A Portable Image Processing System for Computer Vision

85-9

William Havens

Laboratory for Computational Vision
Department of Computer Science
University of British Columbia
Vancouver, British Columbia
Canada V6T 1W5

Technical Report 85-9

Keywords: image processing, computer vision, graphics programming, software engineering

Abstract

Computer vision research is flourishing although its growth has been hindered by the lack of good image processing systems. Existing systems are neither general nor portable despite various attempts at establishing standard image representations and software. Issues of hardware architecture and processing efficiency have frequently dominated system design. Often standard representations are primarily data formats for exchanging data among researchers working at affiliated laboratories using similar equipment. We argue that generality, portability and extensibility are the important criteria for developing image processing systems. The system described here, called *PIPS*, is based on these principles. An abstract image datatype is defined which is capable of representing a wide variety of imagery. The representation makes few assumptions about the spatial resolution, intensity resolution, or type of information contained in the image. A simple set of primitive operations are defined for direct and sequential access of images. These primitives are based on a bit stream access method that considers files and devices to be a long contiguous stream of bits that can be randomly read and written. Bit streams allow the word boundaries and file system architecture of the host computer system to be completely ignored and require only standard byte-wide direct-access I/O support. The standard image representation has encouraged the development of a library of portable generic image operators. These operators support interactive experimentation and make it easy to combine existing functions into new more complex operations. Finally, graphics device interfaces are defined in order to isolate graphics hardware from image processing algorithms. The system has been implemented under the Unix† operating system.

† Unix is a Trademark of AT&T Bell Laboratories.

1. Introduction

Computer vision is an important and growing discipline within Artificial Intelligence. It is central to robotics, automated manufacturing and inspection, remote sensing and other applications requiring the automatic computer interpretation of digitally-sampled spatial information. When the input information has a regular two-dimensional organization of picture elements (*pixels*), its representation is called an *image*. A considerable amount of programming effort in computer vision research is concerned with images and image processing. Datatypes must be defined for various types of images (such as rectangular gray-scale and colour imagery, binary masks, real, complex and vector-valued images and digital terrain models) and for images having varying *spatial resolution* (number of rows and columns) and different *intensity resolution* (number of bits/pixel). Algorithms must be developed that allow the interactive manipulation of image datatypes. Typical operations include simple transformations (such as arithmetic, windowing, scaling, displaying, overlaying and concatenating) as well as more sophisticated operations (such as convolution, fast fourier transform, edge detection, classification and stereo projection).

Hardware and software for image processing are under rapid development in both academic and industrial research laboratories. Unfortunately, much of the programming effort is wasted. The code produced is tailored to the current experimental task and must take into consideration the constraints of the particular graphics devices, computer and operating system being used. Subsequent experiments (in the same laboratory and elsewhere) will be able to use surprisingly little of the existing software without modification. To overcome this dilemma, a number of image representations have been

developed, including [Dehne,77], [Havens,82], [Horn,77], [Kirby,79], [Landy,84], [McKeown,77], [Quam,84], [Selfridge,79], [Sproul,76] and [Tamura,80]. Significantly, the sheer number of standards testifies to the existence of no standard at all. There is good reason for this proliferation. Typically an image processing system is developed with a particular type of application and/or computing system in mind. The voluminous quantities of data[1] and the intensive amount of computation required have made efficiency in image representation and calculation paramount. Portability, generality and extensibility have been sacrificed. In research environments, however, these attributes for software tools are far more important than absolute computational efficiency.

Based on this principle, we have developed a Portable Image Processing System (*PIPS*) which provides the following capabilites:

(1) An abstract *image datatype* is defined which is capable of representing a wide variety of imagery. An image consists of its sampled data, a set of image parameters, plus arbitrary user-defined parameters and documentation. The representation makes few assumptions about the spatial resolution, intensity resolution, or type of information contained in an image. Word and byte boundaries of the host computer architecture are ignored. Image data is stored packed at its specified resolution which facilitates both efficient storage and transmission of images.

(2) A standard set of *image primitives* is defined for direct and sequential access of images. These primitives are based on a subsystem, called *bitio*, [Havens,82] which implements a *bit stream* access method for images. Files and devices are considered

[1] For example, a single Landsat MSS image contains in excess of 57 million bits of information.

to consist of a stream of contiguous bits which can be randomly read and written. Bitio can be easily ported to most modern computing environments, requiring only standard byte-wide direct-access I/O support from the host operating system.

(3) A library of high-level generic *image operators* is being collected. These are useful routines for interactive image manipulation which are coded in the C programming language [Kernighan, 78] and assume the existence of a command-level macro facility in the host for composing new operators from existing ones.[2] Image operators encourage experimentation by making it easy to try various combinations of operations already existing in the library.

(4) The notion of *virtual devices* is introduced to isolate the peculiarities of graphics hardware from the design of image processing algorithms. A virtual device is a software interface to a particular graphics device which accepts standard images as input and/or produce standard images as output. Each virtual device is responsible for mapping the parameters of a given image to the capabilities of its actual physical device, thereby removing that tedium from the programmer. For example, a virtual device for a medium resolution graphics display can expand or reduce the image as necessary to fill the graphics screen. Virtual devices are implemented as user-level programs and therefore require only those I/O capabilities for devices described in 2) above.

An overview of the PIPS system is shown in Figure 1 which indicates the major components in schematic form. In the next section, the design of the system will be dis-

[2] Most modern systems provide such capability. The shell scripts and pipes facilities in Unix are par-

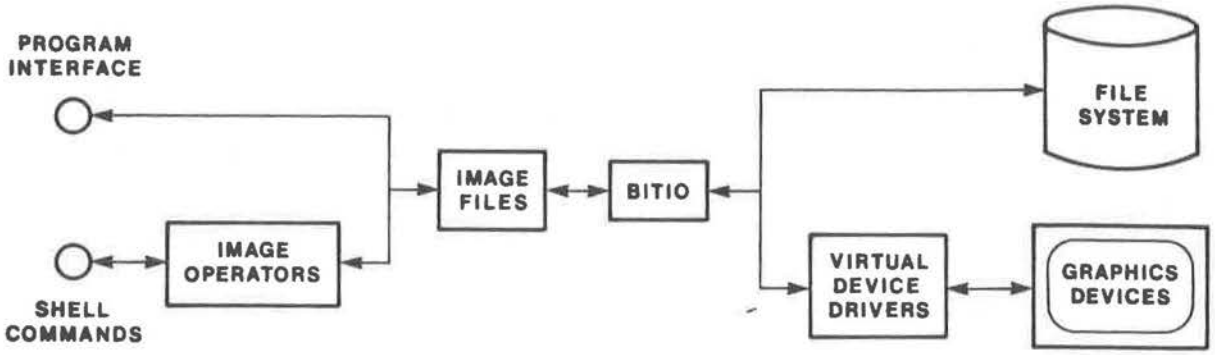

Figure 1: System Overview

cussed.

2. Design Criteria

It is interesting to examine image representations and image processing environments from the criteria of generality, portability and extensibility.

2.1. Generality

Images encompass a wide variety of two-dimensional spatial information including digitally sampled photographs, the output of multispectral scanners (e.g. Landsat imagery), video frames, digitized maps, and graphic display files. There are many possible ways to represent images [Ballard,82] [Shapiro,78] including arrays, vectors, strings, graphs, and pyramids [Tanimoto,80]. Conceptually, an image is a two-dimensional

---

ticularly convenient.

array of pixels which represent regularly sampled values of the image intensity function, $I(x,y)$, for discrete points, (x,y), in the image plane. The value of $I(x,y)$ can encode variously image brightness, colour, depth, surface orientation, or any other image property [Barrow,78]. The two-dimensional pixel array conveniently corresponds to rectangular, triangular and hexagonal tessellations of the image plane.

A standard image representation should be general and make no assumptions about the spatial resolution, intensity resolution, type, or application of the data contained in the image. The spatial resolution of imagery varies widely in both the horizontal and vertical dimensions. For images derived from photographic media, the spatial resolution depends on the photograph's dimensions and the scanning resolution of the image digitizer. For video imagery, the resolution depends on the number of horizontal scan lines in a video frame and the aspect ratio of the video format.[3] As well, many image processing operations reduce and expand the spatial resolution of images.

The intensity resolution of different images also varies considerably. Intensity resolution is often expressed as the number of *bits/pixel* necessary to represent the pixel values stored in the image. This number is a function of the pixel datatype and the dynamic range of possible pixel values. For example, a gray-scale image digitized from a black and white photograph or video frame typically contains a pixel array of unsigned integers whose values represent image brightness. The number of bits required per pixel varies according to the sampling precision of the digitizing hardware. Colour imagery is represented in a similar fashion except that each pixel encodes a vector of three colour

[3] For example, the NTSC format has 525 horizontal lines and an aspect ratio of 4:3 which yields a vertical resolution of 700 pixels.

values. Binary images have only two possible intensity values and are represented by a single bit of intensity information. Other types of imagery may have real or complex-valued pixels. For instance, a *digital terrain model* is a synthetic image whose pixels represent terrain altitude typically as floating-point numbers. As well, complex images are frequently produced in the Fourier analysis of imagery. Each pixel contains a pair of values representing the real and imaginary (or alternatively the magnitude and phase) components of the image at that point. Finally, one-dimensional images are also important. Colour transformation tables, histograms, plot vectors and graphics display files are all conveniently encoded as one-dimensional images. An adequate image datatype must be able to represent all of these common types of imagery using the same data structures and the same set of primitive operations.

Image processing software must also be general. Computer vision experiments require interactive access to sophisticated image processing algorithms. These algorithms are complex, tedious to program and often implemented for a particular application even though many image processing operations are common to a variety of applications. What is needed are generic image operators which can process images having arbitrary spatial resolution, intensity resolution or pixel datatype. The accepted solution to this problem is to associate the parameters of an image with its data representation in a header record at the beginning of the image file. Image operators can then be coded which fetch these parameters from input images and store them in output images. By so doing, a library of useful and general purpose image processing algorithms can be developed.

Most existing image representations satisfy the above requirements. Probably the best known image representation is the NATO standard [Dehne,77] which has been extended at various sites. This standard was developed as part of the distributed Image Understanding (IU) Testbed facility at a number of computer vision laboratories sponsored by DARPA.[4] The image representation chosen contains three parts: a header for storing image parameters, the pixel array, and an index table for efficiently mapping image coordinates into linear memory references. The representation provides a storage format and method of access for large images that can be used in different programming languages on multiple machines. Images are manipulated using a large number of primitive operations which provide efficient access to pixel data by image coordinates. Other related representations include the RIFF system [Selfridge,79] and a predecessor AIS [Sproul,76]. See also [Kirby,79].

An alternate approach is to build a general and powerful image processing environment for one particular computer and display. This methodology is illustrated by the ImagCalc* system of Quam[84] developed for the Symbolics 3600 Lisp machine. This machine provides a single-user Lisp programming environment with a very high resolution black and white or colour display. ImagCalc adds a general purpose image processing environment that allows the display screen to be divided into image panes. Each pane is associated with a stack of images and displays the image currently on top of its stack. There are a sizeable number of image operators which access these images, compute new images and manipulate the stacks. The environment is very interactive and

---

[4] U.S. Defense Advanced Research Projects Agency.

* Trademark of SRI International.

relies heavily on its extensible menu of image operators. No particular image representation is assumed. Instead, the system can process images encoded in a number of existing image formats.

A third approach, which is compatible with our work, is the HIPS system [Landy,84]. The system provides a flexible general purpose image processing environment under the Unix operating system. A standard image representation is defined which consists of a header record and the pixel data. The header specifies the number of frames, rows, columns and bits/pixel contained in the image plus the pixel datatype and an indicator for pixels which are not multiples or sub-multiples of 8-bit bytes. There is also annotation information in the header which is automatically updated by the image software. This is a nice feature for keeping track of the genealogy of images. The system provides a large number of image operators and peripheral interfaces for a number of graphics and video devices.

2.2. Portability

To be truly portable, an image processing system should be as machine independent as possible, defining a small number of primitive operations for manipulating image data. These image primitives should make as few assumptions as possible about the wordsize, I/O functions or file system of any target host and be accessible from high-level programming languages available on the host. To facilitate portability, they should be implemented using only rudimentary I/O operations. Most operating systems provide both sequential and random access reading and writing of files for byte-oriented data. Once the primitives have been ported successfully to a new host system, all of the

existing library of image operators that are coded in a programming language resident on the host automatically become available.

Portability has not been a strong feature of image processing systems which have been developed with existing computing and graphics hardware in mind. Obtaining the most flexibility and performance from this equipment has taken precedence. For example, one of the goals of the ARPA IU Testbed was the portability between sites of both image databases and image processing algorithms. Unfortunately, each site has a substantial investment in its local hardware and software configuration. Consequently, the ARPA standard has served primarily as a transport format for interchange of imagery from one site to another. The different systems are too diverse for extensive sharing of image software. Instead, the approach has been to distribute processing over the ARPANET using remote procedure calls and image transfers.

A second requirement for portability is the removal of dependencies on particular graphics devices from image programming. Software should make no assumptions about which graphics devices are being used for image input or display. All of the information pertinent to the image should be specified by its image parameters. For example, HIPS addresses this requirement by constructing software peripheral interfaces between image processing software and each graphics device. All I/O to a device must pass through its interface which is responsible for mapping between the parameters of a particular image and the architectural constraints of the actual device. The ARPA system follows a different solution. An ideal generic device is assumed and all I/O is made to this device. Peripheral interfaces are then defined which map the generic properties of the ideal device to the actual parameters of each real device.

2.3. Extensibility

Computer vision is an experimental science. New experimental techniques must not require extensive programming effort before they can be tried. The ability to quickly forge new image processing tools from an existing library of useful image operators encourages experimentation. The HIPS system exploits the command macros and multiprocessing facilities provided by the Unix operating system to achieve this capability. Many image processing operations act as filters, accepting a single image as input and producing another as output. By implementing image operators as Unix processes, the output of one operator can be directly connected to the input of another (called a *pipe*) without the need for temporary intermediate files. New operators can be composed by collecting sequences of these operators in a macro file (called a *script*). PIPS relies on these same elegant mechanisms although the macro facilities of other operating systems can also suffice.

A very powerful approach to extensibility can be used with image processing enviroments for which the system command language is also an interactive programming language. For example, the ImagCalc system allows new image operators to be easily coded as new Lisp functions. The image processing system thereby inherits the considerable capabilities of the Lisp programming environment.[5]

Another aspect of extensibility is the ability to augment the information contained in the image representation. HIPS allows arbitrary annotation data in the image header. Other systems have viewed image processing as part of the larger task of image

[5] An example of Lisp-based processing is given in Section 3.2.

database construction and manipulation [McKeown,77] [Tamura,80]. For example, the RIFF system uses an image header that includes an arbitrary number of binary relations, implemented as name/value pairs. Each pair can record relational information about the image such as statistical measures, the parent image of this image and the window coordinates of this image in its parent. The result is a linked database of imagery.

3. Standard Images

PIPS defines the following representation for image data. The representation is straightforward and simple thereby supporting the goals of generality and portability. In the rectangular pixel array of Figure 2, the number of horizontal rows in the image (called *nrows*) and the number of vertical columns in the image (called *ncols*) specify the vertical and horizontal spatial resolution of the image respectively. Individual pixels in the image are referenced by their discrete spatial coordinates, (x,y), in the array, $0 \leq x < nrows$, $0 \leq y < ncols$. Conventionally, the origin of the image coordinate system is the top-left-hand corner of the array. The image can also be usefully viewed as being a contiguous stream of pixels in row-major order beginning at I (0,0) and terminating with I $(nrows-1, ncols-1)$. Such an organization of the image is called a *raster* and is the format assumed by many serial graphics devices. It is desirable to be able to access images by both absolute pixel locations and as sequential streams of pixels. The third image dimension is intensity resolution expressed as the number of bits/pixel (*bpp*) needed to represent the entire range of values of I (x,y) for a given image. For efficient storage and transmission of images, *bpp* must be an independent parameter of

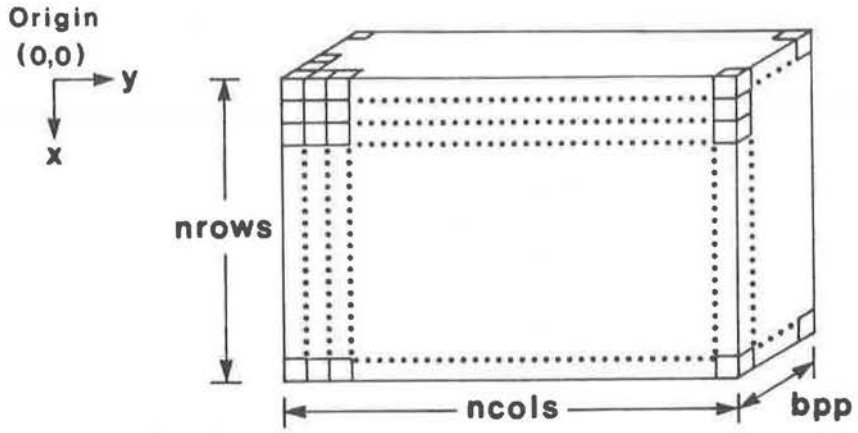

Figure 2: Image Representation

the image.

The image representation includes the two-dimensional array of packed pixel values plus an image file descriptor called the *image header*. The header contains a small set of common image parameters necessary for all images, additional parameters specific to a given image plus an annotation field for optional textual information about the image. The common image parameters are listed in Table 1. The header record was chosen to be a fixed-length field appearing at the beginning of each image file. The header is large enough to hold a considerable number of parameters and annotation yet occupies negligible space compared to the data contained in most images. More importantly, a fixed header allows updating of the image parameter information *in situ* without copying the remainder of the (possibly very large) image file. Two image operators, *iffsee* and

iffedit,[6] allow examining and modifying all image parameters.

The format of the header is a free-form sequence of characters according to the following BNF syntax:

$$
\begin{array}{lcl}
<header> & := & Imagefile: <exp>^{n} <annote> \\
<exp> & := & <parm> = <value>; \\
<parm> & := & version \mid nrows \mid ncols \mid bpp \mid type \mid positive \mid \cdots \\
<annote> & := & <string>
\end{array}
$$

The header contains an initial character sequence, *"Imagefile:"*, indicating that the file is an image followed by set of n parameter expressions, $<exp>$. Each expression

Parameter	**Interpretation**
version	Software release used to produce this image.
nrows	The number of rows contained in the image.
ncols	The number of columns in the image.
bpp	The number of bits/pixel required to represent the full range of intensity values for the image.
type	Datatype of the pixels contained in the image.
positive	Boolean indicating whether the image was produced from a photographic positive or negative source.

Table 1: Common Image Parameters

[6] Both are described in Section 3.3.

encodes the value of either a common or user-defined parameter for the image.[7] $<exp>$ consists of a parameter name, $<parm>$, followed by an equality sign, followed by the value for the parameter, $<value>$, and a delineating semicolon or space character. Appearing after the parameters is the annotation record, $<annote>$, which is terminated by the first occuring NUL character or the end of the header record.

3.1. Image Primitives

In order to make the system as portable as possible, it is necessary to minimize dependencies on any particular host operating system. To this end, we have defined a small simple set of image primitives which provide basic access to image files and devices. The primitives include opening and closing an image file associated with a disk file or graphics device; reading and/or writing of pixel values in either sequential or random order by specifying pixel address, row address, or by default raster order; and obtaining or setting of the parameters associated with the image. The host need provide only the following capabilities to support the PIPS system:

- A direct-access file system for files containing contiguous sequences of 8-bit bytes.
- System calls for opening and closing files with read, write and update modes of access.
- System calls for creating and deleting files under program control.
- Access to the C programming language compiler.

The image primitives described below are also summarized in Table 2. The function,

iopen (*fname*, *mode*),

opens an image file, named *fname*, for access mode, *mode* , specified as reading, writing or both. *iopen* returns a file descriptor to be used for subsequent references to the

[7] In the current version, $n \geq 6$ and all of the common parameters must appear in a fixed order.

Primitive	Operation
iopen/iclose	Open/close a file or graphics device for reading and/or writing.
getheader	Obtain the image parameters associated with an image file or device.
putheader	Write new image parameters to an image file or device.
iget/iset	Gets/sets a specified parameter for an image file.
getpix/putpix	Read/write the next pixel in an image file in raster order.
getrow/putrow	Read/write an entire row of pixels in the image or device.
iseek	Set the pixel stream pointer to the specified row and column.
isync	Synchronize the image with its open file or device.
ieof/ierror	Returns true if an end-of-file/error condition has occurred, else false.

Table: 2: PIPS Image Primitives

image. Its counterpart function,

iclose (*ifp*),

closes the file associated with the image, *ifp*, after image processing has been completed. *getheader* and *putheader* retrieve and store respectively the image header information from the image file. These functions must be called before reading and/or writing pixels begins. The primitive,

getheader (*ifp*, *annote*),

retrieves the image header from file, *ifp*, and copies the annotation record into the string, *annote* . The default image parameters in *ifp* are thus overwritten by the actual

parameters of the image file, while

putheader (*ifp*, *annote*)

performs the reverse function. The image parameters contained in *ifp* and the new annotation, *annote* , are written to the file. The functions, *iget* and *iset* , are used to get and set respectively a parameter associated with an image as follows:

iget (*ifp*, *parm*)

returns the value of *parm* contained in the header for *ifp*, whereas

iset (*ifp*, *parm* , *value*)

sets the value of *parm* to its new value, *value* , in the header of *ifp*.[8]

The actual reading and writing of image data is performed by two functions:

getpix (*ifp*)

which reads the next pixel in raster order from image, *ifp*; and

putpix (*ifp*, *pix*)

which writes the value, *pix* , into the current pixel position of the image. To read more than a single pixel at a time or to read in other than raster sequence, the following primitives are provided:

getrow (*ifp*, *row* , *arr*)

reads an entire row of pixels from *ifp* into an array, *arr* , which must be the same data-type as the pixel data and have a length adequate for the number of columns in each row, *ncols* . The arg, *row* , is an integer number specifying which row to read, $0 \leq row \leq nrows$. Alternatively,

---

[8] Currently this is done by associating an index number with each common parameter in the header. A more powerful approach is to construct a symbol table in the header allowing all parameters to be accessed by name. RIFF provides such a capability.

putrow (*ifp*, *row* , *arr*)

writes an entire row of pixels contained in *arr* into image, *ifp*, beginning at row number, *row* . The procedure,

iseek (*ifp*, *row* , *col*),

is used to access pixels in the image randomly by positioning the file pointer to the pixel beginning at row number, *row*, and column number, *col*. Subsequent read and write operations will proceed in raster order from this pixel position. A related procedure,

isync (*ifp*),

is useful for I/O operations directly to graphics devices. It flushes the pixel buffer associated with *ifp* thereby synchronizing the attached device to the read and write calls of the program. Finally,

ieof (*ifp*)

returns true if the exception, *end-of-file*, has been encountered during the last read on *ifp* and

ierror (*ifp*)

returns true if an I/O error has occurred for *ifp*. Otherwise, both functions return false. For more detailed documentation of these functions, see [Havens,1982].

3.2. An Interactive Programming Example

The primitives defined above help simplify image processing software development. Much of our own experimental computer vision work is interactive and the Lisp language is often the programming environment of choice. For example, Figure 3 lists a protocol from an interactive session using the Franzlisp [Foderaro,1980] interpreter.[9]

[9] Figures 3 and 4 through 9 appear at the end of the paper.

The protocol begins by invoking the Lisp interpreter and loading the PIPS interface to Lisp, called *lispimage* (in steps 1-3). Lines prefixed with the prompt, "->", are typed by the user. All others are output from the interpreter.

In the first part of the example, it is desired to process an image file, *claire*, by reducing its intensity resolution from 8-bits to 3-bits while leaving the resolution in both spatial dimensions unaffected. This image is shown in Figure 4. *iopen* is used to open the image for reading and the resulting image file descriptor is assigned to a variable, *ifp1*, (in steps 3 and 4). Next, the image header information is retrieved using the primitive, *getheader*, and its parameters stored in the file descriptor (step 6). *getheader* returns the annotation record as a character string (in step 7). The values of the parameters, *nrows*, *ncols* and *bpp* are retrieved from the file descriptor using the function, *iget*, (in steps 8-13). A new image, called *claire.3b*, is then created to receive the output image and its file descriptor assigned to a variable, *ifp2*, (in steps 14 and 15). The number of rows and columns for *ifp2* will be identical to *ifp1* but with the parameter, *bpp*, equal to 3-bits. All the parameters are set for the output image (in steps 16-21) and the image header is written out to the file (steps 22 and 23).

The system is now ready to process the pixel data which is performed by the single Lisp *while* expression (in step 24). The pixels are read in raster order from *ifp1*, divided by the scale factor, $32 = 2^{8-3}$, and written to *ifp2*. This loop terminates when an end-of-file is detected by the function, *ieof*, on *ifp1*. Finally, both images are closed using *iclose* (steps 26-29). The result of this scaling operation can be seen in Figure 5 where the reduction in intensity resolution is evident.

Image primitives can also, of course, be incorporated into compiled and interpreted programs. Again using Lisp as an example programming language, the remainder of the protocol of Figure 3 illustrates the definition and application of a function,

slice (*image* 1, *image* 2 ,*low* , *high*)

which reads an input image, *image* 1, of any spatial or intensity resolution and produces a binary image, *image* 2, as output. Each pixel is *image* 2 is set to 1 if its corresponding pixel in *image* 1 has an intensity value between *low* and *high* . Otherwise, the output pixel is set to 0. In steps 47 and 48, the function is applied to the image, *claire* , with *low* = 140 and *high* = 200. The computed image, *claire.bin* , is shown in Figure 6.

3.3. Bit Stream I/O

Images can be very large data objects. To transmit and store images efficiently in the host file system has necessitated implementing the bitio input/output method for packed bit streams. Bitio is similar to normal character stream input and output except that files are considered to be a randomly accessible contiguous stream of bits. Byte boundaries are completely ignored and input/output operations to the host file system are fully buffered.

The bitio primitives are listed in Table 3. Strings of bits can be read and/or written either sequentially or by absolute bit position in the bit stream. The number of bits that can be read or written in the stream is limited by the largest wordsize, *wsize* , on which the host computer can perform unsigned integer arithemetic.[10] The maximum size

[10] In the Vax/Unix implementation, bit strings are limited to *wsize* = 32 bits maximum length. The largest value of *bpp* for an image file is also this value.

Primitive	Operation
bopen/bclose	Open/close a bit stream with a file or device for reading and/or writing.
bget	Read a string of bits from a bit stream into a long word.
bput	Write a string of bits from a long word into a bit stream.
bseek	Move bit stream pointer to specified bit position.
bsync	Synchronize I/O operations with the open file or device.

Table: 3: Bitio Primitives

of a bit stream file is 2^{wsize} bits.[11] The bitio primitive are an independent subsystem of standard images and are potentially useful for other purposes. Most of the image primitives simply call the bitio primitives directly translating pixel coordinates into absolute bit positions where necessary.

3.4. Image Operators

Many high level operations on image datatypes are common to a wide variety of computer vision, graphics, and image processing applications. These operators typically manipulate whole images as a unit, frequently producing new images as their results. The library of image operators currently available in the PIPS system is listed in Table 4.[12] Most of these routines have been contributed by the user community and

[11] The maximum size of an image file is $nrows*ncols*bpp \leq 2^{wsize}$, which is more than large enough for practical purposes. For example, an 8-bit Vax/Unix image may have upto 23170 rows and columns.

[12] Table 4 appears at the end of the paper.

satisfy our design goals of being both generic and portable. By enforcing a flexible standard representation for images, the class of images for which a particular operator is applicable can be made very general. Likewise, by defining a simple set of portable image primitives and requiring that all image operators be coded in a widely accessible high-level programming language (like C or Pascal), the operators can also be made portable.

The library can roughly be divided into three groups of operators. Basic operators perform simple and frequently used graphics operations on images. For examples, the *window* [13] operator accepts as input an image of arbitrary spatial and intensity resolution and unknown pixel type. By specifying the row and column coordinates of a rectangle within the input image, the operator produces a new output image for that rectangle. The new image inherits the *bpp* parameter and annotation record of its parent. The operator, *iscale* , performs both spatial and intensity scaling on an image. The size of the image can be independently scaled in both the x and y directions by an arbitrary scale factor using a variety of different algorithms (for example, a four-point cubic convolution). At the same time, the intensity resolution can also be scaled either increasing or decreasing the value of *bpp* for the image. Again a number of different scaling methods are available including *dithering* to maintain the original tonal range of the image when reducing *bpp* . Finally, general arithmetic can be performed on images using the operator, *adjust* .[14] This operator takes a single image as input, adds/subtracts a constant from each pixel, multiplies/divides the pixel by another constant, and optionally limits the results to specified upper and lower bounds. The output

[13] *window* and *iscale* were implemented by Stewart Kingdon.

image contains the same parameters, *nrows* and *ncols*, as the input but the value of *bpp* may be adjusted to hold the maximum value produced during the arithmetic.

The second group of image operators are more specialized. These routines are usually associated with a particular image processing application. For example, *edge-detection* is an important operation in computer vision. Frequently edges in the image correspond with surface discontinuities in the underlying scene. Many edge detectors are known, but recently there has been considerable interest in a non-directional second derivative operator described by Marr and Hildreth [1980]. The operator looks for zero-crossings of the Laplacian function, ∇^2, applied to a Gaussian-smoothed gray-scale image, $I(x, y)$ as follows:

$$\nabla^2(G * I(x, y))$$

where G is the Gaussian function and "$*$" indicates two-dimensional convolution. Since both ∇^2 and G are linear operators, they can be combined into a single operator, $\nabla^2 G$, and applied directly to the image. To experiment with this technique, we have implemented an image operator, called appropriately *del2g*.[14] The operator takes as input a gray-scale intensity image digitized from a natural scene and a set of parameters for constructing the convolution mask approximating $\nabla^2 G$. The mask, which is also represented as an image, is convolved with the input image producing the convolution as a signed integer image. The locus of points in this image where adjacent pixel values pass through zero corresponds to maximum intensity changes in the gray-scale image. The performance of *del2g* is demonstrated for the input image of Figure 4 producing the convolution image, *claire.cv*, shown in Figure 7 and final zero-crossing image,

[14] Implemented by Alan Carter.

claire. $0x$, of Figure 8.

3.5. Defining New Operators

Functional composition is a powerful mechanism for defining new operators. The existing library of image operators in conjunction with a reasonable macro facility in the host operating system provides the necessary functional composition capability. Most modern operating systems have some sort of command macro facility. In our Unix implementation of PIPS, we make major use of process pipes and shell scripts. For example, the operator, *del2g*, is actually implemented as a script. All of the steps in the method described above are existing image operators. To construct *del2g*, it is only necessary to compose the individual steps into a new operator. A simplified version of this operator is illustrated in Figure 10 below. The operator is called by specifying three arguments: the size of the convolution mask, the frequency of the Gaussian filter, and the input gray-scale image. These arguments are bound to their variables, $*size*, $*sigma* and $*image* respectively (in steps 1-4). In step 5, the image primitive, *mask* is invoked to construct a convolution mask for *del2g* from the parameters, $*size* and $*sigma*. The output of this operator is piped into *convolve* (using the Unix *pipe*

```
1    # del2g size sigma image
2    set size=$1
3    set sigma=$2
4    set image=$3
5    mask -k $size -s $sigma | convolve $image >$image.cv
6    zerox $image.cv | adjust -bpp 1 -thresh 15 >$image.0x
```

Figure 10: Composition of $\nabla^2 G$ Operator

operator, "|") which applies the new mask to *$image* and produces the convolved image, *$image.cv*, as result (stored as the name of the input image suffixed by *".cv "*). In the last step (6), the zero-crossing detector, *zerox* is applied to the convolved image and produces a magnitude image as result indicating the strength of the zero-crossing. This image is thresholded to 1-bit by *adjust* and the final output is saved in a new image called *$image.0x*.

3.6. Virtual Devices

The final group of image operators are intentionally device specific. One of the objectives of this work is to divorce the manipulation of images from the hardware peculiarities of graphical devices. Our approach has been to define software interfaces, which we call virtual devices, that accept images as input and produce images as output. By interpreting the parameters of the image file directly, they free the programmer from having to conform to the physical constraints of the actual device. Ideally, the user of the system should never have to know what devices were used to produce the imagery or what type of displays are available for viewing the results.

We have developed a number of virtual devices for the hardware that we have available in our laboratory although this work is still in progress. Each device is implemented as an image operator that makes use of the normal device driver software provided by the host system. No operating system modifications for image files is necessary. However, it would be tempting to integrate virtual devices with the real devices of the host. By doing so, the file system primitives provided by the host could operate directly on images. Unfortunately, the portability of our system would seriously suffer.

There are currently virtual devices for Raster Technology 1/25 displays, for Telidon [Godfrey,1981] videotext terminals, for Imagen Imprint-10 laser printers, for an Optronics scanning densitometer/film writer, for Comtal Vision/One workstations, and for the Apple Macintosh personal computer.

For example, Telidon terminals make inexpensive graphics displays for low resolution imagery. In our system, the *tshade* [15] operator accepts standard images as input and displays them in eight possible gray levels on the screen. On the other hand, an Imagen Imprint-10 laser printer has a very high spatial resolution but can print only binary images. The operator, *ishade* [15], accepts an image file containing a bit map (represented as a single bit image), converts it to Impress† format, and submits it to the printer.

Recently, the Macintosh has become a popular and inexpensive graphics computer. It has a display of 512 columns and 342 rows with 1-bit of intensity resolution. The image operator, *maciff*, can translate in both directions between MacPaint+ documents and image files having a single bit of intensity resolution. For example, the image of Figure 9 is a MacPaint document, *claire* 1.*pntg* , created by piping the image of Figure 4 first through *iscale* to yield an intensity range of 1-bit and then into *maciff* to produce the result shown. Finally, it should be noted that the original photograph was digitized using the *photoscan* [16] operator and all the images in Figures 4 through 8 were produced

---

[15] Implemented by Stewart Kingdon.

† Impress is a trademark of Imagen Inc.

+ MacPaint is a trademark of Apple Computer Inc.

[16] Written by Bob Woodham.

using the *rtshade* [17] operator for the Raster Technology display.

4. Conclusion

A significant aspect of computer vision research is the development of laboratory computing environments for image processing. Unfortunately, research in the field has been hampered by the lack of good image processing software. Existing software systems are neither general nor portable despite various attempts at defining image standards. An image processing system must be more than an interchange format between affiliated laboratories using similar equipment.

The development of the PIPS system has followed a different approach. We identified a number of design criteria for a system intended for a research environment. These criteria have not previously been adequately addressed. First, we chose a standard representation for raster image data that is as simple as possible yet capable of representing a wide variety of types of imagery. The actual parameters for a given image are stored in the image datatype along with additional user-defined information. Operations on images can access these parameters facilitating the implementation of generic image operators.

Second, the system is portable and can easily be implemented on a wide variety of computing hosts. It makes few assumptions about the wordsize, file system or other peculiarities of any particular host. We achieved this goal by defining a small set of image primitives for accessing and manipulating images. The image primitives provide both direct and sequential access to image files and image devices. The primitives are

[17] Implemented by Marc Majka.

coded in the C programming language which is available on most modern computer systems and require only standard I/O support from the host. The bitio subsystem allows the image routines to ignore any mismatch between the parameters of a particular image and the architecture of the host computer.

Third, the system provides an extensive library of useful and powerful image operators. Once the image primitives have been ported to a new host, all of the image operators become available on that new system. The operators are, as well, extensible. They can be combined into new operators thereby extending the capabilities of the system.

Finally, hardware technology continues to provide new computing and graphics devices. PIPS provides virtual devices to isolate as much as possible image processing algorithms from the architectural constraints of these devices. The image operators and image primitives do not address graphical devices directly. Instead an operator is defined which performs the mapping between standard images and the actual physical device. Virtual devices have been implemented as image operators to preserve the portability of the system.

References

D. H. Ballard & C. M. Brown (1982) *Computer Vision,* Prentice-Hall, Englewood Cliffs, N.J.

H. G. Barrow & J.M. Tenenbaum (1978) "Recovering Intrinsic Scene Characteristics from Images", in A.R.Hanson & E.M.Riseman (eds.), *Computer Vision Systems,* Academic Press, New York, pp.3-26.

J. S. Dehne (1977) The NATO RSG-4/SGIP Tape Format, in *Proc. Workshop on Standards for Image Pattern Recogniton,* Nat. Bureau of Standards, Special Pub. 500-8, Washington, D.C.

J. K. Foderaro (1980) *The Franzlisp Manual,* Univ. of California, Berkeley, CA.

D. Godfrey & E. Chang (1981) *The Telidon Book,* Pocépic Publishing, Victoria, B.C., Canada.

W. Havens, A. Carter & S. Kingdon (1982) Standard Image Files: Generic Operations for Image Datatypes, TR-82-10, Department of Computer Science, University of British Columbia, Vancouver, Canada.

E. C. Hildreth (1980) Implementation of a Theory of Edge Detection, AI-TR-579, MIT AI Lab, Cambridge, Mass., April 1980.

B. K. P. Horn (1977) unpublished technical note, MIT AI Lab, 545 Technology Square, Cambridge, Mass.

B. W. Kernighan & D. M. Ritchie (1978) *The C Programming Language,* Prentice-Hall, Englewood Cliffs, N.J.

R. L. Kirby, R. Smith, P. Dondes, S. Ranade, L. Kitchen & F. Blonder (1979) Subroutines and Programs for Grinnell GMR-27 Display Processor, TR-810, Computer Science Dept., University of Maryland, College Park, Md.

M. S. Landy, Y. Cohen & G. Sperling (1984) HIPS: A Unix-Based Image Processing System, *Comp. Vision, Graphics & Image Processing 25,* pp. 331-347.

D. Marr & T. Poggio (1976) Cooperative Computation of Stereo Disparity, *Science 194,* 1976, pp.283-287.

D. McKeown & R. Reddy (1977) A Hierarchical Symbolic Representation for an Image Database, Proc. IEEE Workshop on Picture Data Description and Management.

P. Selfridge & K. Sloan jr. (1979) Raster Image File Format (RIFF): An Approach to Problems in Image Management, Tech. Report 61, Computer Science Dept., University of Rochester, Rochester, N.Y., Aug. 79.

R. F. Sproul & P. Baudelaire (1976) Proposed 'Array of Intensity' Sample Format, unpublished tech. doc., XEROX PARC, Palo Alto, CA., May 76.

L.G. Shapiro (1978) Data Structures for Picture Processing, *Computer Graphics,* vol 12, no. 3, August 1978, pp.140-146.

L. Quam (1984) *The Image Calc Vision System, Parts I & II,* SRI International, Menlo Park, CA.

H. Tamura (1980) Image Database Management for Pattern Information Processing Studies, in *Pictorial Information Systems,* S. Chang & K. S. Fu (eds.), Lecture Notes in Computer Science 80, Springer-Verlag, New York, pp.198-227.

S. Tanimoto & A. Klinger (1980) *Structured Computer Vision: Machine Perception through Hierarchical Computation Structures*, Academic Press, New York, 1980.

Acknowledgments

Major contributions to this system have been made by the researchers in The Laboratory for Computational Vision, in particular, Alan Carter, Stewart Kingdon, Rachel Gelbart, Jay Glicksman, Tim Lee, Jim Little, Marc Majka, Farzin Moktarian, and Bob Woodham. This work was supported by NSERC under grants A5502 and SMI-51 and by NSF grant MCS-8004882.

```
1       Franz Lisp, Opus 38.79
2       -> (load 'lispimage)
3       t
4       -> (setq ifp1 (iopen "claire" 'read))
5       619524
6       -> (getheader ifp1)
7       "date = July 1, 1985 ; Claire Elizabeth Havens"
8       -> (setq nrows (iget ifp1 'nrows))
9       450
10      -> (setq ncols (iget ifp1 'ncols))
11      535
12      -> (setq bpp (iget ifp1 'bpp))
13      8
14      -> (setq ifp2 (iopen "claire.3b" 'write))
15      619556
16      -> (iset ifp2 'nrows nrows)
17      450
18      -> (iset ifp2 'ncols ncols)
19      535
20      -> (iset ifp2 'bpp 3)
21      3
22      -> (putheader ifp2 "")
23      t
24      -> (while (not (ieof ifp1)) (putpix ifp2 (quotient (getpix ifp1) 32))
25      nil
26      -> (iclose ifp1)
27      t
28      -> (iclose ifp2)
29      t
30      -> (defun slice (image1 image2 low high)
31              (prog (ifp1 ifp2 hdr1 pixel)
32                  (setq ifp1 (iopen image1 read))
33                  (setq ifp2 (iopen image2 write))
34                  (setq hdr1 (getheader ifp1))
35                  (iset ifp2 'nrows (iget ifp1 'nrows))
36                  (iset ifp2 'ncols (iget ifp1 'ncols))
37                  (iset ifp2 'bpp 1)
38                  (putheader ifp2 (concat hdr1 ":slice " low high))
39                  (while (not (ieof ifp1))
40                      (setq pixel (getpix ifp1))
41                      (if (and (greaterp pixel low)(lessp pixel high))
42                          (putpix ifp2 1)
43                          (putpix ifp2 0)))
44                  (iclose ifp1)
45                  (iclose ifp2)))
```

```
46          slice
47          -> (slice "claire" "claire.1b" 140 200)
48          t
```

Figure 3: Interactive Programming in Lisp

Operator	*Operation*
gauss	Applies a Gaussian smoothing filter to an image producing a new image as output.
imask	Masks one image over another.
window	Extracts a rectangular subregion of an image producing a new image as output.
mask	Constructs a convolution mask represented as an image.
convolve	Performs the convolution of an input image with a mask image producing the result as a new real-valued image.
zeroz	Determines the zero-crossings in a signed integer image
del2g	Applies the Marr/Hildreth [1980] edge detection operator to an image. The output is a single bit image indicating the locus of zero-crossings of the second derivative of the smoothed input image.
image/raster	Converts to and from image file representation.
munch/zoom	Reduces/enlarges the spatial size of an image by a specified factor producing a new image as output.
iscale	Changes the number of bits per pixel in an image producing a new image as output.
transpose	Exchanges the rows and columns of an image.
iffsee	Displays the image parameters of an image including the annotation record.
iffedit	Edits the image parameters of an image file using any text editor or a simple prompting dialogue. All image parameters are stored in the image header in a textual format.
randomdot	Produces a random dot stereogram [Marr&Poggio, 1976] from a digital terrain model represented in the input image. Two output images are produced corresponding to the left and right eye views.
stereo	Produces an orthographic left and right stereo pair of images from an input intensity image and a digital terrain model image.
maciff	Translates in both directions between Macintosh Macpaint documents and

	image files having 1-bit intensity resolution.
iff-fft	Performs the fast Fourier transform on an image producing a pair of images representing the real and imagery portions of the transform.
nnc/mlc	Nearest neighbour classification / maximum likelihood classification of a number of spatial registered input images. The single output image encodes the assigned class of each pixel.
adjust	Performs general arithmetic operations on images.
splice/paste	Combines several images into one large image.
histogram	Computes histogram and cumulative histogram for an image. The output is represented as an image containing a single row.
synthetic	Produces a synthetic image of a digital terrain model viewed from an arbitrary viewing position.
transpose	Transposes the rows and columns of any image.
slice	Produces a 1-bit image representing an intensity slice of the input image.
xflip/yflip	Flips an image on either its x or y-axis.
ishade	Converts 1-bit images to Imagen Impress format and submits the results to printer spool.
tshade	Displays images on the Telidon videotext display.
rtshade/rtread/rtover/rtscroll	Controls read/writing of images to the Raster Technology display.
photoscan	Produces digitized images from photographic media on the Optronics film scanner.
photowrite	Produces photographic products from images on the Optronics film writer.

Table: 4: Synopsis of Image Operators

Figure 4: *claire* - 8-Bit Gray-Scale Image

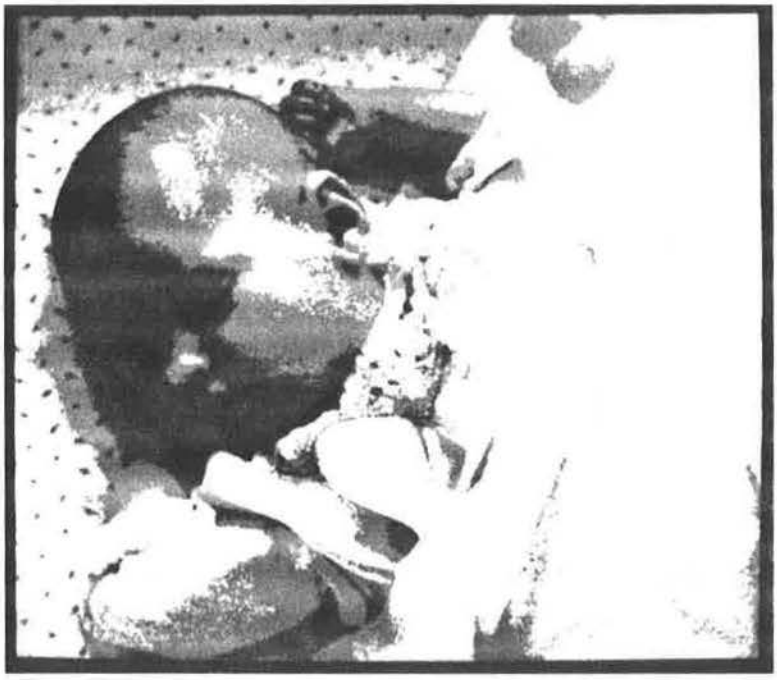

Figure 5: *claire.3b* - 3-Bit Reduced Intensity Resolution

Figure 6: *claire.bin* - Binary Image

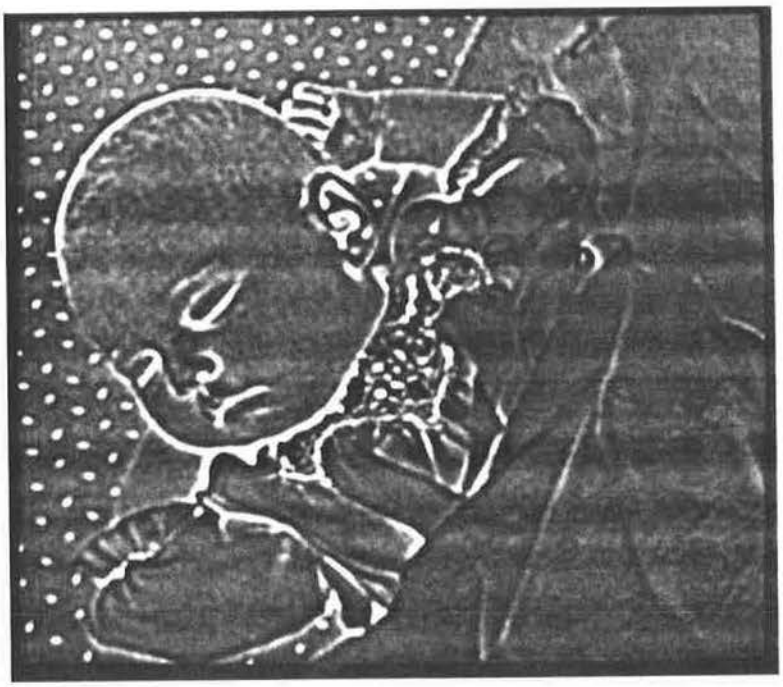

Figure 7: *claire.cv* - $\nabla^2 G$ Image

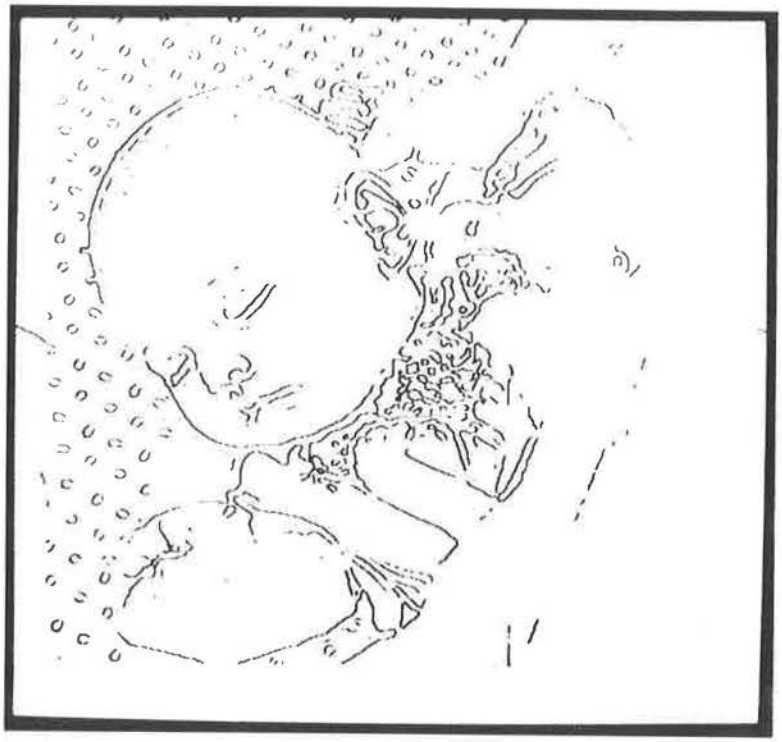

Figure 8: *claire.0x* - Zero-Crossing Edge Detection

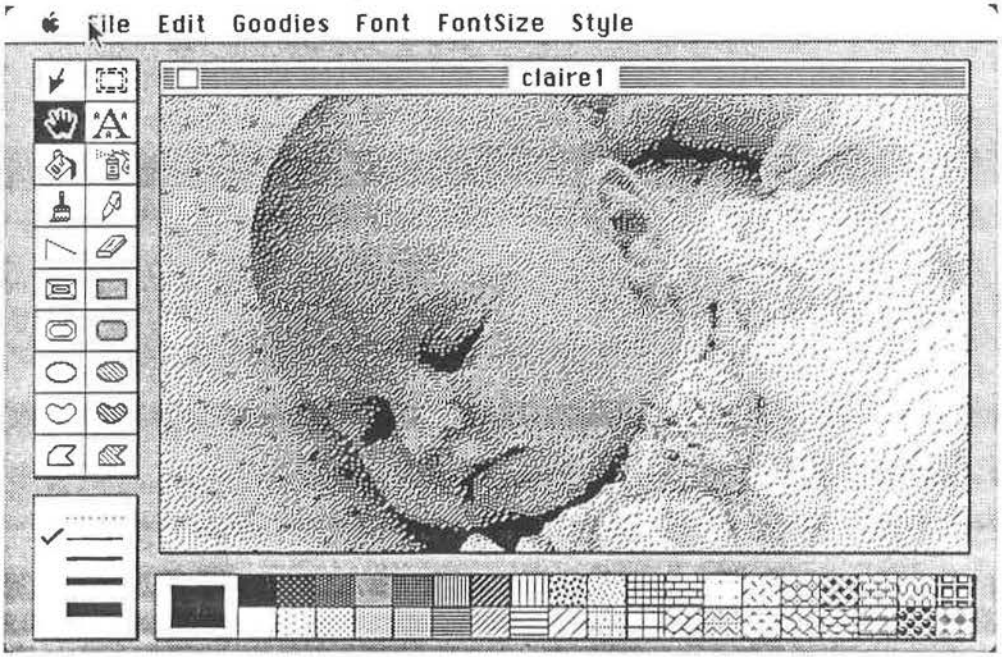

Figure 9: *claire1.pntg* - Macintosh MacPaint Image