Behavior Research Methods, Instruments, & Computers
1984, 16 (2), 199-216

HIPS: Image processing under UNIX. Software and applications

MICHAEL S. LANDY, YOAV COHEN, and GEORGE SPERLING
New York University, New York, New York

HIPS (*H*uman Information Processing Laboratory's *I*mage *P*rocessing *S*ystem) is a software system for image processing that runs under the UNIX operating system. HIPS is modular and flexible: it provides automatic documentation of its actions, and is relatively independent of special equipment. It has proved its usefulness in the study of the perception of American Sign Language (ASL). Here, we demonstrate some of its applications in the study of vision, and as a tool in general signal processing. Ten examples of HIPS-generated stimuli and—in some cases—analyses are provided, including the spatial filtering analysis of two types of visual illusions; the study of frequency channels with sine-wave gratings and band-limited noise; 3-dimensional perceptual reconstruction from 2-dimensional images in the kinetic depth effect; the perception of depth in random dot stereograms and cinematograms; and the perceptual segregation of objects induced by differential dot motion. Finally, examples of noise-masked, cartoon coded, and hierarchically encoded ASL images are provided.

The rapid decline in the cost of computer hardware makes it possible for psychologists to engage in image processing. In this paper, we first describe a powerful, yet flexible, software system, HIPS (*H*uman Information Processing Laboratory's *I*mage *P*rocessing *S*ystem). that is relatively independent of special equipment. Second, we present potential applications of image processing in the laboratory and demonstrate them using the language of our software system.

THE ENVIRONMENT

Our research involves a study of the perception of American Sign Language (ASL), which is a manual form of communication used primarily by the hearing impaired. The aim is to find images that are intelligible to "speakers" of ASL and yet that can be encoded using minimal channel bandwidth (Sperling, 1980, 1981; Sperling, Pavel, Cohen, Landy, & Schwartz, 1983). For this research, we have set up a computing environment that is capable of reading and storing sequences of video images, transforming the images in the spatial and temporal domains, and, finally, presenting the results to readers of ASL in order to assess the legibility of the images.

The work on image processing of American Sign Language was supported by National Science Foundation, Science and Technology to Aid the Handicapped, Grant PFR-80171189. The preparation of this article was supported by the NSF (above) and by USAF Grant AFOSR-80-0279 (for Yoav Cohen and George Sperling). We wish to acknowledge the many contributions of M. Pavel and the assistance of Thomas Riedl and Robert Picardi. Y. Cohen's mailing address is: The National Institute for Educational Testing and Evaluation, Jerusalem, Israel. M. S. Landy and G. Sperling's is: Human Information Processing Laboratory, Department of Psychology, New York University, New York, NY 10003.

The Hardware

A wide variety of special-purpose image-processing peripherals have recently become available. Some of these devices are capable of performing many complex image-processing tasks without drawing on the capacity of a main, general-purpose computer, but their prices are forbidding for the traditionally low-budget psychology laboratory. Therefore, we decided to sacrifice speed of processing, and to emulate in software all image transformations. Rather early, however, we realized that we gained a lot in terms of flexibility and ease of development of image-processing tools.

The present hardware configuration consists of a VAX 11/750 computer to which a special-purpose image processor (a Grinnell GMR 27-30) is attached. In addition, the system has several terminals and printers, and a general-purpose parallel interface (a DR11-C). The Grinnell system is capable of converting between video and digital representation of images. As will become clear, HIPS does not depend on this particular hardware configuration. Much more central to the design of HIPS is the UNIX operating system, under which the image-processing software was developed.

The UNIX Operating System

The design of the image-processing software was influenced by the special features of UNIX (Ritchie & Thompson, 1978). Therefore, we will discuss them briefly. For a fuller, yet nontechnical, description of how UNIX appears to the user, the reader is referred to Kernighan and Mashey (1981).

A program in UNIX is executed by typing its name. Thus, the system program for listing files, *ls*, is invoked simply by typing

```
ls
```

This action outputs a list of files that reside in the user's current file directory.

There is no difference between system and user's programs in this respect. From the user's point of view, any program that resides in the command "search set" can be invoked by typing its name, be it a user or a system program.

Most UNIX system programs can be invoked with a list of arguments. Thus, the line

```
ls dir1 dir2 dir3
```

is a request to output the lists of entries in the three named subdirectories. The list of subdirectory names is an argument list that is parsed by the program *ls*. The same facility is offered to the applications programmer, who can easily write programs that access and parse their arguments.

Every program in UNIX has associated with it three files: the standard input, standard output, and standard diagnostic. By default, the input and output are directed from and to the user's terminal, but the flexibility of program execution can be enhanced by using I/O redirection. For example,

```
ls > dirlist
```

demonstrates output redirection. By using the ">" symbol, the output of *ls* is redirected to the file *dirlist*, instead of being sent to the terminal. As an example of input redirection, consider the program *wc*, which counts the number of characters, words, and lines in a text file. The command

```
wc < dirlist
```

performs the count on the file *dirlist*, which was created in the previous example. Note that the output of *wc* is sent to the terminal, since no output redirection was specified.

One of the most convenient features of UNIX is the possibility of sending the output of one program directly to a second program. For example, instead of the two commands

```
ls > dirlist
wc < dirlist
```

the user can type

```
ls | wc
```

This has precisely the same ultimate effect, but it allows more processes to be run simultaneously and saves mass storage; in the above case, the temporary file *dirlist* is rendered unnecessary. In most cases, this saves execution time and disk access (Stevens & Hunt, 1982). In this facility, which is called a *pipe* and is designated by the "|" symbol, there is more than mere convenience: It greatly affects the way in which programs in UNIX are written. Its main advantage is that the user can write a collection of relatively small programs, each carrying out a simple operation. These are then recombined using the pipe operator in order to perform a wide range of more complex tasks. A single user command can consist of a single program or a sequence of any number of programs combined with pipes, where each program in the sequence reads its input from the previous program and outputs to the following program in the sequence. Such a sequence of programs combined with pipes in one command is called a *pipeline*.

UNIX provides a rich command language (or job control language)—the *shell* (Bourne, 1978; Thompson, 1975). When combined with the *pipe* facility, it provides a powerful tool for integrating lower level programs into higher level "super programs." All of the above-mentioned features make UNIX an attractive programming environment.

THE SOFTWARE SYSTEM

In developing the software system for image processing (HIPS), we adopted a modular design philosophy akin to that of UNIX. We tried to build a collection of simple tools that users can recombine according to their needs.

Modularity, simplicity, and flexibility are major objectives in the image-processing software. Our working environment is one of research, rather than of production; therefore, considerations of program efficiency are secondary to users' productivity. Nevertheless, the flexibility of the UNIX command language (the shell) allows for a very convenient production environment as well. In addition, as UNIX is seen by its designers, we view HIPS as a system in a state of continuing evolution. New tools are programmed as needed, and old programs are discarded when they prove to be cumbersome or superfluous.

Very early in the course of developing HIPS, it became apparent that it might be applied in a wider range of domains than originally intended. Therefore, we expended extra effort to provide a flexible system that could then be extended according to future needs. The result is a system that is a collection of a few general functions and many image-processing tools, all unified by the *sequence-header* concept. This and related concepts are described in the following sections, but for a more technical description and for a fuller descrip-

tion of the available programs, the reader is referred to Landy, Cohen, and Sperling (1984). After more than a year of using HIPS, our choice of design features more than fulfilled our expectations.

The Sequence Header

In HIPS, the user performs operations on single images or sequences of images. An image sequence is an ordered sequence of single images in the same format. For example, a sequence depicting moving objects taken from a video or film source consists of a number of discrete frames and is represented in HIPS as an image sequence.

Each image or sequence of images is preceded by a collection of items that describe the way in which the pictorial material is represented (its format), and the parameters that pertain to this format. As an example, consider the representation by picture elements (pixels). In this format, an image is considered to be an array of points, and the gray level (brightness) of each point is represented by a number (this is usually the output format of image digitizers). In order to apply a transformation to a sequence of images in this format, one has to know the number of bits used to represent each pixel, the dimensions of the array that represents a single image, and how many images are contained in the sequence.

In addition to parameters, the header contains descriptive entries for the purpose of documentation. There are entries for the origin of the sequence, a verbal description of the sequence, and so on. More important, there is an entry that documents the history of the sequence. When a sequence is processed by a program, some of the entries in the header are modified. Thus, if the sequence *inseq* is reduced in size by a factor of 2 by the command

```
reduce 2 < inseq > outseq
```

the numbers of rows and columns are halved, and this change is reflected in the header of *outseq*. In addition, one line is added to the *history* section of the header. This line documents the fact that the program *reduce* was applied and records the date on which this was done. Figure 1 is an example of a header as it would be displayed on the user's terminal by the command

```
rframe | adddesc -d "9/5/82" -o "M. Landy" \
    -s "Face 1" -a "This is the first of the ASL sequences" | \
    reduce | tee outseq | seeheader.
```

This command reads a single digitized frame from the Grinnell interface into memory. The HIPS program *adddesc* adds some descriptive information to the header, including the date the image was digitized (*-d*), the originator of the image (*-o*), the sequence name (*-s*), and general descriptive text (*-a*). The

```
rframe | adddesc -d "9/5/82" -o "M. Landy" \
       -s "Face 1" -a "This is the first of the ASL sequences" | \
       reduce | tee outseq | seeheader
```

(a)

```
Original name          M. Landy
Sequence name          Face 1
Number of frames       1
Original date          9/5/82
Number of rows         256
Number of columns      256
Bits per pixel         8
Bit packing            No
Pixel format           Bytes

Sequence history

rframe "-D Mon Oct 31 10:41:56 1983" | \
reduce "-D Mon Oct 31 10:41:57 1983"

Sequence Description

This is the first of the ASL sequences
```

(b)

Figure 1. Output of the *Seeheader* program: (a) a command; (b) the output on the terminal that results from running the command in (a).

command then reduces the image size by a factor of two (the default). We use the UNIX program *tee* (a pipe-fitting juncture) to save a copy of the sequence in a file, and concurrently to examine the header with *seeheader*. Note that the "\" character serves as an indicator to the operating system that the command continues onto the following line.

The history section serves both as a mechanism for automatic documentation and as an actual script for repeating the same set of operations on a sequence. Each line in the history section begins with a program name followed by a list of arguments. The last argument in each line is a dummy argument, recording the time of execution. The whole history section, unchanged, can serve as an input to UNIX's command language (the shell), so that the same set of operations can be reapplied as one long UNIX pipe. The last entry on each line, the date argument, is ignored by all image-processing programs.

Image Formats

Pixel Format

HIPS is capable of manipulating a variety of image formats. We have already mentioned the "pixel format," in which an image is represented as an array of values. These values can be represented as single bytes, as longer integers, floating point, or complex numbers, according to the precision needed by the user and the particular algorithm that has to be applied. Thus, the pixel format is a general category that subsumes several related formats. Pixels can also be represented by single bits (binary images) or as numbers in ASCII code. This last format, although wasteful in terms of storage and time, is useful for transforming images into regular text files. As text files, sequences can then be processed

by other UNIX programs such as the statistical package of Perlman (1980, 1981).

3-D Format

Although it is the most common, pixel format is not always optimal. For this reason, HIPS offers other formats as well. Consider, for example, the case in which images are initially generated by the computer. Whereas digitized images are usually represented in pixel format, images or sequences that are generated by computer graphics systems are usually encoded as points, lines, and surfaces in a 2- or 3-dimensional space. Currently, HIPS supports images in "PLOT-3D" format, in which points and straight lines are represented by their coordinates in a 3-dimensional reference system. This representation, however, is not isolated from other forms of representation; plots can be converted to pixel format and are then available for processing by the whole gamut of tools in HIPS. We will see examples of the use of this format below as we examine a number of applications of HIPS to psychology.

Hierarchical Formats

Another way to represent images is by viewing an image as a hierarchical structure, in which the nodes on each level correspond to subareas of the node that is higher in the hierarchy; the top of the hierarchy (or the root of the tree) stands for the entire image. HIPS recognizes some of these formats, such as quad-trees (Pavlidis, 1982) and supports conversion between pixel formats and hierarchical formats. One of the advantages of using hierarchical codes is the reduction in the space needed for storage of images.

Spanning Trees; Run-Length Formats

Other formats for image compression supported by HIPS include spanning trees (in the sense of Reingold, Nievergelt, & Deo, 1977), for efficient storage of line drawings, and run-length codes (Rosenfeld & Kak, 1976), for efficient storage and transmission of images that contain large areas of uniform gray level.

New Formats

As the foregoing discussion suggests, the multiplicity of image formats, and the fact that the sequence header is independent of a particular format, pay handsomely in terms of flexibility and generality. In fact, new formats can be added to HIPS at will, the only restriction on the developer of new tools being that new format names should not conflict with prior definitions and that each new program should include a proper test for ensuring that the input is in the correct format for the particular algorithm.

General Structure of a Program

The pipe facility of UNIX, and the concept of a sequence header, together determine the general structure of an image-processing program. In most cases, programs read the pictorial material and the header from the standard input file and write the updated header and the transformed sequence onto the standard output file. The diagnostic file is reserved for error messages, warnings, and other messages that are not an integral part of the output sequence. Before reading the pictorial material, the program parses its arguments and reads the header. The reading of the header and other manipulations of the header are performed by a small set of functions that reside in a system library. If the parameters of the header match the requirements of the program, the header is updated and written onto the output file. Only then does the actual processing of the sequence begin.

Since HIPS was developed on a virtual-memory machine, we set no software limits on the sizes of the header or the images. Memory space for both are allocated at execution time, and usually at least one image of a sequence resides in core. This is an advantage, but can become a handicap if the program is one component in a long pipeline or if the computer system is under a heavy load, because the UNIX system will get very sluggish if too many huge programs are running concurrently. In our experience, however, these cases are rare, and for the type of images that we process (seldom larger than 10K pixels), the freedom from size limitations and the allocation of memory for whole images result in ease and efficiency of operation. On the other hand, this programming strategy restricts the use of HIPS on high-resolution material to systems with a large amount of core. As core becomes much cheaper, this becomes less and less of a restriction. In any case, low-resolution work and one-dimensional signal processing can be performed on even the smallest of systems.

HIPS is written in the language C (Kernighan & Ritchie, 1978). This is the language in which most of UNIX itself is written. It is a block-structured language and bears a slight resemblance to Pascal.

Available Tools

It is an unwritten rule that no program should be written, upgraded, or revised unless necessary. With few exceptions, we have adhered to this rule. The collection of tools in HIPS is capable of performing all the tasks we have required, but not of doing everything.

Currently available tools for image processing are described in the following paragraphs, grouped into broad categories according to function. This is not an exhaustive list; rather, it is intended to give an idea of the breadth and potential of HIPS. More programs and examples are given in the subsequent section, in which we discuss potential applications.

Peripheral Interface

A small number of programs directly involve peripheral devices, such as the image processor (Grinnell),

video cassette recorder (VCR), movie camera, and film projector. These are the only programs that are specifically device dependent and that would need to be modified in order to accommodate a different image processor or control of other devices. Included are programs to start image digitization, read a single digitized frame into memory, control a Betamax VCR (start, stop, pause, etc.), control a Lafayette motion analyzer (single-step a film projector for multiple-frame film input), control a Sony motion analyzer (for multiple-frame video input), single-step a Bolex film camera (for synchronized recording onto film), send single frozen frames to the image processor (for viewing on a video monitor · recording on film or video), and send image sequences to the image processor (synchronized with the video sync).

Header Manipulation

HIPS provides a small number of programs to manipulate image sequence headers. The most useful program is *seeheader* (demonstrated above), which allows the user to examine a sequence header. Other programs allow the user to add documentary infomation to the header and to extract or strip the header from a sequence. After removing the header from a sequence, especially a sequence in ASCII format, the user can then operate on the data with non-HIPS programs, such as statistical packages and editors.

Sequence Generation

In addition to creating a sequence by digitizing a video image, HIPS allows the user to create synthetic images from scratch. There are programs for creating uniform image sequences (*genframe* and *fgenframe*) and checkerboards (*checkers*). Additionally, these programs can be combined with the program *pad*, which places a given sequence in a uniform frame, in order to create a wide variety of rectangular patterns. These patterns can also be thought of as Fourier spectra, rather than as images. In this case, applying the inverse Fourier transform (also a program in HIPS, *inv.fourtr*) allows one to easily generate complex grating patterns. Synthetic images can be entered as arrays of numeric data and then converted to other image formats. Lastly, various sorts of line drawings can be created using the PLOT-3D package, which we describe below.

Operations on Sequences

A number of simple operations on image sequences are available. These programs will extract subsequences (*subseq*), create longer sequences through frame repetition or interpolation (*repframe*), and strobe sequences down to a single image by averaging (*strobe*). Separate sequences can be concatenated into single sequences by *catframes*. Lastly, for motion analysis, difference sequences can be derived such that each image in the new sequence consists of the point-by-point difference of each successive frame pair in a given sequence (*autodiff*).

Transformations on Frames

Programs are available to reduce and enlarge images (*reduce* and *enlarge*), crop, and frame images in a background (*extract* and *pad*), and rotate and reflect images (*rotate180* and *reflect*). Programs *addseq* and *mulseq* allow the user to add or multiply two image sequences point by point. For example, if the user *mulseq*s the Fourier transform of a sequence by the Fourier spectrum of a filter and then applies the inverse Fourier transform, this is equivalent to a filtering operation. Furthermore, for most filters, this method is quite a bit faster than applying the equivalent convolution operator in the spatial domain.

Operations on Pixels

It is quite common in image processing to apply the same transformation to all of the pixels in an image sequence. Programs exist in HIPS to apply logarithmic, exponential, and power function transformation (*logimg*, *stretchpix*, and *powerpix*). Other programs create photographic negatives (*neg*), perform thresholding (*thresh*), and apply linear and second-order scaling (*scale*). Lastly, a number of programs exist that convert between the various image formats, such as floating point to complex (*ftoc*), byte to ASCII (*ptoa*), and so on.

Calcpix

In an image processing laboratory, writing simple, single-pixel-oriented transformations is a fairly routine process, and such transformations are often needed for special, one-time-only purposes. One of the most useful programs we have created is a program that allows the user to define a new image-processing transformation. The program, called *calcpix*, takes as one of its parameters a line of code in the programming language C and creates a new made-to-order image-processing filter that applies this code to image sequences. This code is embedded in a program that handles header processing, input, output, and so on. This creates a program that is general with respect to image parameters (such as frame size and number of frames). The new filter is compiled and then applied to *calcpix*'s input. Thus, *calcpix* can be used in a pipeline like any other HIPS program, and yet define a totally new program. We will see several uses of this tool when we turn to applications.

Image Statistics

A number of programs have been written to compute basic statistics on pixel-formatted images, such as mean gray level, variance, entropy, and so on (*framevar* and *pixentropy*). A gray-level histogram may be computed by *histo*; this results in another image sequence in histogram format. This sequence can then be analyzed further, or converted back to pixel format by a program (*disphist*) that allows the histogram to be viewed on the image-processor output or printer.

Noise Generation

To investigate the effects of noise on the intelligibility of images, as we have been doing (Pavel, Sperling, Riedl, & Vanderbeek, 1984; Riedl, 1984), there are programs that add random noise to image sequences. Two sorts of noise have been implemented. In one case, independent samples from a Gaussian distribution (*gnoise*) are added to each pixel of an image sequence. The other noise source currently implemented in HIPS attempts to mimic digital channel noise by randomly reversing the bits in the binary representation of the image (*noise*). For each bit in the image, an independent decision is made as to whether or not to reverse that bit, according to a given probability of reversals.

Digital Transforms and Filtering

Transform processing and filtering are important features of any image-processing system. HIPS includes programs for transformation to and from the 2- and 3-dimensional Fourier domain (*fourtr*, *inv.fourtr*, and *fourtr3d*), the discrete cosine transform (subroutines *dct_2d* and *dctinv_2d*), and the Walsh transform (Gonzalez & Wintz, 1977) (*walshtr* and *inv.walshtr*). In the Fourier domain, a number of classical low-pass, high-pass, and band-pass filters can be applied, including the Butterworth, ideal, and exponential filters, using programs *lowpass*, *highpass*, and *bandpass*.

Convolution, Edge Detection, and Enhancement

Convolution, the spatial equivalent of filtering in the Fourier domain, can be accomplished directly with the program *mask*. This program allows the user to apply a number of convolution masks to an image simultaneously. The outputs of these masks may then be combined in various ways, both linearly and nonlinearly. A library of predefined mask combinations, which includes many edge-enhancement and Laplacian operators, has been written. A number of nonlinear techniques for filtering and edge detection are available, including median and extremum filtering (Lester, Brenner, & Selles, 1980) (programs *median* and *extremum*), binary noise cleaning, thinning, and thickening (Landy et al., 1984) (programs *bclean*, *thin*, and *thicken*), and the edge-detection methods of Shaw (1979) (program *discedge*) and Abdou (1978) (program *abdou*).

Transmission and Compression

Our own work has involved the examination of a number of image-storage and transmission methods. These techniques allow the user to utilize the redundancy in an image sequence in order to store the image sequence more efficiently. Programs have been written for a variety of hierarchical encoding methods (Pavlidis, 1982) (programs *hc_bin*, *hc_bin_r*, *binquad*, *binquad_r*, *ahc3*, and *ahc3_r*) and DPCM encoding (Gonzalez & Wintz, 1977) (programs *dpcm_r* and *dpcm_t*). The UNIX system that we use (Berkeley 4.2) includes its own program for adaptive Huffman coding (Huffman, 1952), which is useful as a baseline for analysis of other techniques.

3-Dimensional Plotting

A subpackage within HIPS has been written for generation and manipulation of 3-dimensional line drawings. A large number of programs are available in this package, allowing the user to create images populated with various solid objects drawn in outline (*gpoly* and *gcube*). These objects can be moved and rotated (*tshift* and *trot*), as can the viewpoint of the scene (*vshift* and *vrot*). This allows the user to to generate scenes that move and rotate in either parallel or polar perspective (using *view*). This format is fully convertible to and from pixel format (*plot3topix* and *pixto3d*), allowing line drawings to be further analyzed by any and all of the above-mentioned transformations.

POTENTIAL APPLICATIONS OF IMAGE PROCESSING

The software that we have just overviewed was written specifically for a project concerning the perception of images of American Sign Language (Sperling, 1980, 1981; Sperling et al., 1983). It is our purpose here to demonstrate that the potential applications of a system such as this to psychology go well beyond the particular project for which it was developed. We provide 10 examples to illustrate how an image-processing system such as HIPS can be a useful tool for the experimental psychologist. We outline how HIPS can profitably be employed for other areas of vision research and for the processing of one-dimensional signals.

Visual Spatial Frequency Channels: Examples 1-3

Sine-Waves and Linear Systems Analysis

Image-processing tools have natural applications to studies of spatial vision. Ever since the introduction of sine-wave-modulated flicker to the study of temporal factors in vision (DeLange, 1952; Ives, 1922) and of the sine-wave grating to studies of spatial vision (e.g., Robson, 1966; Schade, 1956), the use of linear systems theory to characterize human performance in visual tasks has been widespread. An image-processing system can be useful in this kind of research, both as a tool for simulating models of human performance and for the generation of psychophysical stimuli.

Models of human performance in visual tasks typically postulate a number of simple linear or quasi-linear transformations of the input luminance profile (Campbell & Robson, 1968; Graham & Nachmias, 1971; Wilson & Bergen, 1979) followed by a nonlinear decision process that generates the response (Sperling, 1964). At each step in the process, a noise source may corrupt the information.

Simulation

Each of the stages proposed for human processing of visual information is easily simulated in an image-processing environment such as HIPS. The input image is generated within HIPS or is photographed and digitized. Filtering operations corresponding to the optics of the eye and the transformations that we assumed for the various sorts of geniculate or cortical cells (or, equivalently, by "psychophysical" channels) can be applied. This can be accomplished either by convolution with a receptive field profile or by filtering in the Fourier domain. Other linear and nonlinear operations can be applied subsequently, and statistics can be computed in order to yield the simulated response of the subject. At any stage, the noise programs may be used to add noise to the system at that point.

Example 1: Spatial Filtering of the Müller-Lyer Stimulus

The first example comes from Ginsburg (1978), whose dissertation is concerned with the spatial frequency channels hypothesis (Graham, 1981), and the application of this hypothesis to a wide variety of perceptual phenomena. A good deal of this work involves the transformation of images (derived from text and from visual illusions) by linear filters (derived from the human MTF—modulation transfer function). For example, Ginsburg proposed that the familiar Müller-Lyer illusion is a direct consequence of low-pass spatial frequency filtering, that is, of the output of the low-frequency channel. Figure 2 illustrates the familiar illusion and an "MTF-L" (Ginsburg's terminology) filtered version of it. The hypothesis is that the perceived difference of length of the two central line segments in Figure 2a derives from the output of the low-frequency channel, represented in Figure 2b. On the other hand, it turns out that a high-pass filtered representation of the Müller-Lyer stimulus also yields the illusion (Carlson, Anderson, & Moeller, 1980), so simple low-pass filtering cannot be the whole explanation.

Figure 2 is generated as follows. First, the Müller-Lyer stimulus is created by typing the image in ASCII as an array of 0s and 1s (Figure 2c). This gives a coarse approximation to the illusion similar to that used by Ginsburg (1978). (A more typical Müller-Lyer stimulus can easily be obtained by digitizing a photograph of it.) The array of Figure 2c is converted to byte format as follows:

```
atob -c 32 < ml > ml1
```

which converts from ASCII to byte format (*atob*) and creates (*-c*) an image sequence in byte format with 32 x 32 pixels and one frame. The pixels are read from file ml, and the output is placed in file ml1. Next, the Fourier transform of Ginsburg's MTF-L (5 x 5) is typed, similarly to the image. This is converted to image format by *atof*:

```
atof -c 64 < mtfin > mtfin1
```

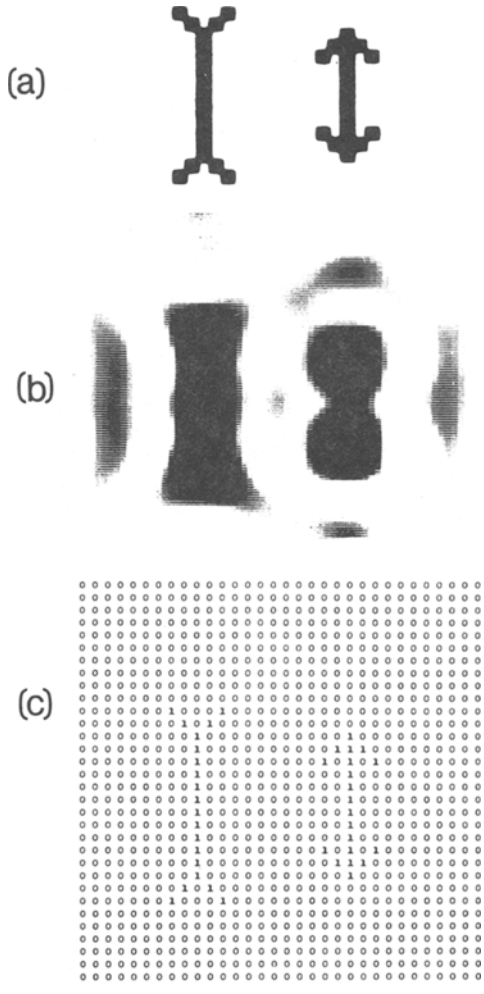


Figure 2. (a) Reconstruction of a demonstration by Ginsburg (1978) of the classical Müller-Lyer illusion. (b) MTF-L (5 x 5) version of the stimuli in (a). This demonstration purports to explain the illusion as resulting from the representation of the stimuli in a low spatial frequency channel, such as the one pictured here. For printing purposes, (a) and (b) are actually photographic negatives of the images as they appear on the display monitor. (c) The ASCII input used to create image (a).

which creates a 64 x 64 frame in floating-point format. The original illusion is displayed on the image processor's video monitor by the command

```
enlarge 4 < ml1 | wframe
```

which enlarges the image by a factor of four and sends it off to the image processor by *wframe* ("write frame") to be converted to standard video. Figure 2a is a photograph of this video image.

The filtered version of the Müller-Lyer stimulus shown in Figure 2b is created with the following pipeline:

```
enlarge < ml1 | btof | fourtr | mulseq mtfin1 | inv.fourtr -f | \
        scale | enlarge 4 | wframe
```

This pipeline enlarges the illusion frame to 64 x 64 pixels, converts it to floating-point format, and transforms it to the Fourier domain. The Fourier frame is then multiplied by the low-pass filter spectrum and returned to the spatial domain. At this point, the frame is still in floating-point format. The program *scale* converts the frame to byte format, linearly scaling the values to fill the entire dynamic range. Lastly, the image is enlarged and converted to video.

Example 2: Sine- and Square-Wave Gratings

HIPS can be used for the generation of psychophysical stimuli. The image-processing primitives may be combined to form complex sine- and square-wave gratings, masked by noise, tapered, and so on. We illustrate the creation of complex stimuli in Examples 3-10, but include this example here in order to introduce the ideas.

There are several ways to generate a grating pattern using HIPS. The simplest method is to use *fcalcpix*, the floating-point version of *calcpix*, as follows:

```
fgenframe 0 128 | fcalcpix "opix = sin(3.1415926 * c / nc);"
```

The program *fgenframe* (generating floating-point frames) generates a uniform frame of size 128 x 128 with all gray values set to zero. *Fcalcpix* then converts this image to a sine-wave grating of one period and vertical orientation. The *fcalcpix* calcuation utilizes the variables *c*, which is the column number of this pixel, and *nc*, which is the number of columns in the image. The input image (from *fgenframe* or any other source) can be of any dimension and still yield a sine-wave grating with one period and vertical orientation. Variables *c* and *nc* are predefined within *fcalcpix* (and the byte-formatted version of the program, *calcpix*), along with other variables for the row (*r* and *nr*), the frame (*f* and *nf*), and temporary variables made available to the user. In any case, given the grating pattern, the program scale may then be used in order to control mean luminance and contrast.

A second method for the generation of sine-wave gratings involves the Fourier transform. Since gratings have particularly simple Fourier spectra, this method of generating gratings begins by using HIPS to create the spectrum and then converting it to the spatial domain. The pipeline

```
fgenframe 1 1 | pad 0 128 128 0 1 | inv.fourtr -f
```

accomplishes this by first creating the spectrum [a 1 in pixel (0,1), and 0s elsewhere] and then applying the inverse Fourier transform, with the output constrained to be in floating point. In actuality, because the spectrum we generated was asymmetric (in the discrete Fourier transform sense), its inverse is complex. Constraining the inverse transform to give only the real part yields the desired grating pattern.

Creating a square-wave grating pattern is simple, given a sine-wave grating. Applying the program *thresh*, which can be directed to threshold the sine-wave grating at zero, yields a square-wave grating of equal black and white bar sizes. Again, *scale* can be used to control mean luminance and contrast. Lastly, different gratings may be combined by using *addseq* and *mulseq*.

Example 3: Frequency Band-Limited Spatial Patterns

Mostafavi and Sakrison (1976) measured the detection thresholds for two kinds of complex pattern stimuli containing only a narrow band of spatial frequencies, but with scrambled phase. Figure 3a illustrates a HIPS-generated pattern corresponding to their isotropic filter case; Figure 3b illustrates their nonisotropic case, in which the angle of the components is restricted to be close to the vertical. Mostafavi and Sakrison generated these stimuli by first creating a Gaussian noise pattern, and then by filtering either isotropically:

$$H(f) = \exp\left[\frac{-\left[\frac{f - f_0}{\omega}\right]^2}{2}\right]$$

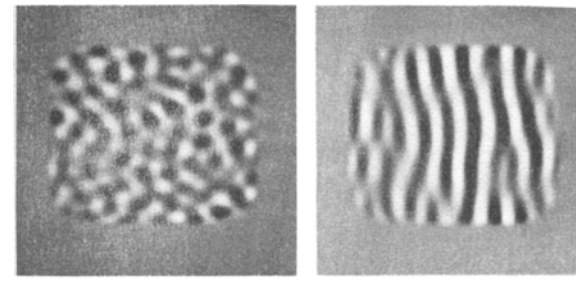
or nonisotropically:

$$H(f,\theta) = \exp\left[\frac{-\left[\frac{f - f_0}{\omega_1}\right]^2}{2}\right] \cdot \exp\left[\frac{-\left[\frac{\theta - 0\ \text{deg}}{\omega_2}\right]^2}{2}\right] \cdot \exp\left[\frac{-\left[\frac{\theta - 180\ \text{deg}}{\omega_2}\right]^2}{2}\right],$$

where f and θ are the frequency and angle, respectively, of the Fourier transform of the filter in polar coordinates, f_0 is the center frequency of the passband of the filter, ω and ω_1 are the spatial half-bandwidths of the isotropic and nonisotropic filters, respectively, and ω_2 is the angular half-bandwidth.

The filtering operations are performed quite simply using HIPS. The isotropic filter is generated with the following command string:

```
fgenframe 0 256 256 | fcalcpix "\
        d1 = (r <= nr/2) ? r : r-nr;\
        d2 = (c <= nc/2) ? c : c-nc;\
        opix = exp(-pow(((sqrt(d1*d1+d2*d2)-8.)/\
                2.),2.)/2.);" > symfilter
```



(a) (b)

Figure 3. (a) Isotropically and (b) nonisotropically bandpass filtered Gaussian noise (after Mostafavi & Sakrison, 1976).

As before, a single uniform frame is generated, and then filled using *fcalcpix*. The Mostafavi and Sakrison (1976) equations treat the Fourier domain in polar form; much of the *fcalcpix* parameter string converts the row and column (*r* and *c*) to frequency (using the predefined temporary variables *d1* and *d2*), keeping in mind the folding of the discrete Fourier transform. The final result is stored in the output pixel *opix*. The nonisotropic filter spectrum is generated as follows:

```
fgenframe 0 256 256 | fcalcpix " \
        #define PI 3.1415926 \
        d1 = (r <= nr/2) ? r : r-nr; \
        d2 = (c <= nc/2) ? c : c-nc; \
        d3 = sqrt(d1*d1+d2*d2); \
        d4 = (d2 != 0.) ? atan(d1/d2) : PI/2.; \
        opix = exp(-pow(((d3-8.)/2.),2.)/2.) * \
        exp(-pow(((d4-0.)/.2),2.)/2.); " > asymfilter
```

The additional complication in the nonisotropic case is the need to define the filter for both the frequency and the angle.

Figures 3a and 3b are generated by first generating Gaussian noise, applying the filters in the Fourier domain, and then inverse transforming back to spatial coordinates. The pipelines for Figure 3a are:

```
fgenframe 0 256 256 | gnoise 1 | fourtr | mulseq symfilter | \
        inv.fourtr -f >symout
genframe 128 512 512 | wframe
mulseq taperfilt < symout | scale 2560 128 | wframe
```

The first line generates the Gaussian noise, converts to the Fourier domain and applies the isotropic filter, and then converts to the spatial domain. The next line fills the monitor screen with a uniform gray background. The last line smoothly tapers the edges of the pattern and embeds this image in the gray background, scaling the floating image for the same mean luminance as the background. The nonisotropic filter, asymfilt, is applied instead in order to derive Figure 3b.

The tapering mask was not described in Mostafavi and Sakrison (1976), and we generate ours as follows:

```
fgenframe 0 256 256 | fcalcpix " \
        i1 = (r>(nr/2)) ? r-(nr/2) : (nr/2)-r; \
        i2 = (c>(nc/2)) ? c-(nc/2) : (nc/2)-c; \
        d1 = i1*2./nr; \
        d2 = i2*2./nc; \
        d3 = (1.-pow(pow(d1,5.)+pow(d2,5.),1./5.))*8.; \
        if (d3<0.) d3=0; \
        if (d3>1.) d3=1; \
        #define PI 3.1415926 \
        d3 = (d3-.5)*PI; \
        opix = (sin(d3)+1.)/2.; " > taperfilt
```

Form Perception: Examples 4-5

HIPS can be used for studies of the perception of form. Form stimuli can be generated using HIPS, by using the PLOT-3D package to generate line drawings, by synthetically generating form stimuli, or by digitizing pictures of stimulus materials. All these stimuli, however created, may be processed by HIPS in order to simulate models of form perception.

Example 4: The Kinetic Depth Effect

Schwartz and Sperling (1983) studied the perception of direction (and form) in the kinetic depth effect. The stimuli in their experiments were 2-dimensional projections of rotating 3-dimensional wire cubes. The 2-dimensional projections were inherently ambiguous and could be perceived as rotating in either of two directions. These dynamic 2-dimensional projections were biased for particular directions of rotation by adding various depth cues. In other cases, opposing cues were pitted against each other.

Figure 4 shows one frame of a sequence that contains two cubes, one inside the other, plotted in polar perspective. Both cubes are rotating about the same vertical axis. Either cube can be perceived as rotating in either direction. Because of polar perspective, perceiving one of these directions will cause the cube to be perceived (correctly) as a rigid 3-dimensional cube; the other direction will cause the cube to be perceived (incorrectly) as a nonrigid, rubbery, distorting 3-dimensional object. Spontaneous reversals of depth occur while viewing this pattern, causing the direction of rotation and perceived form to reverse, for one or both cubes. When the cubes are depicted as rotating in opposite directions, the cues of rigidity (provided by polar perspective) and context (provided by rigid notation of the other cube) are in conflict.

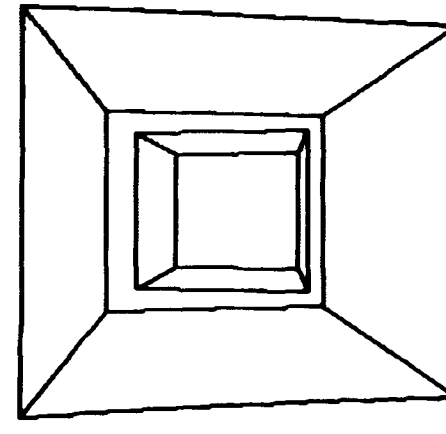


Figure 4. One frame from a sequence depicting two 3-D wire cubes shown in polar perspective, rotating about a common vertical axis (based on Schwartz & Sperling, 1983). When the sequence is viewed, observers perceive each cube rotating in either of two directions. In one of these modes, the cube is rigid; in the other, it is distinctly nonrigid. By rotating the two cubes in opposite directions, this demonstration pits the tendency to perceive objects as rigid against the tendency to perceive objects as rotating in the same direction as their context. For printing purposes, this figure is actually a photographic negative of the image as it appears in the display.

The PLOT-3D package allows the user to generate 2-dimensional projections of 3-dimensional wire objects, such as these cubes, quite easily. The frame in Figure 4 is taken from a sequence generated as follows:

```
gcube | gmag 100 100 100 | trot 29 0 3 > x
gcube | gmag 200 200 200 | trot 29 0 -3 > y
gsync x y | gshift 0 0 600 | view | plot3topix | bpack | bmovie
```

The first two lines generate cubes of two different sizes rotating in opposite directions. The program *trot* is requested to rotate the cubes in 3-deg increments for 30 frames in all. Because of the symmetry of the cube, 90 deg of rotation is sufficient to represent the entire stimulus, and repetitive presentations yield a smoothly rotating cube. The *gsync* command combines the two cubes frame by frame. *Gshift* moves the scene away from the viewer, and *view* applies the perspective transformation. The plot is then converted to pixel format, and then to binary packed format (for more efficient presentation). Finally, *bmovie* presents the binary sequence repeatedly on the video monitor.

Example 5: Form Perception and Spatial Frequency Channels

One of the more useful features of HIPS is the ease of transformation to and from the Fourier domain. This feature may be employed in studies of form perception in which the application of a channels model is envisioned, as in recent studies by Palmer and Bucher (1981, 1982) of the disambiguation of an ambiguous form. Their most often used form is an outline drawing of an equilateral triangle (first suggested by Attneave, 1968). This figure is perceived as an arrow pointing (ambiguously) in one of three directions. Palmer and Bucher manipulated perceived direction by manipulating the context in which the triangle appeared. For example, Figure 5a shows an image of three triangles aligned along their bases. Palmer and Bucher found that subjects most frequently perceived the triangles in this stimulus as pointing upwards.

It is natural to apply a Fourier model to this phenomenon (Janez, 1983). Janez's hypothesis is that the subject transforms the image to the Fourier domain and computes the average spectral energy at various angles. The direction with the most energy is used to define a pair of orthogonal axes that are then applied as a reference frame to the figure. If either axis aligns with a corner of the triangle, it is posited that the triangle is most readily perceived as pointing in that direction. Thus, the bases-aligned condition of Figure 5a should be perceived as pointing vertically if more spectral energy is contained in the vertical direction than in any other direction.

HIPS is easily applied to the problem of finding the direction of maximum spectral energy. The bases-aligned stimulus is generated by the PLOT-3D system:

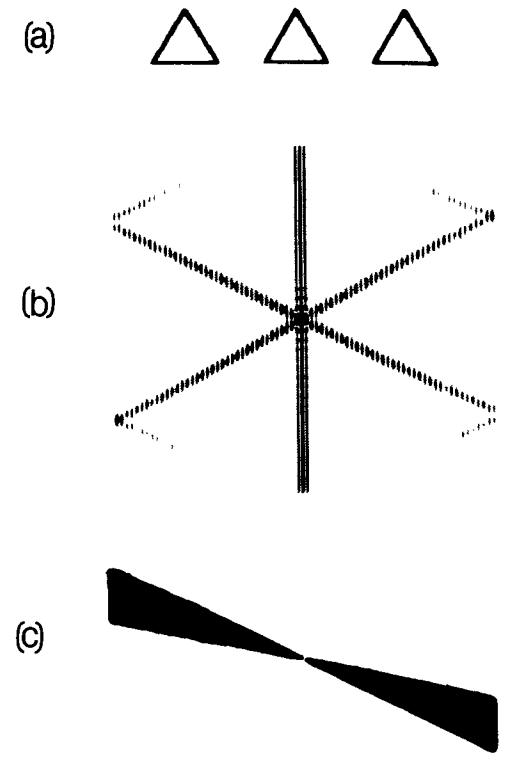


Figure 5. (a) Three equilateral triangles in the bases-aligned condition described by Palmer and Bucher (1982). In this configuration, triangles are perceived as pointing upward. Fourier analysis of this phenomenon suggests that the ambiguity is resolved by finding the direction in which there is peak spectral energy. For example, Janez (1983) proposed that the perceived direction of pointing is either parallel or perpendicular to the direction of peak spectral energy. (b) The Fourier power spectrum of (a); peak spectral energy is in the vertical direction, the perceived direction of pointing. (c) An angular ray pattern for the directional analysis of (b). For printing purposes, (a), (b), and (c) are actually photographic negatives of the images as they appear in the display.

```
gpoly 3 | gshift 6 | gpoly 3 | gshift -3 | gpoly 3 | \
    gmag 25 25 25 | plot3topix | \
    extract 256 256 128 128 | bpack > bal
```

This pipeline generates three triangles (with *gpoly*) and shifts them to be side by side (with *gshift*). It then magnifies the vector drawing (with *gmag*, which magnifies the drawing by a factor of 25 in x, y, and z), converts to pixel format, and crops the image to 256 x 256 pixels. This image is finally stored in file "bal" ("base-aligned") in bit-packed format (for space-saving considerations, since the figure is, after all, binary). The Fourier spectrum of the figure is created by

```
bunpack < bal | fourtr -s > bal.spectrum
```

This spectrum, shown in Figure 5b, is displayed on the monitor with the following command:

```
scale < bal.spectrum | wframe
```

The next step is to create images that extract the spectral energy at given angles. This is accomplished by creating binary images that are nonzero only for a given span of angles. In this case, we create patterns centered at angles of 0, 5, 10, ... 180 deg and of widths of 15 deg. The 20-deg pattern is shown in Figure 5c. These patterns are generated by writing a HIPS pipeline using *calcpix*. The pipeline is then repeatedly run using the ability of the Berkeley shell (Joy, 1980) to "program" the running of programs. The following shell script runs the pipeline over and over again, with shell variable i taking on the values 0, 5, 10, ... 180 and inserting that value in the *calcpix* argument. These patterns are then stored (in bit-packed format, using *bpack*) in files patt0, patt5, patt10, ... patt180.

```
@ i = 0
while ($i <= 180)
echo $i
genframe 0 256 256 | \
        calcpix "i1 = r - nr/2; i2 = c - nc/2; \
        d1 = (i2!=0) ? atan(((double) i1)/i2)*180./ \
             3.1415926 : 90.; \
        d2 = fabs( $i - d1); \
        opix = (d2 <=7.5 || (d2 >= 172.5 && d2 <= \
              187.5)) ? 1 : 0;" \
        | bpack > patt$i
@ i = $i + 5
end
```

Next, two analyses, energy computation and normalization, are carried out. First, the spectrum of Figure 5b is multiplied by each ray pattern, and the average energy in that ray is computed by summing the squared magnitudes of all the resulting components. Second, since each ray contains a different number of components (we are in discrete Fourier space in a square image, rather than a circular image), the average pixel value of each ray pattern is computed (which is proportional to the number of nonzero elements in the pattern). Dividing these two numbers yields a normalized result that is proportional to the average spectral energy in the angular band.

The energy computation is carried out in the following shell script:

```
@ i = 0
while ($i <= 180)
echo $i
bunpack -b < patt$i | btof | mulseq $1 | framevar
@ i = $i + 5
end
```

This script multiplies the power spectrum by each ray pattern (after first unpacking the ray pattern and converting it to floating-point format, using *bunpack* and *btof*) and computes the frame statistics with *framevar* (which yields both mean and variance). The mean gray level of the image consisting of the power spectrum multiplied by the ray pattern is used here and is proportional to the average spectral power in the angular ray.

We then carry out the same computation of average gray level for the ray patterns alone:

```
@ i = 0
while ($i <= 180)
echo $i
bunpack -b < patt$i | framevar
@ i = $i + 5
end
```

Dividing the means for each angle from the energy computation by those from the normalization yields the final result. As can be seen from the results given in Table 1, the energy is most concentrated at an angle of 90 deg, which should imply a set of axes oriented vertically and horizontally, and the triangles should be perceived most readily as pointing upward.

Studies of Stereopsis and Motion Perception: Examples 6-7

HIPS can be used to generate a wide variety of stimuli used in the study of binocular stereopsis and motion

Table 1
Average Spectral Power in Various Angular Bands—a Fourier Analysis of the Bases-Aligned Stimulus of Palmer and Bucher (1981) Shown in Figure 5a

Angle	Average Power	Angle	Average Power	Angle	Average Power	Angle	Average Power
0	19	45	23	90	100	135	23
5	20	50	20	95	91	140	31
10	23	55	18	100	33	145	65
15	30	60	17	105	21	150	76
20	46	65	17	110	18	155	74
25	77	70	18	115	17	160	35
30	74	75	20	120	18	165	27
35	59	80	32	125	19	170	23
40	29	85	91	130	21	175	21

Note—Columns show the orientation angle of the center of the band and the average spectral power in each band (normalized to the largest power) for the spectrum shown in Figure 5b. Maximum power is at 90 deg, consistent with Janez's (1983) model for the resolution of the ambiguous pointing.

detection. This includes random-dot stereograms and cinematograms (Julesz, 1971), disparate views of static and moving 3-dimensional line drawings (generated with PLOT-3D), and stimuli involving random-dot motion for image-segregation studies (e.g., Anstis, 1970; Baker & Braddick, 1982; Braddick, 1974; Nakayama, 1981; cf. Sperling, 1976). Sequences generated in this fashion can be used as stimuli for psychophysical experiments, and they can then be transformed to pixel format and subjected to various other transformations (e.g., spatial filtering) in order to verify the action of a proposed model upon them. We consider two examples of the generation of random-dot stimuli.

Example 6: Random-Dot Stereograms and Cinematograms

Stereograms. The generation of random-dot stereograms is quite easy using HIPS. Figure 6a shows a random-dot stereogram on a 100 x 100 grid with 25% black pixels, illustrating a central square floating in depth above a surround. The central disparity is three pixels. The figure is generated with the following commands:

```
genframe 128 100 100 | shiftpix -7 | noise .25 | shiftpix 7 > right
calcpix "if (r>= 25 && r<75) { \
            if (c>=25 && c<75) \
                opix = picin[r][c-3]; \
            else if (c>=22 && c<25) \
                opix = picin[r][c+53]; \
        }" < right > left
genframe 128 512 512 | wframe
pad 0 110 110 5 5 < left | enlarge | wframe 140 22
pad 0 110 110 5 5 < right | enlarge | wframe 140 282
```

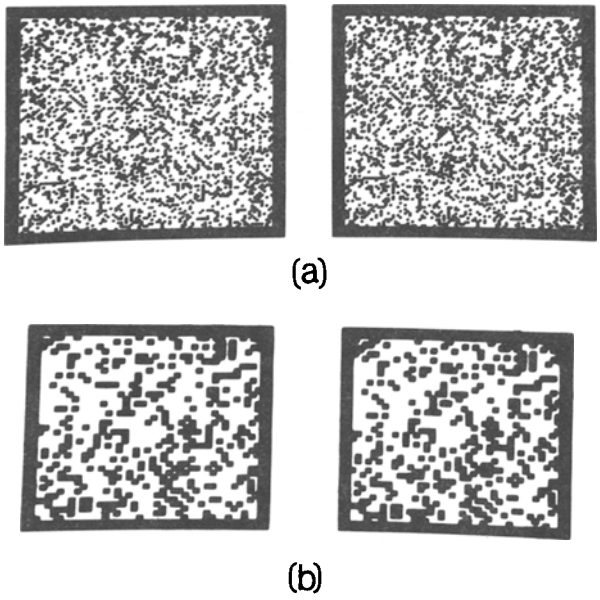


Figure 6. (a) A random-dot stereogram (after Julesz, 1971). (b) Two consecutive frames from a random-dot motion sequence (after Baker & Braddick, 1982). The dots in the center square move upward, while those in the surrounding area move downward.

The first command creates a random-dot pattern by generating a uniform 100 x 100 frame, shifting each pixel so that there is only one bit per pixel, adding bit reversal noise to 25% of the bits (creating 25% random black pixels), and then shifting the pixels again for contrast. This image is presented to the right eye. The second command (*calcpix*) creates the left eye's image by adding a crossed disparity of three pixels in the central 50 x 50 square. The pixels that are uncovered by shifting the square to the right are grabbed from those that are covered by it on the right. The last three commands present the patterns on the monitor with a uniform black border on a white background to facilitate binocular fusion.

Cinematograms. A cinematogram is a sequence of stereograms. In this next example, each frame of the cinematogram presents the same disparity information—a square floating above a surround—but the points are chosen randomly and independently in successive frames. The cinematogram is a simple extension of the stereogram. It is generated by the following commands:

```
genframe 128 60 100 100 | shiftpix \
  -7 | noise .25 | bpack > right
bunpack -b < right | calcpix "if (r>=25 && r<75) { \
            if (c>=25 && c<75) \
                opix = picin[r][c-3]; \
            else if (c>=22 && c<25) \
                opix = picin[r][c+53]; \
        }" | bpack > left
bunpack < right | pad 0 104 104 | pad 255 104 114 0 10 | \
        pad 0 104 218 0 104 | bpack > b.right
bunpack < left | pad 0 104 104 | pad 255 104 114 0 0 | \
        pad 0 104 218 0 0 | bpack | andseq b.right > combined
bmovie < combined
```

The first command generates a sequence of 60 independent random-dot patterns. The second command is identical to that in the previous example and generates the left eye's disparate pattern. The next two commands generate the borders and combine the sequences into one side-by-side set of images. The last command shows the sequence repeatedly on the monitor. When viewed through a stereoscope, the sequence is perceived as a dynamically noisy square floating above an equally noisy surround.

Example 7: Image Segregation With Differing Dot Motion

In an experiment by Baker and Braddick (1982), a field of random dots was presented in which the dots moved in a different manner in different portions of the field. The following pipeline illustrates how such a stimulus might be generated. It creates a sequence in which dots within a central square move upward, while dots in the surrounding area move downward.

```
genframe 128 40 40 | shiftpix -7 | noise .25 | \
repframe 40 -n | calcpix \
"if (r>=10 && r<30 && c>=10 && c<30) \
                opix = picin[10+((r-10+f)%20)][c]; \
        else \
                opix = picin[((40+r-f)%40)][c]; " | \
pad 0 44 44 2 2 | enlarge | bpack | bmovie
```

This pipeline first generates a sequence of identical 40 x 40 random-dot fields. The *calcpix* command then performs the dot shifting, moving each dot up or down by the number of the given frame *f*. This sequence is then given a border with *pad*, enlarged by a factor of two, packed so that only one bit per pixel is used, and then presented on the monitor. Two frames of this sequence can be seen in Figure 6b, where close examination reveals the dot motion. Viewing this sequence, subjects easily segregate the central square from the surround.

Other applications to motion and stereopsis. There are several instances in which filtering or other image transformations might be useful in stereopsis research. The most obvious example is generating stereograms in which one or both of the images are filtered in various ways (Julesz, 1971). A second example would be in cases in which a channels model might be invoked to explain a certain phenomenon. In the letter stereograms of Kaufman and Pitblado (1965), different rivalrous characters in the left and right images can produce a sensation of depth, presumably based on gross features of the letters, such as size and font. In current terms, this phenomenon is likely to be based on processing in low-frequency channels. As we have seen, predictions of a spatial-frequency-channel analysis can be tested using image-processing software, which can filter images (spectrally or by convolution with receptive fields) and further analyze the resulting images, or present the filtered versions as psychophysical stimuli to determine whether the perception of depth is preserved. Similarly, spatial filtering of motion stimuli may help to define the characteristics of channels responsible for motion detection.

The Effects of Visual Noise: Examples 8-9

Because HIPS allows for the generation of various sorts of static and dynamic noise, it is a flexible tool for the generation of stimuli involving noise and for the examination of models in which noise plays a significant role. Two examples are chosen from studies involving noise stimuli; the first involves sensory psychophysics, and the second involves cognitive, linguistic elements.

Example 8: Noise in Psychophysical Experimentation

In his thesis, Pelli (1981) analyzed the internal noise in psychophysical spatial frequency channels by a common technique used in the analysis of analog electronics: determining the input noise that would produce an equivalent response in an internally noiseless device. His target stimuli were 1-dimensional, sinusoidal gratings masked by dynamic noise. The gratings were tapered in the temporal and in both spatial dimensions by Gaussian windows. The target gratings were masked by either 1- or 2-dimensional dynamic noise, and the threshold modulation for the grating was determined. The noise was generated by a simple electronic circuit, depicted in Figure 7a. The circuit utilized a 31-bit shift register, in which each noise sample was taken from the rightmost bit (yielding a 0 or a 1) and then the register was shifted one position to the right, with the previous bits from positions 13 and 31 exclusive or-ed and put back in bit position 1. This digital noise generator yields a pseudorandom sequence of 0s and 1s that takes $2^{31}-1 \approx 10^{10}$ shifts before the pattern repeats. In one condition, the noise sequence was then interpreted as a sequence of +1s and −1s, scaled by a noise contrast factor and added to the target grating. This method yields a dynamic "snow" added to the grating pattern.

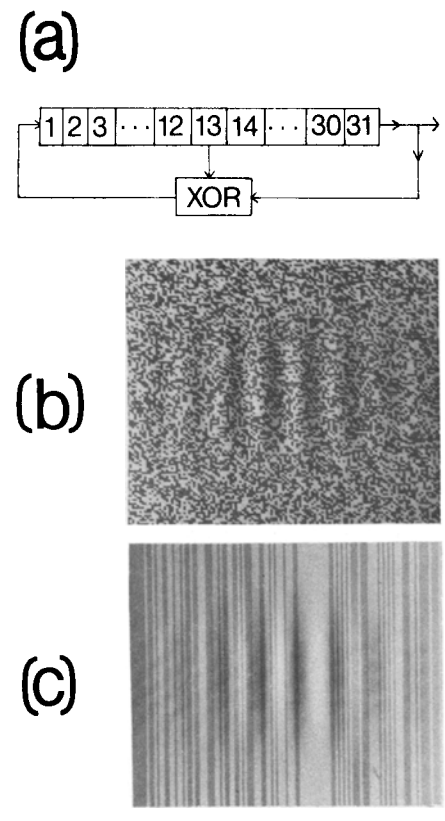


Figure 7. Edge-tapered grating patterns with added noise. (a) Block diagram of the noise source. (b) Vertical sine-wave grating with added 2-dimensional noise. (c) Vertical sine-wave grating with added 1-dimensional noise. (After Pelli, 1981)

In the other condition, the noise values for one raster scan line were repeated on all other scan lines, yielding a 1-dimensional noise mask.

We simulate both noise conditions using the HIPS software. Consider first 2-dimensional noise created, along with the target, by the following HIPS command:

```
fgenframe 0 128 128 | fcalcpix -o sinewave \
        "opix = sin((double) c * 5. * 2 * 3.1415926 / nc);" \
        | repframe 60 -n | fcalcpix -o taper \
        "opix = ipix * exp(-pow(((double)c\
                -(nc/2))/(nc/4),2.)/2.) \
        * exp(-pow(((double)r-(nr/2))/(nr/4),2.)/2.) \
        * exp(-pow(((double)f-30)/(f/5),2.)/2.);" \
        | fcalcpix -o bitnoise "if (first) {il = 0x7f81ea92; \
                first = 0;}\
        i2 = i1 & 1; i3 = (i1>>18)&1; i1 = (il>>1)&0x3fffffff; \
        i1 |= ((i2 ^ i3) & 1) << 30; opix += i2 ? 1 : -1;" \
        | scale 50 128 | movie -d
```

This command generates a uniform frame and utilizes *fcalcpix* to create a vertical sinusoidal grating. *Repframe* is used to create a sequence of 60 frames by frame repetition (adding new frame using *-n*). The next *fcalcpix* command applies a Gaussian taper to each of the three dimensions. The final *fcalcpix* command simulates the shift-register noise generator and adds or subtracts the noise values from each pixel. This sequence is then scaled and converted to byte format, and presented on the monitor. Figure 7b shows the middle frame of this sequence.

For the case of 1-dimensional noise, the situation is slightly different, and it turns out that a sequence must be stored temporarily. The command sequence for this case (whose output is illustrated in Figure 7c) is the following:

```
fgenframe 0 128 128 | fcalcpix -o sinewave \
        "opix = sin((double) c * 5 * 2 * 3.1415926 / nc);" \
        | repframe 60 -n | fcalcpix -o taper \
        "opix = ipix * exp(-pow(((double)c\
                -(nc/2))/(nc/4),2.)/2.) \
        * exp(-pow(((double)r-(nr/2))/(nc/4),2.)/2.) \
        * exp(-pow(((double)f-30)/5,2.)/2.);" > temp
fgenframe 0 60 1 128 | \
        fcalcpix -o bitnoise "if (first) {i1 = 0x7f81ea92; \
                first = 0;}\
        i2 = i1 & 1; 13 = (i1>>18)&1; i1 = (i1>>1)&0x3fffffff; \
        i1 |= ((i2 ^ i3) & 1) << 30; opix += i2 ? 1 : -1;" \
        | enlarge 128 -v | addseq temp \
        | scale 50 128 | movie -d -l2
```

The first command generates a tapered grating and saves it in a temporary file. The second command generates shift-register noise in a 1 x 128 pixel sequence and then enlarges that sequence vertically in order to create the 1-dimensional noise. This sequence is then added to the tapered grating sequence, and the result is treated as before.

Example 9: Noise in Perceptual Experiments

The next example involves a study from our own laboratory. In this work (Pavel et al., 1984; Riedl, 1984), sequences of video images of a deaf person communicating in American Sign Language (ASL) are presented to deaf subjects in order to gauge comprehensibility of the sequences. In different conditions, different amplitude noise is added to the ASL image sequences. The intelligibility of the resulting sequences yields an index against which to measure the resistance of ASL to more general distortions and as part of our continuing investigations into the perceptual features of ASL images.

Since investigations of noise-perturbed ASL image sequences using HIPS will be reported in more detail in the following paper (Riedl, 1984), we will only remark here that HIPS efficiently accomplishes the component tasks required for generating stimulus materials for formal tests. HIPS is used to automatically digitize and archive the ASL image sequences from VCR and film originals. Image transformations are then applied in order to vary contrast and noise content. These new stimuli are then arranged in counterbalanced orders and recorded on a VCR automatically in preparation for their use as stimulus tapes for our experiments.

Applications of Image Processing to American Sign Language (ASL)

Our research on the application of image processing to the perception of ASL was the original impetus for the development of HIPS, and therefore the greatest number of programs were written for that project. This research involves the transformation of video ASL image sequences in a wide variety of ways in order to gauge the effects of such transformations on the intelligibility of the images. A large number of programs were written for standard image-processing transformations, including gray-scale stretching, contrast enhancement, edge enhancement and detection, filtering and convolution, thresholding, and so forth. Because the project involves not only the intelligibility of distorted images, but also the ability to efficiently encode and transmit this information, we have also programmed a number of image-coding techniques. Since this paper is intended as an overview of potential uses for image processing, we will restrict ourselves to just one example.

Example 10: Hierarchical Coding of Edge-Detected ASL Image Sequences

Figure 8a illustrates one frame from a "cartoon" sequence of ASL images. The image is generated by the following command:

```
rframe 384 256 48 128 | reduce 4 | btof | scale | dog .6 7 -i | \
        thresh 90 | neg | bclean 3 | neg | bpack
```

This command digitizes a single frame from the video

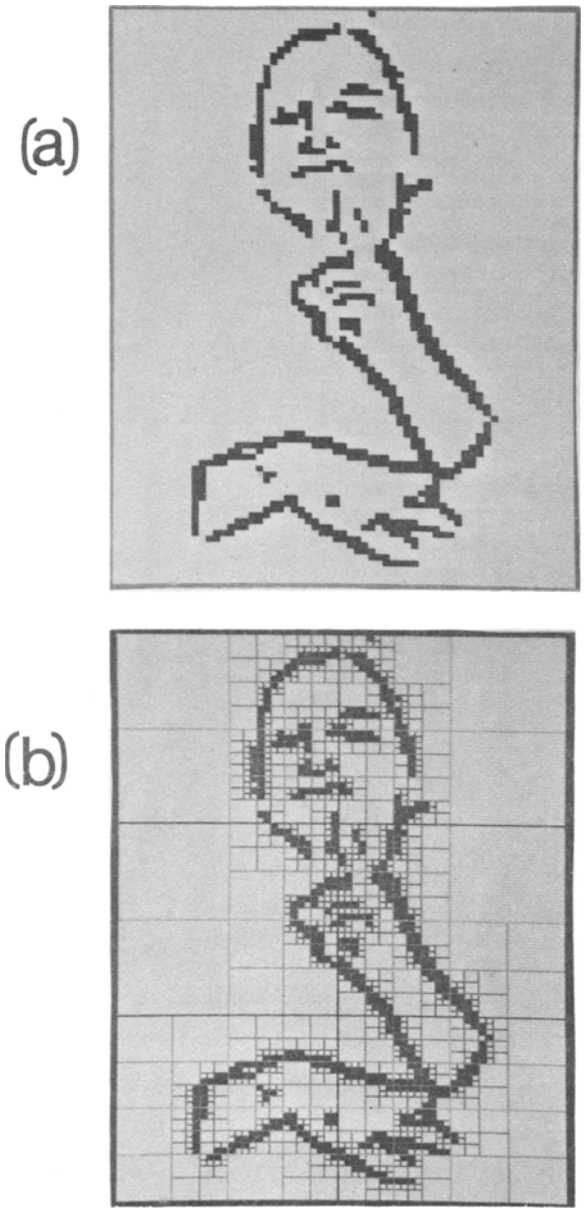


Figure 8. ASL imagery. (a) A frame from an ASL cartoon produced by darkening the 10% most negative points in a DOG transformation of the original. Both the original and cartoon are 96 x 64 pixels; additional "cleaning up" of small groups of black pixels has been applied to the cartoon. (b) A representation of the "cuts" in a quadtree encoding of the image in (a). (After Cohen, Landy, & Pavel, 1983)

input, reduces it in size by a factor of four, and contrast-enhances it. Then a variant of the edge-detection algorithm of Marr and Hildreth (1980) is applied. Their method involves convolving an image with a receptive field that is a difference of two Gaussian distributions of unequal variance (a difference of Gaussians, or DOG, mask). Marr and Hildreth used the zero crossings of the convolution output. We, instead, threshold the output, yielding the most negative-going convolution outputs as black points in the output image (thus drawing cartoon lines on the dark side of edges in the original image). Lastly, we apply a transformation that deletes small unconnected regions in the image.

We have been coding cartoon images such as this in a variety of ways. One promising coding scheme is the quadtree method. In this method, an image is transmitted as a tree structure in which each node in the tree represents a square subarea of the image. At each level of the tree, if the image area corresponding to a given node is uniform [all white (W) or all black (B)], then that node becomes a leaf of the tree and is labeled with the color of that image region. Otherwise, the node is labeled G, for gray, and given four sons that correspond to the four quadrants of that image area. Thus, areas of the image are recursively cut until uniform regions–single pixels, if necessary–are reached. As an encoding method, quadtrees transmit a binary representation of the tree we have described, and can be quite efficient (Cohen, Landy, & Pavel, 1983). Figure 8b illustrates, in image form, the cuts made by a quadtree algorithm on the cartoon image of Figure 8a. It is generated with the following HIPS pipeline:

```
binquad 32 1 -g < cartoon | binquad_r -d 20 200 | \
    shiftpix -1 | pad 50 393 265 5 5 | wframe
```

This applies the *binquad* program with the *–g* option, indicating that it should actually generate the binary hierarchical code. Then the binquad receiver program *binquad_r* decodes the encoded image, creating a displayable (*–d*) version of the cuts. This is then placed in a border using *pad* and sent to the image processor using *wframe*. The gray outlines designate cuts that the encoding tree make in the image. In large, uniform areas, fewer cuts are made to the image than in textured areas, and consequently the encoding tree has fewer nodes than there are pixels. Hierarchical quadtree codes are among the most efficient codes available for these cartoon images (Cohen et al., 1983).

General Signal Processing

Testing Models

The image-processing software may be effectively used in the analysis of psychophysical models. For example, if a model refers to a number of spatial frequency channels being applied to a given image, with various sorts of noise added at each stage, and some form of linking hypothesis, then the user can apply these operations directly to digital images. Thus, a picture of a psychophysical stimulus may be digitized by the image processor, or the stimulus may be synthesized directly by HIPS. This image is subjected to various linear and nonlinear transformations, and is further corrupted by noise at any stage. Stimuli can be presented to humans

for psychophysical testing and, concurrently, be further processed by HIPS for statistical evaluation of models. As psychophysical models become increasingly complex and mathematically intractable, it is our feeling that research methods involving simulation such as we have outlined will become increasingly useful.

1-Dimensional Signals

As we have seen in several of the preceding examples, HIPS is flexible with respect to image parameters such as image size. This flexibility allows the user to use HIPS for 1-dimensional as well as 2-dimensional signal processing. An image with only one row but a large number of columns is, in effect, a 1-dimensional signal. Any of the image transforms may be used with such an image, including filtering, convolution, scaling, stretching, thresholding, encoding, and so forth. Thus, without modification, HIPS can process 1-dimensional signals.

One-dimensional signals occur in a number of psychological domains, particularly audition, in which signal processing has a long history. Another domain of 1-dimensional signals is physiological measures such as evoked potentials, blood pressure, measures of eye position, etc. Other continuous 1-dimensional signals occur in measurements of pursuit and tracking (by hand or eye) and in other dynamic measures of motor performance. Discrete 1-dimensional signals, such as recording of heartbeats, so-called cumulative records of animal responding under various reinforcement contingencies, etc., are also amenable to HIPS. HIPS may be used for signal averaging and smoothing, correlating individual recordings with the average, detecting peaks and troughs by convolution, and so forth. A large number of specialized, digital, signal-processing tools already exist for 1-dimensional signals, and it is therefore not likely that a general image-processing system like HIPS can outperform any particular specialized system. But there is much convenience for the user in learning and maintaining only one system, and it is still noteworthy that HIPS can have potential applications in such a wide variety of domains.

CONCLUSIONS

HIPS is a software image-processing system with a modular design. It has proved to be highly flexible, powerful, and easy to use. Although it was written with a particular application in mind, we have demonstrated its potential utility in a wide range of psychological research.

REFERENCES

Abdou, I. (1978). *Methods of edge detection* (Tech. Rep. No. 830). University of Southern California, Image Processing Institute Report.

Anstis, S. M. (1970). Phi movement as a subtraction process. *Vision Research*, **10**, 1411-1430.

Attneave, F. (1968). Triangles as ambiguous figures. *American Journal of Psychology*, **81**, 447-453.

Baker, C. L., & Braddick, O. J. (1982). Does segregation of differently moving areas depend on relative or absolute displacement? *Vision Research*, **22**, 851-856.

Bourne, R. S. (1978). An introduction to the UNIX shell. *Bell System Technical Journal*, **57**, 2797-2822.

Braddick, O. J. (1974). A short-range process in apparent motion. *Vision Research*, **14**, 519-527.

Carlson, C. R., Anderson, C. H., & Moeller, J. R. (1980). Visual illusions without low spatial frequencies. *Investigative Ophthalmology & Visual Science*, **19**, 165-166.

Campbell, F. W., & Robson, J. G. (1968). Application of Fourier analysis to the visibility of gratings. *Journal of Physiology*, **197**, 551-566.

Cohen, Y., Landy, M. S., & Pavel, M. (1983). Hierarchical coding of binary images. *Mathematical Studies in Perception and Cognition*, **83-8**, New York University, Department of Psychology.

De Lange, H. (1952). Relationship between critical flicker-frequency and a set of low frequency characteristics of the eye. *Journal of the Optical Society of America*, **44**, 380-389.

Ginsburg, A. (1978). *Visual information processing based on spatial filters constrained by biological data.* Doctoral dissertation, University of Cambridge, Cambridge, England.

Gonzalez, R. C., & Wintz, P. (1977). *Digital image processing.* Reading, MA: Addison-Wesley.

Graham, N. (1981). Psychophysics of spatial frequency channels. In *Perceptual organization* (pp. 1-30). M. Kubovy & J. Pomerantz (Eds.), Hillsdale, NJ: Erlbaum.

Graham, N., & Nachmias, J. (1971). Detection of grating patterns containing two spatial frequencies: A comparison of single-channel and multiple-channel models. *Vision Research*, **11**, 251-259.

Huffman, D. A. (1952). A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers*, **40**, 1098-1101.

Ives, H. E. (1922). A theory of intermittent vision. *Journal of the Optical Society of America*, **6**, 343-361.

Janez, L. (1983). *Orientation of the visual reference frame: Quantitative theory.* Paper presented at the 16th annual Mathematical Psychology meeting, Boulder, CO.

Joy, W. N. (1980). An introduction to the C shell. In *The Unix programmer's manual* (7th ed., Vol. 2c, p. 46). Berkeley, CA: Department of Electrical Engineering and Computer Science, University of California at Berkeley.

Julesz, B. (1971). *Foundations of cyclopean perception.* Chicago: University of Chicago Press.

Kaufman, L., & Pitblado, C. B. (1965). Further observations on the nature of effective binocular disparities. *American Journal of Psychology*, **78**, 379-391.

Kernighan, B. W., & Mashey, J. R. (1981, April). The UNIX programming environment. *Computer*, pp. 12-22.

Kernighan, B. W., & Ritchie, D. M. (1978). *The C programming language.* Englewood Cliffs, NJ: Prentice-Hall.

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

Lester, J. M., Brenner, J. F., & Selles, W. D. (1980). Local transforms for biomedical image analysis. *Computer Graphics and Image Processing*, **13**, 17-30.

Marr, D., & Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London*, **B 207**, 187-217.

Mostafavi, H., & Sakrison, D. J. (1976). Structure and properties of a single channel in the human visual system. *Vision Research*, **16**, 957-968.

Nakayama, K. (1981). Differential motion hyperacuity under conditions of common image motion. *Vision Research*, **21**, 1475-1482.

Palmer, S. E., & Bucher, N. M. (1981). Configural effects in perceived pointing of ambiguous triangles. *Journal of Experimental Psychology*, **7**, 88-114.

PALMER, S. E., & BUCHER, N. M. (1982). Textural effects in perceived pointing of ambiguous triangles. *Journal of Experimental Psychology*, **8**, 693-708.

PAVEL, M., SPERLING, G., RIEDL, T., & VANDERBEEK, A. (1984). *The effect of signal-to-noise ratio on the perception of American Sign Language.* Manuscript in preparation.

PAVLIDIS, T. (1982). *Algorithms for graphics and image processing.* Rockville, MD: Computer Science Press.

PELLI, D. G. (1981). *Effects of visual noise.* Doctoral dissertation, Cambridge University, Cambridge, England.

PERLMAN, G. (1980). Data analysis programs for the Unix operating system. *Behavior Research Methods & Instrumentation*, **12**, 554-558.

PERLMAN, G. (1981). An example of cooperating compact data analysis programs. *Behavior Research Methods & Instrumentation*, **13**, 290-293.

REINGOLD, E. M., NIEVERGELT, J., & DEO, N. (1977). Combinatorial algorithms: Theory and practice. Englewood Cliffs, NJ: Prentice-Hall.

RIEDL, T. (1984). HIPS in action: Application of HIPS in research. *Behavior Research Methods, Instruments, & Computers*, **16**, 217-222.

RITCHIE, D. M., & THOMPSON, K. (1978). The UNIX time-sharing system. *Bell System Technical Journal*, **57**, 1905-1929.

ROBSON, J. G. (1966). Spatial and temporal contrast-sensitivity functions of the visual system. *Journal of the Optical Society of America*, **56**, 1141-1142.

ROSENFELD, A., & KAK, A. C. (1976). Digital picture processing. New York: Academic Press.

SCHADE, O. H. (1956). Optical and photoelectric analog of the eye. *Journal of the Optical Society of America*, **46**, 721-739.

SCHWARTZ, B., & SPERLING, G. (1983). Luminance controls the perceived 3-D structure of dynamic 2-D displays. *Bulletin of the Psychonomic Society*, **21**, 456-458.

SHAW, G. B. (1979). Local and regional edge detectors: Some comparisons. *Computer Graphics and Image Processing*, **9**, 135-149.

SPERLING, G. (1964). Linear theory and the psychophysics of flicker. *Documenta Ophthalmologica*, **18**, 3-15.

SPERLING, G. (1976). Movement perception in computer-driven visual displays. *Behavior Research Methods & Instrumentation*, **8**, 144-151.

SPERLING, G. (1980). Bandwidth requirements for video transmission of American Sign Language and finger spelling. *Science*, **210**, 797-799.

SPERLING, G. (1981). Video transmission of American Sign Language and finger spelling: Present and projected bandwidth requirements. *IEEE Transactions on Communications*, **COM-29**, 1993-2002.

SPERLING, G., PAVEL, M., COHEN, Y., LANDY, M., & SCHWARTZ, B. J. (1983). Image processing in perception and cognition. In O. J. Braddick & A. C. Sleigh (Eds.), *Physical and biological processing of images* (pp. 359-378). Berlin: Springer-Verlag.

STEVENS, W. R., & HUNT, B. R. (1982). Software pipelines in image processing. *Computer Graphics and Image Processing*, **20**, 90-95.

THOMPSON, K. (1975). The UNIX command language. In *Structured programming—Infotech state of the art report* (pp. 375-384). Berkshire, England: Infotech International, Ltd.

WILSON, H., & BERGEN, J. (1979). A four mechanism model for threshold spatial vision. *Vision Research*, **19**, 19-32.

Appendix

A List of the Programs in HIPS Mentioned in This Article (for a More Complete Listing, See Landy, Cohen, & Sperling, 1984)

Program	Synopsis
abdou	Edge-fitting method (Abdou, 1978).
adddesc	Descriptive information is added to an image header.
addseq	Add two image sequences, point by point.
ahc3	Adaptive hierarchical code, transmitter.
ahc3_r	Adaptive hierarchical code, receiver
atob	Convert ASCII to byte format.
atof	Convert ASCII to floating-point format.
autodiff	Compute differences between successive frames.
bandpass	Parameterized bandpass filtering.
bclean	Binary image cleaning.
binquad	Binquad/quadtree coding, transmitter.
binquad_r	Binquad/quadtree coding, receiver.
bmovie	Display binary sequences.
bpack	Pack a sequence in one-bit-per-pixel, bit-packed format.
btof	Convert from byte to floating-point format.
bunpack	Unpack from bit-packed to byte format.
calcpix	Create a new image filter.
catframes	Concatenate separately stored frames into a sequence.
checkers	Generate a checkerboard pattern.
discedge	Edge-fitting method (Shaw, 1979).
disphist	Display histograms.
dog	Apply a Gaussian filter or a difference of Gaussian filters.
dpcm_r	Dpcm encoding, receiver.
dpcm_t	Dpcm encoding, transmitter.
enlarge	Enlarge an image.
extract	Crop an image.
extremum	A nonlinear filter for edge sharpening.
fcalcpix	Create a new floating-point image filter.
fgenframe	Generate a homogeneous floating-point image.
fourtr	Fourier transform.
fourtr3d	Fourier transform in 3 dimensions (including time).
framevar	Compute gray-level mean and variance.
ftoc	Convert floating to complex.

gcube	Generate a vector plot of a cube.
genframe	Generate a homogeneous byte-formatted image.
gmag	Globally scale a vector plot.
gnoise	Gaussian noise generation.
gpoly	Generate a vector plot of a polygon.
gshift	Globally translate a vector plot.
gsync	Synchronize and combine two vector plots.
hc_bin	Binary tree coding, transmitter.
hc_bin_r	Binary tree coding, receiver.
highpass	Parameterized high-pass filtering.
histo	Gray-level histogram computation.
inv.fourtr	Inverse Fourier transform.
inv.walshtr	Inverse Walsh transform.
logimg	Natural log of an image.
lowpass	Parameterized low-pass filtering.
mask	Image convolution.
median	A nonlinear filter for image smoothing.
movie	Display an image sequence in real time.
mulseq	Multiply a sequence by a fixed frame.
neg	Negate an image.
noise	Add bit-reversal noise.
pad	Frame an image with a homogeneous background.
pixentropy	Computer the entropy of an image.
pixto3d	Convert a byte image to vector plot format.
plot3topix	Convert a vector plot image to byte format.
powerpix	Stretch image contrast with a power function.
ptoa	Convert from any pixel format to ASCII.
reduce	Reduce image size using pixel averaging.
reflect	Reflect an image about the vertical.
repframe	Frame repetition.
rframe	Read a frame from the image processor.
rotate180	Rotate an image by 180 deg.
scale	Apply linear or second-order scaling.
seeheader	Print out image header information in a readable format.
shiftpix	Binary shift pixel values.
stretchpix	Stretch pixel contrast.
strobe	Collapse a sequence to a single frame by averaging.
subseq	Extract a subsequence.
thin	Thin out a line drawing to one pixel wide.
thicken	Thicken a thinned line drawing.
thresh	Threshold an image.
trot	Rotate a vector plot over time.
tshift	Translate a vector plot over time.
view	Compute polar perspective.
vrot	Rotate the viewer coordinates in a vector plot over time.
vshift	Translate the viewer coordinates in a vector plot over time.
walshtr	Walsh transform.
wframe	Write a frame on the image processor.