COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 23, 239–257

Image Random File Access Routines

SCOTT KRUSEMARK AND ROBERT M. HARALICK

Spatial Data Analysis Laboratory, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061

Received March 17, 1982; revised April 19, 1982 and July 20, 1982

In any large image processing system a set of routines that access the image data is vital. This paper discusses the required image access routines. This set of routines maintains a file format called SIF (standard image format) that is a structured random access file storing both pixel data and image processing history information. The image access routines are built on top of the kernel, an "operating system" set of routines which is fully described in "An operating system interface for transportable image processing software," *Computer Vision, Graphics, and Image Processing*, **23**, July 1983.

1. THE DIGITAL IMAGE

Image processing software must resolve some of the most difficult problems confronting programmed systems: handling large amounts of data which can be accessed only a segment at a time, and carrying out processing tasks that provide heavy and varied loads to the computer system. The solution to these problems require complex software for even the simplest image processing tasks. In this paper we discuss an image access protocol which can be used in solving these problems.

Section 1 defines an image and describes a digital image format capable of handling any image, including the bookkeeping records which contain the entire image processing history associated with the image. Section 2 gives the image access subroutines.

1.1 What Is an Image?

An image is a spatial representation of an object or scene. A recorded image may be in photographic, video, or digital format. A digital image is a quantized representation of an image area mapped into a finite two-dimensional array of small, uniformly shaped, mutually exclusive regions, known as resolution cells. Associated with each resolution cell is a representative gray shade intensity. A digital image may be thought of as a function whose domain is the finite two-dimensional set of resolution cells and whose range is the set of gray shade intensities. A multi-image is a set of images all of the same size, each taken of the same subject at different times, or of the same subject from different angles, or with different sensors. Each image of the multi-image is called a band. A multidigital image is a multi-image in digital format. It can be, for example, a set of digital images obtained from the bands of a multi-image.

1.2 Digital Image Format

An image processing system must maintain digital image information in a file format which is flexible enough to permit a variety of mechanisms for accessing the

0734-189X/83 $3.00

image data. The standard form of digital image representation discussed here allows multidigital images containing both "picture" and "map" information as well as processing history information to be maintained via the kernel system routines. These image I/O routines permit the creation of image files in a manner that gives the user easy access to image data and minimizes storage space.

The routines described here are the specifications for the access routines in GIPSY (general image processing system) which is in use at VPI and SU. They have been designed to take advantage of the capabilities of random access files which may be maintained in five data modes. These data modes are (1) INTEGER, (2) FLOATING POINT, (3) DOUBLE INTEGER, (4) DOUBLE PRECISION, and (5) HALF INTEGER. Some modes may not be implemented at a particular installation and so the actual implementation may use an alternative mode in its place. Integers can be in two formats: absolute binary (positive and zero only) and two's complement (positive, zero, and negative). The integer mode allows any number of bits per pixel data value to facilitate disk storage compaction.

Due to the random I/O required by some image processing algorithms, the access to the image is random access. For example, the user may want processing history information with one command and want image information with another. Random access is more appropriate than sequential access for tasks which may sometimes require only a particular band of a multiband image and other times require all bands.

The standard image format (SIF) is designed so that all information about an image is contained in the image file itself; the user needs to know nothing about the image file to access its data. The SIF multiband image file exists on the various storage devices as a random binary file with fixed length records. The length of the record is determined by the amount of image data to be stored in a logical record. Logical records are separated into three basic categories.

(1) The identification record—contains information with respect to image size, data mode, and the relationship between logical record and subimage size by which the image is stored.

(2) The image data—is stored as mutually exclusive equal sized subimages, band by band. Bands can be numeric or symbolic, the distinction being that arithmetic operations are valid only on numeric bands.

(3) The descriptor records—

(a) provide a history of the image processing that has led to the creation of the current image file;

(b) can contain statistical information such as scaling factors, histograms, eigenvectors, and covariance matrices; and

(c) can contain free format information about the image or file that cannot be stored within the other types.

1.3 The Identification Record

Each identification record contains .IDLENGTH (currently set to 20) words of pertinent information about the SIF file. This places a lower bound on the logical record size of the file. These .IDLENGTH words of information are ordinarily stored in an array during the period when a particular SIF file is being manipulated.

The contents of the IDENT array are as follows:

IDENT(.IDUSR1)	Unused at present, available for program use.
IDENT(.IDUSR2)	Unused at present, available for program use.
IDENT(.IDSYS1)	Currently unused but reserved for internal use.
IDENT(.IDSYS2)	Currently unused but reserved for internal use.
IDENT(.IDNBITS)	Number of bits used to represent one resolution cell in the image.
IDENT(.IDNPPL)	Number of resolution cells per line in the image.
IDENT(.IDNLINS)	Number of lines in the image.
IDENT(.IDHRCS)	Relative size of the resolution cell in the horizontal dimension.
IDENT(.IDVRCS)	Relative size of the resolution cell in the vertical dimension.
IDENT(.IDNDSCRS)	Number of descriptor records in the file.
IDENT(.IDNQL)	Number of quantized levels (the number of discrete levels from the minimum to the maximum gray tone; significant only with respect to INTEGER files.).
IDENT(.IDNWDS)	Number of words per logical record.
IDENT(.IDNCOLS)	Number of columns contained in each subimage or logical record.
IDENT(.IDROWS)	Number of rows contained in each subimage or logical record.
IDENT(.IDMIN)	Minimum gray tone over all bands on the file (may be significant only with respect to INTEGER files).
IDENT(.IDMAX)	Maximum gray tone over all bands on the file (may be significant only with respect to INTEGER files).
IDENT(.IDNBNDS)	Total number of bands in the multiband image.
IDENT(.IDNSBNDS)	Number of symbolic or "map" bands in the multiband set. Symbolic bands are the last bands to occur on the file.
IDENT(.IDMODE)	Data mode indicator = 0 Absolute binary representation (non-negative INTEGERs only); = 1 Two's complement representation (negative as well as positive INTEGERs); = 2 FLOATING POINT representation; = 3 DOUBLE INTEGER representation (may be implemented as an unpacked INTEGER file); = 4 DOUBLE PRECISION representation; = 5 HALF INTEGER representation (may be implemented as unpacked INTEGER file.).
IDENT(.IDVER)	File format version number.

The IDENT array is an argument in the call statement to each of the image access routines. The IDENT array must always have certain specified parameters to be legal. These parameters relate to the size of the logical image, the number of gray tones on the image, and the size of the subimage pieces into which the image is

partitioned. By convention, the IDENT array is always zeroed first and, therefore, the "not set" value is taken to be zero.

(1) Size of image—the size of an image must be specified by specifying the number of columns and the number of rows in the image (IDENT(.IDNPPL) and IDENT(.IDNLINS)).

(2) Mode of image—specified by IDENT(.IDMODE):

(a) Integer mode—in this data mode, image data is compacted to achieve maximum space utilization. Specification of IDENT(.IDNBITS) determines the number of binary digits to be used to represent the discrete gray shades of the image. Alternately, the user can specify the minimum/maximum gray level (IDENT(.IDMIN) and IDENT(.IDMAX)) or the number of quantized levels IDENT(.IDNQL)). Specification of any one of these three units is enough to enable the RDKINL open subroutine to determine the other two. If more than one is specified, the values must be consistent. If none are specified, IDENT(IDNBITS) defaults to four bits.

(b) Floating point mode—floating point data is not compacted in SIF files. Therefore, IDENT(.IDNBITS) is automatically set to the number of bits in a REAL variable on the host machine. IDENT(.IDNQL), IDENT(.IDMIN), and IDENT(.IDMAX) may be set at the user's discretion but are not required. If they are not set they default automatically to the largest and smallest integers on the host machine.

(c) Double integer mode—same as floating point except that some machines do not have DOUBLE INTEGER so this mode may be implemented as single INTEGER but not packed.

(d) Double precision mode—same as floating point except number of bits is for DOUBLE PRECISION variables.

(e) Half integer mode—same as DOUBLE INTEGER. It may be implemented as nonpacked INTEGER for machines whose compiler does not have a half integer mode.

(3) Subimage size—the resolution cells of an SIF image are organized into rectangular groups of resolution cells called subimages. A subimage may be a row, part of a row, a column, or an NR row by NC column block of resolution cells. Each subimage is written to the SIF file as an indivisible unit (one logical record). Subimage size is determined by IDENT(.IDNROWS) and IDENT(.IDNCOLS). Subimages cover an image in columns moving from left to right. The number of subimages in a column is determined by the number of rows in the image and the number of rows in a subimage. The number of subimages in a column is the minimum number needed to cover all rows in the image. Similarly, the number of columns of subimages is determined by the number of columns in the image and the number of columns in a subimage. The product of the number of subimages per column times the number of subimages per row yields the number of subimages in an image.

Pixels contained within any subimages that lie outside the logical image are left undefined. If the subimage size is not specified, row format is assumed, i.e., IDENT(.IDNCOLS) = IDENT(.IDNPPL) and IDENT(.IDNROWS) = 1.

(4) Other information contained in the IDENT array:

(a) IDENT(.IDUSR1) and IDENT(.IDUSR2) are available for the user to pass information in at the debug stage of program development. The values will not be changed by any of the I/O subroutines.

(b) After initialization IDENT(.IDNWDS) contains the number of words per logical record. This determines the logical record size of every record on the file.

(c) IDENT(.IDNBNDS) specifies the number of bands on the file. If it is not set, it defaults to 1.

(d) IDENT(.IDNSBNDS) specifies the number of symbolic bands on the file. If it is not set, it defaults to 0.

(e) IDENT(.IDVER) specifies the version number of the file and is set by the open subroutine RDKINL. Having a built-in version number allows the natural expansion, extension, and modification of the file format without disrupting any user's existing file.

(f) IDENT(.IDHRCS) and IDENT(.IDVRCS) default to 1 and specify the relative length to width ratio of the resolution cell.

1.4 The Image Data

Image data records make up a major portion of the SIF file. The digital multiband image is divided into mutually exclusive rectangular regions called subimages and each logical record contains one subimage. The data that is stored in these subimages corresponds to the mode indicated by IDENT(.IDMODE). Most often, SIF files contain INTEGER data. INTEGER data is compacted so that each datum uses a minimum amount of storage space. The number of pixels per word is determined by the number of significant bits per datum—IDENT(.IDNBITS), the "byte" length. The total number of data values per logical record is determined by the subimage size.

Example Calculation

Given (1) subimage size of 64 pixels,
(2) seven (7) significant bits per data value,
(3) INTEGER mode,
(4) machine word size of 32 bits per word,
yields four data values in the first word with a portion of the fifth data value in that first word. The total number of words needed is

$$\text{wrd} = [64 * 7, 32]$$

where $[a, b]$ is smallest integer greater than or equal to a/b.

1.5 Image Data Organization

(1) Pixel values within a SIF subimage are ordered sequentially in a row by row manner as shown in Fig. 1.

(2) These subimages cover the entire image in a columnwise fashion as shown in Fig. 2.

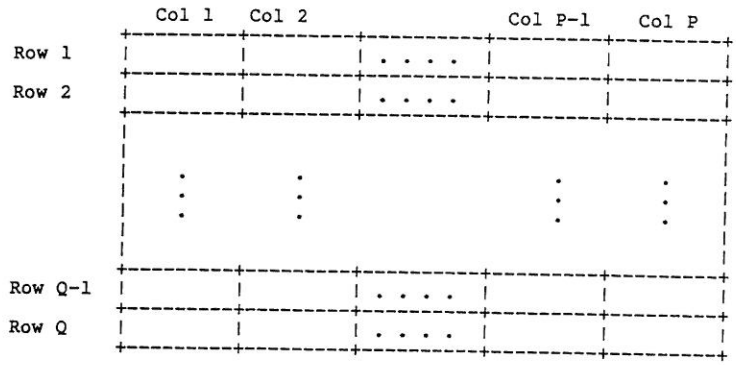

FIG. 1. Illustration of logical image.

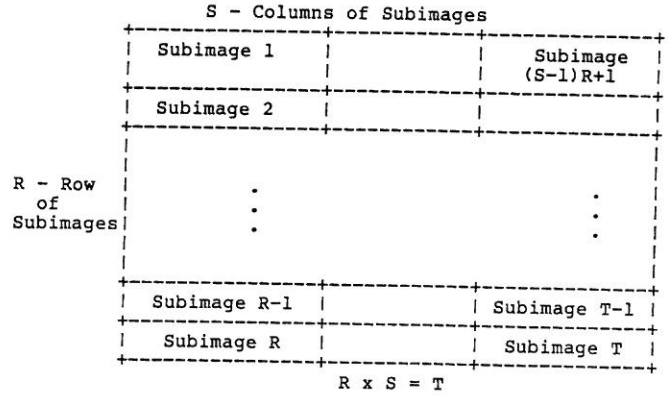

FIG. 2. How an image is covered by subimages.

(3) Fig. 3 illustrates how the entire SIF multiband image file appears on the storage device. Each record is K words long, where K is determined by the subimage size, the data mode, and the number of bits/point if the data mode is integer. The identification record is first, followed by IDENT(.IDNDSCRS) descriptor records and then the image data. If the entire data area is viewed as a four dimensional array (L, M, N, O) where

L = number of bands

M = number of subimages

N = number of rows in subimage

O = number of columns in subimage

then the subscripts would vary from the fastest to slowest in the following order:

O columns	(1 to IDENT(.IDNCOLS))
then N rows	(1 to IDENT(.IDNROWS))

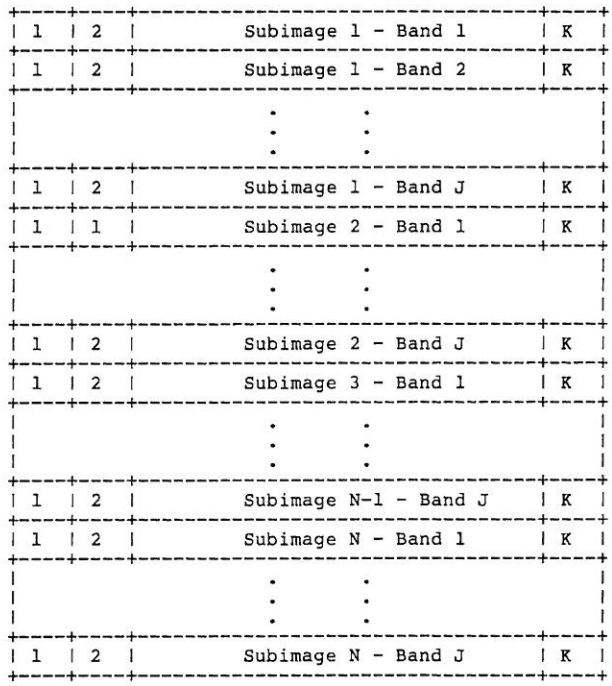

FIG. 3. Image/subimage layout in standard image file.

then L band (1 to IDENT(.IDNBNDS))

then M subimage (1 to
[IDENT(.IDNLINS), IDENT(.IDNROWS)]*
[IDENT(.IDNPPL), IDENT(.IDNCOLS)]

where $[a, b]$ is the smallest integer greater than or equal to a/b.

1.6 The Descriptor Records

The standard image data format for gray tone intensity data has been described. This kind of data, however, is not the only information kept with the image file. Images are often processed with one operator followed by another. If the order of application of these operations is not written down along with their parameters, then an entire interactive session may have to be repeated. To help the user, an image processing system needs to keep the entire processing history tied to the image file. This history is stored in descriptor records. Descriptor records are an optional part of the SIF file in the sense that the file is a legal file even if there are no history records. The number of descriptor records in an image file is given by IDENT(.IDNDSCRS). Each descriptor record contains a maximum of .IDLENGTH words of actual information. This insures that as descriptor records are copied from file to file, no information is lost as IDENT(.IDNWDS) changes. Descriptor records provide general information about the image as well as a detailed history of the processing steps, command by command with all the user specified parameter values listed.

2. IMAGE ACCESS PROTOCOL SUBROUTINES

Before discussing the actual routines for image access and descriptor record access several conventions need to be described. These are the use of the alternate return mechanism and the file descriptor.

2.1 IEV and Alternate Return Convention

The argument lists for all the system I/O access routines end with

IEV, %XXXX).

The variable IEV is the integer event variable. It takes on a negative value when an error has occurred. The error list tells what each specific value represents. This variable will have the value represented by the token .OK if the routine functioned as expected.

The argument %XXXX is an alternate return as defined by FORTRAN. The XXXX needs to be replaced by a statement number such as 1234. When all proceeds as expected in the routine, a normal return is taken and processing continues at the statement following the call. If, however, an error occurs the alternate return is taken and processing continues at the statement with the label given in the call: %XXXX. For instance, in the example

```
CALL SUB(..., IEV, %9000)
STOP 100
.
.
9000 STOP 200
```

if all goes well, it will stop with a value of 100. If an error had occurred, then the variable IEV will be set and the alternate return will be taken. Execution will continue at 9000 and finally execution will terminate with a value of 200.

The percent sign (%) has been selected as the standard character for alternate returns. This character is changed on output from the RATFOR preprocessor to whatever the host system needs. For example, on both the IBM 370 under CMS and on the VAX 11/780, the percent is changed to an ampersand (&), and on the PDP-15 to an "at" sign (@).

2.2 Input/Output File Descriptor

The file descriptor (FD) used in the following routines is an array that contains system dependent information such as the filename, where the file is located, and logical unit number (if one is used or needed). The details about how file descriptors are created can be found in Ref. [1]. The FD contains in system dependent form all information that specifies the image file and needs to be passed between I/O routines needed to open, close, read, and write to/from a file.

2.3 SIF Image Data Access Routines

The SIF primitive I/O access routines consist of routines to open, close, read, and write random files.

2.3.1 Random File Open

CALL RDKINL(FD, IDENT, OLDNEW, IEV, %XXXX)

The subroutine RDKINL (random disk initialize) performs the random file open for both new and old files. The IDENT is the identification array as described earlier. The argument OLDNEW has one of the two values denoted by the token definitions .NEW and .OLD, for new and old files, respectively, to be opened.

In a new file, the values in the IDENT array are checked for consistency and unspecified values are initialized. A random file of the correct size is then created and the IDENT array is written to the first record of the file.

In an old file, the routine first checks for its existence on the disk. If it does not exist, an error is generated and the alternate return is taken. If the file does exist, it is opened and the first record is read into the IDENT array. These values are assumed to be correctly initialized when the file was created.

2.3.2 Random File Read/Write

CALL RREAD(FD, BUF, BND, BLKNO, IDENT, WAIT, IEV, %XXXX)

CALL RWRITE(FD, BUF, BND, BLKNO, IDENT, WAIT, IEV, %XXXX)

The routine RREAD takes the data on the image and copies it to the buffer and the routine RWRITE copies the data in the buffer to the image, unpacking and packing where needed. After a SIF file has been opened the image data is available via RREAD and RWRITE. The file descriptor (FD), identification array (IDENT), and alternate return arguments (IEV, %XXXX) have been described previously.

Since image access is random, the two arguments BND and BLKNO are needed to specify the correct band and block number from the SIF file. The word "block" corresponds to a subimage or logical record. Blocks are numbered starting in the upper left corner and proceeding down the left side to the bottom of the image.

The buffer (BUF) must be dimensioned for the correct size of one block. Only one block can be obtained at a time. However, if a value of zero (0) is specified for the band number argument (BND), the block specified is returned for all bands. Normally the band number and block number are specified, thus returning only one block from one band. Also, the most common shape of a block is an entire image row. The WAIT argument is usually set to .WAIT to indicate that the routine is to wait for the I/O to complete before returning. The WAIT argument is fully described in [1].

2.3.3 CLOSE

CALL CLOSE(FD)

CLOSE complements "open," in this case, RDKINL. All files must be closed. (It is not an error to close an already closed file, but all new files must be closed before exiting to guarantee that they continue to exist in the expected form.)

2.4 History Record Routines

History record routines must permit a program to write out all the parameter values which define its execution as well as any summary statistics that it produces and is required to save in the history records. Since the history records are part of the random access image data file, and since to open a random file it is generally required to know how many records there are to be in the file, a program creating history records must know how many history records it is adding. This calculation is a nuisance for the programmer to make. To alleviate the nuisance the history records to be copied from the input file(s) and the history records added by the current processing are first written to a sequential file while a count is automatically kept of how many records are on the file. After all history records have been written to the sequential file and a total count of them is known, the output random access file's size is known and it can be opened.

The routine to copy the history records from the input file to a temporary sequential file is called CPYIDR. The call to CPYIDR is followed by a call to DSCNAM which stands for descriptor record name information. It puts the program name in the history records along with the command name and all input image file names which are held in the labeled common area GIPCOM.

All the user specified parameters such as number of bands to process, threshold value, first–last rows, etc., are then put to the temporary file via the PDSCXX routines. The XX is replaced with I, R, etc. for integer or real constants or IA, RA, etc. for integer or real arrays. These routines have a character string argument describing the numeric data they put out to the history records. Finally, a call to COPYDS must be made, creating the actual output to the SIF file. COPYDS copies the history records on the temporary file to the SIF and deletes the temporary file. The call for the output RDKINL is done inside of COPYDS. At this point all files are open, the history records have been written, and everything is ready for the image processing to be done.

2.4.1 Copy Descriptor Records

CALL CPYIDR(FDI, IDENT, TEMP, IEV, %XXXX)

This routine opens the input file (FDI) and when TEMP takes the value .OPNTMP creates the temporary sequential file. This can be viewed as a replacement for RDKINL on the input file. It should be the first history record subroutine called. The history records on the input file are copied to the temporary sequential file, and the IDENT array is returned filled.

In some situations, more than one input file is needed so two calls to CPYIDR are needed, one for each file. The first call needs the argument TEMP to be the symbolic token of .OPNTMP (for open temporary file). For the second call (and any others if more than two input files), the argument TEMP must have the value of .NOOPNTMP which does not open the temporary file and which just continues to write onto the temporary sequential file initially opened.

2.4.2 Open Descriptor Record

CALL OPNDSR(IEV, %XXXX)

When the input file either does not exist or is not a SIF file, there are no history records to copy. CPYIDR has no meaning, therefore, beyond opening the temporary

file. In these cases the call to CPYIDR is replaced with a call to OPNDSR. No calling arguments are needed except the IEV and alternate return.

2.4.3 Name Record Information

CALL DSCNAM(PRGNM, IEV, %XXXX)

This routine, used in conjunction with the common GIPCOM, puts out to the history records the name of the command, the program name, the name of all files in the labeled common area GIPCOM (three input and three output) and all the flags specified by the user interactive command string. The output is to the temporary sequential file opened by CPYIDR.

The argument PRGNM in quotes must be padded to six characters and be the subroutine name that this line of code is in. This is a packed string format.

2.4.4 Put Value

CALL PDSCI(PSTRG, VALUE, IEV, %XXXX)

CALL PDSCR(PSTRG, VALUE, IEV, %XXXX)

CALL PDSCD(PSTRG, VALUE, IEV, %XXXX)

CALL PDSCP(PSTRG, VALUE, IEV, %XXXX)

CALL PDSCH(PSTRG, VALUE, IEV, %XXXX)

CALL PDSCL(PSTRG, VALUE, IEV, %XXXX)

This set of routines writes a data value to the descriptor records. The value as well as a character string description of that value is written to the temporary file opened by CPYIDR. The packed string (PSTRG) should be in quotes with a period terminator:

"NUMBER OF BANDS."

Note that a period cannot be used as part of the text as this would signal the end of the string. The variable VALUE contains the data value to be written to the descriptor records and is of the type designated by the routine name where *I* is for integer, *R* for real, *D* for double integer, *H* for half integer, *P* for double precision real, and *L* for logical.

2.4.5 Put Array

CALL PDSCIA(PSTRG, VALARY, LEN, IEV, %XXXX)

CALL PDSCRA(PSTRG, VALARY, LEN, IEV, %XXXX)

CALL PDSCDA(PSTRG, VALARY, LEN, IEV, %XXXX)

CALL PDSCPA(PSTRG, VALARY, LEN, IEV, %XXXX)

CALL PDSCHA(PSTRG, VALARY, LEN, IEV, %XXXX)

CALL PDSCLA(PSTRG, VALARY, LEN, IEV, %XXXX)

This set of routines does the same functions as the corresponding routines without the trailing "A" but puts out an array instead of a constant. VALARY is the array to be written to the descriptor records and is of the correct type defined by the routine name. LEN is the length of the array.

2.4.6 Packed String Routines

CALL PDSCS(A1STR, IEV, %XXXX)

CALL PDSCSA(A1STR, CNT, IEV, %XXXX)

CALL PDSCPS(PSTRNG, IEV, %XXXX)

This set of routines writes character information to the descriptor records. The argument A1STR is an A1 format character string. The argument PSTRNG is a packed character format character string. The call to PDSCS expects a .EOS or period (.) as a terminator. The call to PDSCSA expects the count to be specified. Finally, the call to PDSCPS expects the period (.) terminator in the packed string.

2.4.7 Put Matrix

CALL PDSCMT(TITLE, CLBL, RLBL, MATRIX, ROW,
COL, MMODE, IEV, %XXXX)

This routine writes a matrix with the corresponding titles to the descriptor records. The arrays TITLE, CLBL, and RLBL are packed strings with a period (.) terminator. The matrix is a two-dimensional array of size ROW by COL. The argument MMODE is the matrix mode (.INTMODE, .REALMODE, etc.).

2.4.8 Open Output File

CALL COPYDS(FDO, JDENT, IEV, %XXXX)

This routine opens the output file. Since all history records are on the temporary file, the number of records needed to hold the history information on the SIF is known. Hence, the size of the output file is known. The JDENT array must have the values correctly set up so that RDKINL can be called by COPYDS. These values generally include image size, number of bands, and number of bits per pixel, number of gray tone levels, minimum, and maximum. If something other than a row image is desired, the subimage sizes needs to be specified (See Section 1.3).

After the file is opened, the history records on the temporary file are copied to the output file and the temporary file is deleted. At this point, all files are open and image processing can start.

2.5 Readable Descriptor Records

The format of the descriptor records described above is stored in a free format way that allows for compactness but not for easy retrievability. There needs, therefore, to be a way to put information into the descriptor records so that the information can be located and retrieved under program control. This second way of putting the descriptor records out uses a numeric indicator instead of a character string for identification. The subroutines are called PDSCAX and GDSCAX.

2.5.1 Put Retrievable Record

CALL PDSCAX(TYPE, BUFF, LEN, IEV, %XXXX)

This routine puts out the information specified in the buffer. The TYPE is an indicator of what the information type and format is. This type specifies the format to be followed and decides the type of information, such as ground truth or statistical data. For example, TYPE 1 is ground truth. The data in BUFF is LEN integers long and is organized as specified for the specific TYPE.

2.5.2 Retrieve Retrievable Information

CALL GDSCAX(FDI, TYPE, BUFF, IDENT, LEN, IEV, %XXXX)

This routine gets the information back from the history records. The TYPE specifies which information type to retrieve. In this way only the information desired is retrieved. If multiple types exist in a file, only the matching type is returned. The FDI and IDENT are of an open file such as one just called on CPYIDR or RDKINL with the .OLD option.

2.6 Use and Example of History Record Routines

The following example will help to clarify the order and use of the routines.

```
  INCLUDE MACA1                                  contain all token
#                                                definitions
#
  SUBROUTINE NUMBCH(FDI, FDO, BND, NBND,LBUF, THRSH,
                    IEV, *)
#
  IMPLICIT INTEGER (A–Z)
#                                                declare all variables
#                                                to default to type
#                                                INTEGER
#
  REAL THRSH
  CHARACTER FDI(.FDLENGTH), FDO(.FDLENGTH)
  INTEGER BUF(LBUF), IDENT(.IDLENGTH)
  INTEGER JDENT(.IDLENGTH), BND(NBND)
#
  CALL PPUSH('NUMBCH')                  #        document subroutine
#                                                call
#
  CALL CPYIDR(FDI, IDENT, .OPNTMP, IEV, %9000)
#                                                open input file and temp
#                                                file
  IF(IDENT(.IDNPPL) > LBUF)GOTO 9010
#                                                check if too large
#                                                an image
```

```
#
  CALL DSCNAM('NUMBCH', IEV, %9000)
#                                                    put out name information
#
#                                                    enter user information
#                                                    section
#
  CALL PDSCIA('BAND NUMBERS.', BND, NBND, IEV, %9000)
  CALL PDSCR('THRESHOLD.', THRSH, IEV %9000)
#
#                                                    set up output IDENT
#                                                    record
#
  DO I = 1, .IDLENGTH
      JDENT(I) = 0
#
  JDENT(.IDNPPL) = IDENT(.IDNPPL)          #         output number of points
#                                                    per line same as input
#                                                    file
  JDENT(.IDNLINS) = IDENT(.IDNLINS)        #         output number
#                                                    of lines = input
  JDENT(.IDNBITS) = 8                      #         number of bits is 8
#                                                    bits (0 to 255)
  JDENT(.IDNBNDS) = NBND
#                                                    default to row format
#                                                    image INTEGER image.
#
#CALL COPYDS(FDO, JDENT, IEV, %9000)
#
#                                                    This section of code
#                                                    is the "real"
#                                                    image processing.
#
#                                                    This example is simply
#                                                    a copy from input to
#                                                    output
#NLIN = IDENT(.IDNLINS)
  DO OBND = 1, NBND
      $(
      IBND = BND(OBND)
      DO LIN = 1, NLIN
          $(
          CALL RREAD(FDI, BUF, IBND, LIN, IDENT,
                     .WAIT, IEV, %9000)
          CALL RWRITE(FDO, BUF, OBND, LIN, JDENT,
                      .WAIT, IEV, %9000)
          $)
      $)
```

```
#
  CALL CLOSE(FDI)                    #    close input file
  CALL CLOSE(FDO)                    #    close output file
#
  CALL PPOP                          #    document program exit
#
  RETURN
#
  9000 CONTINUE
#
#                                         Error in lower
#                                         routine.
#                                         IEV already set.
  GOTO 9999
  9010 CONTINUE
#
  IEV = -3013                        #    buffer smaller than
#                                         data to go in it.
#
  9999 CONTINUE
  CALL CLOSE(FDI)                    #    even on error
#                                         must close all
  CALL CLOSE(FDO)                    #    files input and output
#
  RETURN 1                           #    take alternate return
#
  END
```

2.7 Explanation of Example

The general INCLUDE that defines all tokens is the first line of code. The file descriptors FDI and FDO are for input and output name information. The IMPLICIT typing is used to simplify coding. The data BUF is passed from above. The calling program makes it large enough to do what needs doing.

By convention, the call to PPUSH is the first executable statement with the routine name as its argument. This allows program controlled dynamic tracing, if desired.

The history records created in the above program use a call to CPYIDR and then a call to DSCNAM to put out the general information; PDSCIA and PDSCR output some parameters to the history records. Next the output image file IDENT array is set up, and a call to COPYDS to open the actual output file is made.

The next section of code is the actual algorithm which in this example simply copies the input to the output.

After the algorithm is finished for all bands and all lines, the files must be closed; the program stack is then popped, and finally, a return to the calling program can be made.

A check for errors such as that the number of pixels per line is less than or equal to the buffer size is made. Once detected, errors are dealt with at the bottom of the routine in the manner shown. This starts the alternate return chain.

2.8 Postprocessing Information for History Records

In some situations in image processing, all the information to go into the history records is not known before the actual image processing is done. For example, connected components processing could generate a parameter such as the number of connected components, a number known only after the actual output data is created. This postprocessing requirement is solved with a set of routines (MRKDXX) that work on the output SIF data rather than the temporary file data. The method is to mark a starting point and put out place holders (using PDSCXX), open the output as usual, then process the image data and finally call a special set of routines that operate directly on the SIF file history records, overwriting the place holders.

The call to MARKDS separates the data that is already known from the unknown data values. At some time after COPYDS but before the close are a series of calls to the MRKDXX routines that exactly duplicate the calls to the PDSCXX routines after MARKDS and before COPYDS. The character strings must be the same and the order and type of calls must be the same or unpredictable results could occur.

CALL MARKDS

This routine marks the current location of the character stream of the history records. It separates the history records which are complete from those which are not complete. It should be called only once.

CALL MRKDI(FDO, IDENT, PSTRG, VALUE, IEV, %XXXX)

CALL MRKDR(FDO, IDENT, PSTRG, VALUE, IEV, %XXXX)

CALL MRKDP(FDO, IDENT, PSTRG, VALUE, IEV, %XXXX)

CALL MRKDD(FDO, IDENT, PSTRG, VALUE, IEV, %XXXX)

CALL MRKDH(FDO, IDENT, PSTRG, VALUE, IEV, %XXXX)

CALL MRKDL(FDO, IDENT, PSTRG, VALUE, IEV, %XXXX)

This set of routines corresponds to the calls of PDSCI, PDSCR, etc., and should be in the same order with the same character strings (PSTRG). The file descriptor (FDO) and IDENT array are for the output file which is assumed open.

CALL MRKDIA(FDO, IDENT, PSTRG, ARRY, LEN, IEV, %XXXX)

CALL MRKDRA(FDO, IDENT, PSTRG, ARRY, LEN, IEV, %XXXX)

CALL MRKDPA(FDO, IDENT, PSTRG, ARRY, LEN, IEV, %XXXX)

CALL MRKDDA(FDO, IDENT, PSTRG, ARRY, LEN, IEV, %XXXX)

CALL MRKDHA(FDO, IDENT, PSTRG, ARRY, LEN, IEV, %XXXX)

CALL MRKDLA(FDO, IDENT, PSTRG, ARRY, LEN, IEV, %XXXX)

These routines are similar to MRKDX except for array rather than constant. The lengths also should match the corresponding call of PDSCXA called before COPYDS. The next section gives an example program using the postprocessing history records.

2.9 Example of Postprocessing History Records

```
  INCLUDE MACA1
#
  SUBROUTINE NMBCH1(FDI, FDO, BND, NBND, LBUF, THRSH,
                    IEV, *)
#
  IMPLICIT INTEGER (A–Z)
#                                                  declare all variables
#                                                  to default to type
#                                                  INTEGER
#
  REAL THRSH
  CHARACTER FDI(.FDLENGTH), FDO(.FDLENGTH)
  INTEGER BUF(LBUF), IDENT(.IDLENGTH)
  INTEGER JDENT(.IDLENGTH), BND(NBND)
#
#
  CALL PPUSH('NMBCH1')                    #      document subroutine
#                                                  call
#
#
  CALL CPYIDR(FDI, IDENT, .OPNTMP, IEV, %9999)
#                                                  open input file and
#                                                  temp file
  IF(IDENT(.IDNPPL) > LBUF) GOTO 9010
#                                                  check if too large
#                                                  an image
#
  CALL DSCNAM('NMBCH1', IEV, %9999)
#                                                  put out name
#                                                  information
#
#                                                  enter user infor-
#                                                  mation section
#
  CALL PDSCIA('BAND NUMBERS.', BND, NBND, IEV, %9999)
  CALL PDSCR('THRESHOLD.', THRSH, IEV, %9999)
#
#                                                  mark this place in
#                                                  the history records
#
  CALL MARKDS
#
#                                                  put out the
#                                                  place holder filler
#
  CALL PDSCI('TRUE IMAGE MINIMUM.', 0, IEV, %9999)
```

```
  CALL PDSCI('TRUE IMAGE MAXIMUM.', 0, IEV, %9999)
#
#                                                    set up output IDENT
#                                                    record
#
  DO I = 1, .IDLENGTH
       JDENT(I) = 0
#
  JDENT(.IDNPPL) = IDENT(.IDNPPL)          #         output number of
#                                                    points per line same
#                                                    as input file
  JDENT(.IDNLINS) = IDENT(.IDNLINS)        #         output number
#                                                    of lines = input
  JDENT(.IDNBITS) = 8                      #         number of bits is 8
#                                                    bits (0 to 255)
  JDENT(.IDNBNDS) = NBND
#
#                                                    default to 1 band,
#                                                    row format image
#                                                    INTEGER image.
#
  CALL COPYDS(FDO, JDENT, IEV, %9999)
#
#
#                                                    This section of code
#                                                    is the 'real' image
#                                                    processing.
#
#                                                    This example is simply
#                                                    a copy from input
#                                                    to output that gets the
#                                                    true image min/max
#
  NPPL = IDENT(.IDNPPL)
  NLIN = IDENT(.IDNLINS)
  MAX = 0
  MIN = .LARGEINTEGER
#
  DO OBND = 1, NBND
       $(
       IBND = BND(OBND)
       DO LIN = 1, NLIN
            $(
            CALL RREAD(FDI, BUF, BND, LIN, IDENT,
                       .WAIT, IEV, %9999)
            DO PNT = 1, NPPL
                 $(
                 IF(BUF(PNT) < MIN) MIN = BUF(PNT)
```

```
                IF(BUF(PNT) > MAX) MAX = BUF(PNT)
                $)
            CALL RWRITE(FDO, BUF, BND, LIN, JDENT,
                        .WAIT, IEV, %9999)
            $)
#
#                                                   put out the post
#                                                   descriptor records
#
  CALL MRKDI(FDO, JDENT, 'TRUE IMAGE MINIMUM.',
                MIN, IEV, %9999)
  CALL MRKDI(FDO, JDENT, 'TRUE IMAGE MAXIMUM.',
                MAX, IEV, %9999)
#
  CALL CLOSE(FDI)                          #    close input file
  CALL CLOSE(FDO)                          #    close output file
#
  CALL PPOP                                #    document program exit
#
  RETURN
#
  9010 CONTINUE
#
  IEV = -3013                              #    buffer smaller than
#                                                data to go in it.
#
  9999 CONTINUE                            #    Error in lower routine
#                                                IEV already set.
  CALL CLOSE(FDI)                          #    even on error must
#                                                close
  CALL CLOSE(FDO)                          #    all files input and
#                                                output
  RETURN 1                                 #    take alternate return
#
  END
```

REFERENCE

1. Scott Krusemark and Robert M. Haralick, An operating system interface for transportable image processing software, *Computer Vision, Graphics, and Image Processing*, **23**, July 1983.