The HIPS-2 Software for Image Processing: Goals and Directions

Michael S. Landy

SharpImage Software, P.O. Box 373, Prince St. Sta., NY, NY 10012-0007
Department of Psychology and Center for Neural Science, NYU, NY, NY 10003

ABSTRACT

HIPS-2 is a set of image processing modules which provides a powerful suite of tools for those interested in research, system development and teaching. In this paper we first review the design considerations used to develop *HIPS-2* from its predecessor (*HIPS*). Then, a number of further developments are outlined which are under consideration. These include the development of a graphical user interface, more general approaches to color from the standpoint of color reproduction and linear models of color, and extensions to the software and data format to deal with production-oriented tasks.

1. THE HIPS SOFTWARE

HIPS[*] is an image processing package for the UNIX environment.[7-8] It was originally developed at New York University to support a research project on low bandwidth coding of images of American Sign Language[14], and was completed in 1983. In 1991, the software was completely rewritten in a design which involved a collaboration between SharpImage Software in New York and The Turing Institute in Glasgow, resulting in the *HIPS-2* software. The HIPS and HIPS-2 packages are now in use at over 200 sites worldwide in a wide variety of application areas including research in computer vision and robotics, visual perception, satellite imaging, medical imaging, mechanical engineering, oil exploration, etc. In this section the design of HIPS is reviewed briefly. In later sections the new features of HIPS-2 as well as possible future directions will be discussed.

The design and implementation of HIPS was started some ten years ago when the most common UNIX environment consisted of a mid-size computer (such as a DEC Vax) and terminals. Thus, HIPS was written based on the old model of UNIX software. It consists of a large set of commands (currently numbering over 200) which perform different basic image processing tasks (enlarge, rotate, Fourier transform, edge enhance, etc.). These are used within a Unix shell session, taking advantage of the UNIX *pipe* facility (denoted in a command line with a '|' character). For example, the following command

```
neg < image | enlarge | fourtr | mulseq filter | inv.fourtr | framevar
```

would be used to compute a negative of the data in file 'image', enlarge the image, transform to the Fourier domain, multiply the transform by the transfer function in file 'filter', back-transform to the image domain, and compute statistics on the resulting filtered image. The pipe operator automatically treats the output of each command as the input of the subsequent command. Each command may be controlled by various switches (to specify amount of rotation, enlargement, etc.), and more elaborate image processing is carried out using sequences of commands with intermediate results stored in temporary files. Thus, an experienced user will use the commands in an interactive fashion while developing an image processing

[*] *HIPS* stands for the *H*uman Information Processing Laboratory's *I*mage *P*rocessing *S*ystem, as that was the laboratory at New York University in which it was originally developed.

0-8194-1200-7/93/$4.00

algorithm, but then develop *shell scripts* (macro files of shell commands) for more production-oriented tasks.

The pipeline concept remains a flexible one because of the image data structure that HIPS uses. Each image or image sequence is preceded by an *image header*. In HIPS, this is ASCII text in a relatively fixed format which specifies the size of the image (number of rows, columns and frames) and the format of individual pixels (bits, bytes, short integers, integers, floats, complex, etc.) as well as various documentary fields. (Note that HIPS works with multiple image sequences as well as single images, where it is up to the user to interpret the meaning of the sequence: animation in time, stereo pair, spectral bands, etc.) Thus, each command in the above pipeline first reads the header, and then allocates image buffers sufficient to handle the size of input image. If a command changes anything about the image (its size or pixel format), it changes the header accordingly. In addition, the header structure contains a *sequence history*, which is simply a list of all the HIPS commands which have been applied to the sequence. Each command automatically updates this history, so that the user may examine a sequence header to find out what commands gave rise to it.

The structure of HIPS which supported these facilities was quite simple. It consisted of a small low-level library and a large collection of commands. The low-level library handled the details in common between different programs: memory management, image header I/O, image sequence history update, error handling, and little else. Each HIPS program was an independent entity, but the form of programs was similar: read the header, allocate buffers, then for each image in the sequence: read, process and write the image.

A principal design goal of HIPS since it became a commercial product (in mid-1983) was to keep it standardized. In particular, the advantage of UNIX software is the ability to use the same set of sources on a wide variety of hardware from different manufacturers. As such, HIPS is written in standard Kernighan and Ritchie[6] C with as few system calls as possible, and using only those system calls (and include files) which are likely to be standard across different versions of Unix. Thus, we have not upgraded to ANSI C (function prototypes and the rest) as this would limit the machine-independence. The HIPS software has been used on nearly every UNIX hardware platform at this point.

The only hardware-dependent part of HIPS is the device support for scanning or digitizing images and for displaying single images or image sequence animations. Here, device independence is not entirely possible, but device support has been developed for a variety of framebuffers and window systems (such as SunView, X, XView, NextStep, etc.).

2. FROM HIPS TO HIPS-2

There have been several developments since HIPS was first implemented. First, we were involved in a project to develop software for the simulation of models of visual processing. This led to the EVE software[9-10] (for *E*arly *V*ision *E*mulation) which was based heavily on the HIPS design, although its low-level library was enhanced to make typical programs easier to write, an enhancement which was further developed in HIPS-2. But, the primary event was the development of HIPS-2 itself.

HIPS-2 grew out of a number of discussions (between SharpImage Software and The Turing Institute) about the deficiencies we perceived in HIPS. The obvious deficiency (discussed further in Section 3) is the lack of an easy to use, graphical interface to HIPS' capabilities. But, as a prelude to the development of such a capability, we saw it as necessary to first provide access to HIPS' capabilities for the programmer. In other words, HIPS always had a command-line interface, but the code did not provide a set of image processing subroutines. This was obviously a glaring omission in the original design. Since the provision of a subroutine interface would involve a substantial amount of reprogramming, some time and care was taken to specify what other aspects of HIPS could be improved with the rewrite. This led to a specification for HIPS-2 which was programmed in early 1991. Here, we review a number of the more

important capabilities which appeared in HIPS-2. In the design and implementation we were careful to keep the software reasonably upward-compatible (although the command interface did change slightly), and to continue to maintain the strict machine-independent approach to the code.

2.1 Subroutine interface

The primary motivation for the move to HIPS-2 was to provide a subroutine interface to the HIPS image processing capabilities. However, it was clear that such an interface would be of use to a variety of applications with different needs, including uses from programs which otherwise did not make use of HIPS itself (e.g. did not read images in HIPS format, did not manipulate image headers, etc.). Therefore, the subroutine interface operates at several levels of abstraction. At the lowest level no reference to HIPS-specific data is required; a routine is provided with image sizes and pointers to buffers as parameters. At higher levels of abstraction, a routine is provided instead with a memory-resident image header structure. This structure is similar in content to the image header structure used for HIPS image files with the addition of a pointer to the memory-resident image data. The header structure includes all of the other details required by the routine including image geometry and format.

2.2 Automatic format conversion

The reprogramming of the low-level image header library also involved the provision of routines for image I/O as well as header I/O. This allowed us to easily provide a new capability to the command-level HIPS-2 interface which is a kind of minimal object-oriented facility. The HIPS command-level interface was often frustrating to the user for the following reason. HIPS manages a number of different image formats (the raster formats such as byte, integer, float, complex, and so on, as well as other nonraster formats). Some HIPS programs will change the image format (e.g. a Fourier transform of a floating point image results in a complex image). Most HIPS programs are only equipped to handle a subset of image formats. Thus, with HIPS it was up to the user to keep track of what format image was output by one program and to convert it to a form required by the next program if necessary (by inserting a program such as *ftoc* in a pipeline to convert from *f*loat *to* *c*omplex). In HIPS-2, this is no longer necessary. Each program has coded in it the list of image formats that it knows how to deal with directly. Header and image I/O routines are provided which read images and, if necessary, convert them automatically to the nearest available type handled by a given program. The conversion routines do their best to preserve information whenever possible (for example, preference would be given to a conversion from short integer to long integer over converting to byte), and notify the user in any case that such a conversion is taking place. This minor convenience has made HIPS-2 far more pleasant to use.

2.3 Standardized command-level interface

With HIPS-2 a routine is provided for the standardized parsing of command arguments. This has made it possible to simplify the command-level interface to HIPS-2 and make it more consistent across programs. In addition, this has made it possible to provide a standard set of switches across the entire set of HIPS-2 programs. These include a standard way of requesting a short usage message (quicker and far more abbreviated than the online manual pages) and specifying how binary (1 bit per pixel) images are to be unpacked, how real images are to be converted to complex and how complex images are to be converted to real. As a command-line system, HIPS-2 still requires far more learning than other systems (menu-driven or graphical interfaces), but this consistency across programs lessens the load on the novice user a bit.

2.4 Data structure

The header structure (both in stored image files and in its memory-resident version) has been enhanced in HIPS-2 to provide several new capabilities. The most useful of these is the ability to extend the header by adding parameters. Thus, the header structure becomes a database tool in itself, able to flexibly store both numerical and ASCII data. The user may add new named fields to the header which store single numbers, arrays, or strings. This can be done using HIPS commands or, of course, by subroutine call in the context of a larger task.

These extended parameters have been used along with other HIPS-2 header extensions as the basis for a number of new capabilities. First, HIPS-2 has several added capabilities for dealing with color imaging. The header has a new field for the number of colors in a sequence, thus allowing for time sequences of RGB images. Pseudocolor is supported using the extended parameters by storing a lookup-table in the image header. Image histograms and image pyramids are also supported by additional header data saved as extended parameters. Additional header fields also enable the specification of a user-defined rectangular region-of-interest. A recent addition allows for sequences with mixed image types (such as an image sequence consisting of a byte image, a float image, and an integer image) by storing the array of formats in the header as an extended parameter. The extended parameters have made it easy to extend HIPS-2 in major ways while retaining upward compatibility.

2.5 General functionality

HIPS-2 includes a number of minor changes as well. The low-level library takes advantage of the kinds of enhancements first used in EVE which are all geared toward providing subroutine support for the most common header and image manipulation and I/O activities. The library was designed to be upward-compatible with HIPS, and in particular HIPS-2 routines can read images previously created by HIPS routines. HIPS-2 includes and extends the set of capabilities which were available with HIPS. It includes programs for:

- Header Manipulation
- Image Format Conversion
- Hardcopy and Halftoning
- Operations on Image Sequences
- Image Generation
- Geometric Manipulation of Images
- Single Pixel Transformations
- Image Statistics
- Image Arithmetic
- Convolution, Correlation, Edge Detection and Other Spatial Operations
- Digital Image Transforms and Filters
- Gaussian and Laplacian Pyramid Operations
- Feature Extraction and Rule Generation
- Image Compression
- 3D Vector Graphics
- Color and Colormap Support
- Device and Window System Tools

3. PROSPECTS FOR A HIPS-2 GRAPHICAL USER INTERFACE

As has been discussed, HIPS-2 may be used both as a programming library and as an interactive tool. The user-interface is of the old Unix style: a large collection of programs that are used by typing command lines to a Unix shell, and combined using the Unix pipe facility. This often remains the most effective way to use the software, but it has become increasingly clear that a friendly interface is needed which has windows and menu-access to HIPS-2's capabilities. In designing HIPS, few other image processing packages existed for the UNIX environment, and our design decisions were our own. In considering a graphical user interface (or GUI) for HIPS now, we have several examples of image processing GUIs to use as examples to guide our design.

3.1 Forms of Image Processing GUI

The most important consideration in designing a GUI is to consider the tasks the user will carry out most often so that the GUI will ease such tasks. There appear to be at least three different forms of image processing GUIs at the present time: database, dataflow, and interactive. Some packages treat image database manipulations as the prime consideration and image processing on the images in the database is a subsidiary consideration. HIPS was never designed with image database management in mind, and so this form of GUI will not be considered further.

A number of packages (such as Khoros) include GUI support for composing a dataflow representation of an image processing algorithm. The mouse and menus are used to create and manipulate a diagram representing the flow of data in the algorithm. Typically, icons are used to represent single image computations such as enlarge, rotate, Fourier transform. In other words, icons represent operations that would generally be carried out by a single HIPS (or Khoros) program. Icons are connected using directed arrows to represent the flow of data (either images or single parameters). These diagrams can certainly be used to represent the pipeline commands of HIPS (with a linear, boxcar arrangement of the icons). But, they are more general than that as they encompass diagrams with parallel dataflows and/or feedback (which can occur commonly in image processing applications including motion analysis, motion-predictive coding, and so on), all of which require multiple commands and temporary files in the HIPS command-level package. Such a dataflow representation is more flexible for easily representing complex image processing algorithms, and the dataflow diagram is a useful way to document an algorithm. Once the diagram has been set up, it documents a general algorithm and hence can be *compiled* to provide a convenient way to apply a set of commands to more than one image. At the same time, the manipulation of these diagrams is time consuming, and users may find the diagrams an impediment when only a few operations are required, leading one to revert to shell command usage or to a preference for a more interactive GUI.

The third, and perhaps most common type of image processing GUI is one geared to interactive manipulation of images. In such a system, one or more images are displayed at a time. The user brings the focus to one of the images, chooses an operation to perform on that image (using buttons, menus, and optionally filling in parameters which control that operation), and the result then appears. Such an interface is better suited to more interactive use of image processing capabilities as it allows the user to quickly choose an operation and view the results.

In the area of interactive image processing GUIs, we are impressed with the object-oriented stack approach used in the Obvius image processing package.* In Obvius, the user may manipulate one or more image *stacks*. Each stack is simply a stack of image objects. The software is object-oriented in that each

*Obvius is a noncommercial package written in Common Lisp and C. For information on the package contact its authors: David Heeger (heeger@white.stanford.edu) and Eero Simoncelli (eero@whitechapel.media.mit.edu).

image in the stack may be any number of different types of object: raster images in various formats, image pyramids, etc. The user specifies the input images to an operation by focusing on elements of the stacks, and the result is also placed on top of a stack. Thus, both the inputs and results from each operation are readily available to the user. The object orientation means that the user can easily manipulate a variety of image objects. This brings up issues of how different objects are to be displayed (scaled, formatted, etc.), but these are easily addressed by the GUI.

3.2 GUI implementation considerations

A second set of considerations in creating a GUI concerns the method of implementation. There are two principal means of adding a GUI to an image processing package. In one route which I term the layered approach, the addition of the GUI is almost independent of the image processing package. The GUI designer creates a generic GUI package which is then scripted. The script specifies what menus and panels are available at each level of the GUI and what they do. The specification of what a button or menu item does is such that the GUI program constructs a shell command. This command is then executed and the results stored and/or displayed. The layered approach makes it possible to add a GUI to a package which was entirely command-line oriented. The Fresco package produced by Celadon is one such layered GUI. The Thomson CSF laboratory has recently produced such a layered GUI (called ICE) which interfaces to HIPS-2 and is due for release in the coming months.

The alternative approach is to produce what I term a seamless GUI. This is interface code that is compiled together with the image processing code and for which the image processing code forms an integral part. This requires, of course, a subroutine interface to the image processing code. It is likely to be more efficient as it eliminates the overhead of running shell commands. At the same time, unless runtime loading of image processing binaries is implemented, there is a cost in terms of a far larger executable. In addition, with the lack of scripting it becomes more difficult to tailor the interface to the task or to the user. A compromise between the two approaches is also possible: a seamless GUI which has the image processing code integral to the GUI, but for which the details of the GUI are scriptable.

Finally, a concern of utmost importance in the design of an image processing GUI is memory management. Images are generally quite large objects. A program which allows the user to quickly and interactively generate a variety of images must contend with their storage. To keep all of the currently active images (those currently available to the user as input to subsequent commands) in memory is most efficient, but may not be feasible. The system must therefore maintain a set of images in memory, in temporary disk files, and an image database and, at the same time, hide the less interesting details of this image management from the user. Once images are stored permanently on disk, the package should also maintain sufficient information to support the image database activities of the user as well as to document the operations that gave rise to each image.

3.3 Toward a HIPS GUI

As mentioned above, the HIPS-2 package has already become part of a layered image processing GUI. The GUI (ICE, by Thomson CSF) allows the user to construct dataflow diagrams, to apply those diagrams to images and image sequences, and to examine the results of those operations as still images or as image sequence animations.

However, the design of HIPS-2 was intended to make HIPS the basis of future seamless image processing GUIs. For this reason, HIPS-2 includes subroutine interface to virtually all of its capabilities. The most abstract versions of the subroutines are passed image headers and parameters, and thus treat images in an object-oriented fashion. As mentioned, both dataflow and interactive GUIs are useful, but for different purposes, and we foresee the creation of both forms of interface for HIPS-2. In both cases

the interface should be able to handle both images and image sequences in a variety of formats, and the HIPS-2 subroutine interface is designed to support this. Such an interface should take advantage of useful features found in previous packages such as the image stacks concept of Obvius.

4. COLOR MODELS FOR IMAGE PROCESSING SOFTWARE

In the past, digital image processing has been dominated by two forms of image representation: monochrome and RGB. Image datasets abound which are passed around from laboratory to laboratory in one or the other of these two formats. Each format consists of an array of pixel values each of which is either a single number (monochrome) or three numbers (representing the three values which are converted to voltages and sent to the red, green and blue guns of a color monitor). In the RGB case, these three *image planes* are often treated independently for image enhancement, noise reduction, and so on, and then displayed together on the user's color monitor. This is the model that has been used in HIPS as well.

However, this simple model is severely limited in the sorts of information that it can represent. It is true that under photopic conditions the color sensation of the human eye presents us with a three-dimensional color world[19] and the existence and design of color monitors (and the use of red, green and blue primaries) reflects various aspects of that sensory system. Nevertheless, this use of RGB as the primary color data hides several problems. First, images are displayed on a number of different display devices which likely do not use the same primaries (e.g. phosphors). Second, display devices (and some digitizing devices) are generally nonlinear. Third, images are not objects unto themselves but rather are derived from real-world scenes with their own chromatic content and relevant conditions (e.g. illumination). If one is given an RGB image, the RGB values alone are neither sufficient to create on a monitor the same image (as perceived by a human observer) as that seen by the originator, nor are they sufficient to recreate the scene as viewed by the digitizing camera. Thus, it is important to consider a more elaborate data format which would allow the transfer of color *images*, meaning both raster data and sufficient side information to enable one to render those data so as to be indistinguishable from the original by a human observer (at least up to limits imposed by the sampling density of the image data).

Part of the problem, as mentioned above, is the nonlinear characteristic found in both display devices and some sensors. This can be handled by providing either a lookup-table or parameters of a modeled nonlinearity (such as the so-called *gamma curve*: $out = A\ in^{\gamma}$, where *in* is the input to the monitor such as the pixel value, and *out* is the monitor output in physical units of luminance). Thus, if one is given a set of RGB values as well as gamma curve parameters for each of the three components (and assumes *phosphor independence*[1]), one can determine the actual phosphor intensity at each location.

For the full color rendering problem where one wants to render an image that was originally rendered on another monitor, more information is needed. For an image to be rendered correctly, not only is knowledge of the nonlinearity of the original display device required, but one also needs to know the color primaries of that device. This can be either their full spectral power distribution curves, or at least the 3-dimensional perceptual equivalents in some agreed-upon coordinate system such as CIE XYZ. With these data (as well as the modeled nonlinearity of both the originator's and current display device) one is in a position to render the image correctly (subject to possible mismatches in the color gamuts of the two display devices).

If our interest is deeper, and one wants to analyze the scene which gave rise to the image, the problem becomes more difficult. Consider the problem of modeling the scene itself so as to estimate properties of the scene including surface color (i.e. reflectance) and illumination. For this task, the data required to render the color percept at the originator's monitor may well be insufficient (information may have been lost along the way from scene to monitor). One solution is to scan the scene for spectral energy at a series of closely spaced wavelengths. This results in a series of images, one for each spectral band, much as one sees in satellite imaging applications. Any resulting multispectral image may be converted to a 3-

dimensional color space by use of estimates of the cone spectral sensitivities[13] or color matching functions.[19] The multispectral representation will certainly provide more flexibility, but at the expense of a far larger dataset. For example, a 5 nm spacing between samples across the visible spectrum (400-700 nm) results in 61 numbers representing each pixel. This is clearly an unwieldy representation. And, if the interest is in the human perceiver's response to those data, it is likely to be more data than is necessary for the task.

A compromise is needed, as has already been noted in the computer graphics community.[5] The solution is to provide a model for the kinds of spectral power distributions and spectral reflectance functions that are seen in common illuminants and surfaces. Recently, a number of researchers have suggested that both illuminants and reflectances may be approximated quite well using linear models with low-dimensionality and that these models can then be used to handle problems of color rendering and analysis of surfaces and illumination.[2-4,11-12,15-18] In other words, for the illuminants one is provided with a fixed set of illuminant *basis functions*, and any given illuminant is modeled as a linear combination of those basis elements. A similar logic is applied to the surface spectral reflectance functions. The surface basis functions may be chosen in order to best fit measured spectral distributions[11] or so as to best approximate the observer's response to those surfaces,[12] and similar considerations apply to the choice of illuminant basis set.

Thus, it would be useful for an image processing package such as HIPS to support a variety of models of image formation and display. For display, an image would optionally be accompanied by models of the display phosphor nonlinearities and/or phosphor chromaticities. To model the content of a scanned image, the image would also include a model of the nonlinearities and spectral sensitivities of the device which gave rise to the image. For applications involving modeled scenes, the image data representation should support an image representation where each pixel consists of a vector of values which relate to a set of basis functions which are also stored with the image (or separately). Finally, the software should support standard colorimetric calculations such as conversions between RGB measurements for a particular set of phosphors and other color representations such as cone excitations, XYZ, and so on. All of these capabilities exist in the public domain package CAP.[2-3] HIPS-2 has a number of capabilities which could be used to support these kinds of calculations including the ability to store several color planes and to store side information in the image header. We propose adding sufficient additional software to HIPS-2 to support this range of color calculations.

5. ODDS AND ENDS: DATA FORMAT AND PRODUCTION TASKS

A number of developments in HIPS-2 have stemmed from feedback from our users, and in this section we discuss several suggestions which originated at the Lawrence Berkeley Laboratories. The issue is how to use the software in a stream-oriented production environment. For example, suppose we have a device which is digitizing images at a moderate rate, and a user is controlling when to start using those images, and when to complete the sequence. These digitized sequences are then fed through several image processing calculations, and the end result is stored in a file. In order to keep up with these demands, the calculations are performed on several different machines (and intermediate results passed on via shared memory or via a network).

This scenario raises several issues. First, the individual operations (corresponding to single HIPS-2 programs) do not know the number of frames in the sequence in advance, and thus the image header must be able to indicate 'process until end-of-file' in addition to the current indication of the total number of frames. Second, in such a scenario additional documentary data would likely be collected on each frame (such as a time stamp). These data are gathered after the image header has already been passed on to the next calculation in the sequence, necessitating the addition of an extended parameters section to the image format which is stored for each image frame, in addition to the parameters which are stored once per

sequence in the image header (much as is available in the TIFF standard, although we find the ASCII, hence readable form of the HIPS header to be advantageous). Finally, a capability is required to 'unstream' an image file by storing the correct number of frames in the image header. An alternative would be to store the frames in separate HIPS files and treat the image sequence as a database with a master file which indexes the individual frame files.

6. DISCUSSION

In this paper the HIPS-2 software has been introduced. When HIPS was first introduced, it was one of the most flexible packages available for image processing under the UNIX environment. In the ten years that have passed since that time, the HIPS package has matured, but it has become clear that to meet with the variety of uses to which it is put, a variety of image processing GUIs will be required.

7. ACKNOWLEDGMENTS

HIPS-2 is a commercial product. For more information on *HIPS-2* contact Michael Landy, SharpImage Software, P.O. Box 373, Prince Street Station, New York, NY 10012-0007; Voice: (212) 998-7857. In his more academic pursuits, Michael Landy is supported by NIH grant EY08266 and an NRC-NASA Ames Research Associateship. We thank Mark Young, David Heeger, Joyce Farrell and Brian Wandell for their helpful comments.

8. REFERENCES

1. D.H. Brainard, "Calibration of a computer controlled color monitor," *Color Research and Application,"* Vol. 14, pp. 23-34, 1989.

2. D.H. Brainard and B.A. Wandell, "The color analysis package," *Stanford Applied Psychology Laboratory Tech. Report 98-3*, 1989.

3. D.H. Brainard and B.A. Wandell, "Calibrated processing of image color," *Color Research and Application*, Vol. 15, pp. 266-271, 1990.

4. M.H. Brill and G. West, "Contributions to the theory of invariance of color under the condition of varying illumination," *J. Math. Biology*, Vol. 11, pp. 337-350, 1981.

5. J.D. Foley, A. van Dam, S.K. Feiner and J.F. Hughes, *Computer Graphics Principles and Practice* (2nd Ed.), Addison-Wesley, New York, 1990.

6. B.W. Kernighan and D.M. Ritchie, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, NJ, 1978.

7. M.S. Landy, Y. Cohen and G. Sperling, "HIPS: image processing under UNIX. Software and applications," *Behavior Research Methods, Instruments, & Computers*, Vol. 16, pp. 199-216, 1984.

8. M.S. Landy, Y. Cohen and G. Sperling, "HIPS: A Unix-based image processing system," *Computer Vision, Graphics, and Image Processing*, Vol. 25, pp. 331-347, 1984.

9. M.S. Landy, L.Z. Manovich and G.D. Stetten, "All about EVE: The Early Vision Emulation software," *Behavior Research Methods, Instruments, & Computers*, Vol. 21, pp. 491-501, 1989.

10. M.S. Landy, L.Z. Manovich and G.D. Stetten, "Applications of the EVE software for visual modeling," *Vision Research*, Vol. 30, pp. 329-338, 1990.

11. L.T. Maloney, "Evaluation of linear models of surface spectral reflectance with small numbers of parameters," *Journal of the Optical Society of America A*, Vol. 3, pp. 1673-1683, 1986.

12. D.H. Marimont and B.A. Wandell, "Linear models of surface and illuminant spectra," *Journal of the Optical Society of America A*, Vol. 9, pp. 1905-1913, 1992.

13. V. Smith and J. Pokorny, "Spectral sensitivity of the foveal cone photopigments between 400 and 500 nm," *Vision Research*, Vol. 15, pp. 161-171, 1975.

14. G. Sperling, M.S. Landy, Y. Cohen and M. Pavel, "Intelligible encoding of ASL image sequences at extremely low information rates," *Computer Vision, Graphics, and Image Processing*, Vol. 31, pp. 335-391, 1985. Also published in: A. Rosenfeld (Ed.), *Human and Machine Vision II*, Vol. 13 in W. Rheinboldt and D. Siewiorek (Eds.), *Perspectives in Computing*, pp. 256-312, Academic press, New York, 1986.

15. H.J. Trussell, "Applications of set theoretic methods to color systems," *Color Research and Applications*, Vol. 16, pp. 31-41, 1991.

16. B.A. Wandell, "Color rendering of camera data," *Color Research and Applications, Supplement*, Vol. 11, pp. S30-S33, 1986

17. B.A. Wandell, "The synthesis and analysis of color images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-9, pp. 2-13, 1987.

18. B.A. Wandell and D.H. Brainard, "Towards cross-media color reproduction," *Proceedings OSA Applied Vision Topical Meeting*, Optical Society of America, Washington, DC, 1989.

19. G. Wyszecki and W.S. Stiles, *Color Science*, Wiley, New York, 1982.