image processing.

The Xprism Pro package runs independently of Khoros, but is meant to complement the Khoros product. 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 Khoros 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.
**Khoros** allows extensive integration with **MATLAB** through its *Mfile Toolbox*, making **MATLAB** functions and programs available to **Khoros**. This toolbox is available on most, but not all, of the platforms on which **Khoros** is supported. The **MATLAB** programs can be treated as source code inside **Khoros** objects. This toolbox includes the MATCOM compiler for converting **MATLAB** code to C++ code. It is based on Matrix, a C++ library consisting of over 500 mathematical functions. This in turn significantly increases the execution speed of interpreted **MATLAB** code. It also supports type single and double-precision float calculations, but not all **MATLAB** functionality is supported.

**Khoros** runs on Windows and Unix (DEC Ulrix, Linux, SGI, and Sun Solaris) operating systems. The Khoral Research Web site ([http://www.khoral.com](http://www.khoral.com)) contains freely distributable add-ons for **Khoros** and other Khoral Research software packages. The **Khoros** newsgroup is comp.soft-sys.khoros.

### 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 several free libraries ([http://developer.intel.com/vtune/perflibst/](http://developer.intel.com/vtune/perflibst/)) of 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 95, 98, or NT) and C/C++
compilers (Intel, Microsoft, or Borland). 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 2-D 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 [9]. 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 [10].

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.

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 and C languages for a wide variety of mathematical functions. As long as users have a Fortran or C 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 1-D and 2-D 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 http://www.jpeg.org/. Links to the C source code for the JPEG encoding and decoding algorithms 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 for 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 handle data sets of 1 Gb and larger. 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 2-D and 3-D 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 2-D 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 2-D 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 statistically 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: PCI and Envi. Other popular packages include ERDAS, ERMapper, PV-Wave, Raytheon, and ESRI.

5.1 PCI

PCI software by the PCI company is a geomatics software package. It supports imagery, Geographical Information Systems (GIS) data, and many ortho-projection tools [11]. PCI 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.

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 PCI and other software packages in either ASCII or binary format. PCI provides a limited native language so that some user-defined operations can be performed without having to leave the PCI environment. PCI 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, PCI also
supports vector data. PCI 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.

PCI can display images in a specialized 3-D perspective view, in which the gray levels of a particular channel correspond to heights. This format is useful for displaying topographic data. PCI also supports “fly throughs” in this perspective, allowing the user to scan over the data from different vantage points. PCI has a complete set of co-registering and mosaicking functions, and standard image filtering and FFT routines. PCI also includes its own drivers for accessing magnetic tape drives for reading data. Some applications for which PCI is well suited include watershed hydrological analysis, flight simulation, land cover classification.

PCI is available on Windows, Macintosh, and Unix operating systems including DEC Ultrix, HP-UX, SGI, and Sun Solaris. The Web site (http://www.pci.on.ca/) contains demonstration and image-handling freeware, as well as a subscriber discussion list discuss-request@pci.on.ca.

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 PCI. PCI has more classification capability and more options for ortho-projection and hydrological analysis of the data. Envi has more user-friendly access to its functions and more up-to-date formatting for some sensors. 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 (ftp://www.cis.upenn.edu/pub/xv/) is shareware. ImageMagick by John Cristy at E.I. du Pont de Nemours and Company Inc. (http://www.wizards.dupont.com/cristy/) is freely distributable. ImageMagick can also compose images, and create and animate video sequences. Both tools run on Windows NT and Unix operating systems under X windows.

Symbolic mathematics environments are useful for deriving algebraic relationships and computing transformations algebraically (such as Fourier, Laplace, and \( z \) transforms). These environments include Mathematica [12] from Wolfram Research Inc. (http://www.wolfram.com) and Maple [13] from Waterloo Maple Software (http://www.maplesoft.com). Mathematica has the following application packs related to signal and image processing: Signals and Systems, Wavelet, Global Optimization, Time Series, and Dynamic Visualizer. Commercial application packs are not available for Maple. A variety of notebooks on engineering and scientific applications are available for download from the Web site, 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.

System-level design tools, such as SPW by Cadence (http://www.cadence.com), COSSAP by Synopsys (http://www.synopsys.com), DFL by Mentor graphics (http://www.mentor.com), ADS by HP EEsof (http://www.tmo.hp.com/tmo/hpeesof/), and Ptolemy by the University of California at Berkeley (http://ptolemy.eecs.berkeley.edu), are excellent at simulating and synthesizing 1-D 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, 2-D FIR filtering, 2-D FFTs, 2-D 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. These system-level design tools, however, currently have limited but evolving support for synthesizing image and video processing systems into hardware and/or software.

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 small tasks. Electronic design automation tools for image and video processing systems are evolving.
References

file_id = fopen('mandrill', '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', 'tiff')
I2 = fft2(I1);
I2 = abs(I2);
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', 'tiff')

Figure 1. MATLAB example: (a) image, (b) FFT, and (c) code to display images, compute the FFT, and write out the TIFF images.
Figure 2. **IDL** example: (a) image, (b) FFT, and (c) code to display images, compute the FFT, and write out the TIFF images.
Figure 3. **LabVIEW** example: (a) image, (b) FFT, and (c) code to display images, compute the FFT, and write out the TIFF images.
Figure 4. Khoros example. (a) image, (b) FFT, and (c) code to display images, compute the FFT, and write out the TIFF images.