NIST Special Database 18

Mugshot Identification Database

C.I.Watson

National Institute of Standards and Technology
Advanced Systems Division
Visual Image Processing Group
December 1994

1.0 Introduction

This document describes *NIST Special Database 18*, which contains 8-bit gray scale images of mugshot photographs. This database is being distributed for use in developing and testing of mugshot identification systems. The database contains images of 1573 individuals (cases) with a total of 3248 images stored in NIST's IHead raster data format. The mugshots are mainly of male cases, with the database containing 1495 male cases and 78 female cases. The sex and age of each individual are stored in the header of each file as well as an included log file. The included log file also gives information about which files are of the same individual (see section 5).

The database images consist of both front views and side views (profiles), although not every case has both a front and profile. Looking at the front views and profiles separately, there are 131 cases with two or more front views and 1418 with only one front view. There are 89 cases with two or more profiles and 1268 with only one profile. In cases that have both fronts and profiles, there are only 89 cases with two or more of both fronts and profiles, 27 with two or more fronts and one profile, and 1217 with only one front and one profile.

The size of each image varies, because the mugshot photographs vary in size from 1" - 2 1/2" in height. Rather than storing an image that was more than 50% background pixels, some of the background was discarded. All of the images except for 43 were scanned at 19.685 pixels/mm (500 dpi). The 43 that weren't scanned 19.685 pixels/mm were full front views of the individual. In order to get sufficient facial information the heads of these image were scanned at 39.37 pixels/mm (1000 dpi).

2.0 Modified JPEG Lossless Compression

The compression used was developed from techniques outlined in the WG10 "JPEG" (draft) standard [1] for 8-bit gray scale images with modifications to the compressed data format. The NIST IHead format already contained most of the information needed in the decompression algorithm, so the JPEG compressed data format was modified to contain only the information needed when reconstructing the Huffman code tables and identifying the type of predictor used in the coding process. Codes used to compress and decompress the images are still developed per the draft standard, but only applied to 8-bit gray scale images.

3.0 Database Scanning Procedures

The images were scanned using a Kodak MegaPixel[1] camera [2] and lighting powered by a direct current (DC) power supply to eliminate light flicker. In order to scan images of consistent quality the reflectance and focusing were checked approximately every two hours using the procedures described in the next two sections.

3.1 Database Reflectance Calibration

The reflectance for the images scanned in *NIST Special Database 18* was calibrated using two reflectance charts. The first one consisted of continuous gray tones from black to white and the second contained only 64 gray levels from black to white partitioned into distinct blocks. The lighting intensity was adjusted so that the charts, when scanned, produced gray levels from 38 to 255. The range of 217 gray levels was the maximum range the camera was able to distinguish.

3.2 Camera Focusing

The focusing procedure used a target image which consisted of equally spaced, alternating black and white lines. Using a software tool, the standard deviation of the gray level values on a line perpendicular to the black and white lines in the target image was calculated. The procedure used was to adjust the camera focus so that the standard deviation of the cross-section was maximum. The maximum standard deviation is the point at which there is the least "blurring" in the transition regions between the white and black lines thereby focusing the camera.

4.0 Mugshot File Format

Image file formats and effective data compression and decompression are critical to the usefulness of image archives. Each mugshot was digitized in 8-bit gray scale form at 19.6850 pixels/mm (500 pixels/inch), 2-dimensionally compressed using a modified JPEG lossless algorithm, and temporarily archived onto computer magnetic mass storage. Once all prints were digitized, the images were mastered and replicated onto ISO-9660 formatted CD-ROM discs for permanent archiving and distribution.

After digitization, certain attributes of an image are required to correctly interpret the 1-dimensional pixel data as a 2-dimensional image. Examples of such attributes are the pixel width and pixel height of the image. These attributes can be stored in a machine readable header prefixed to the raster bit stream. A program which manipulates the raster data of an image is able to first read the header and determine the proper interpretation of the data which follows it.

Numerous image formats exist, but most image formats are proprietary. Some are widely supported on small personal computers and others on larger workstations. A header format named IHead [3] has been developed for use as a general purpose image interchange format. The IHead header is an open image format which can be universally implemented across heterogeneous computer architectures and environments. Both documentation and source code for the IHead format

---

1. The Kodak MegaPixel is identified in order to adequately specify or describe the subject matter of this work. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment is necessarily the best available for the purpose.

are publicly available and included with this database. IHead has been designed with an extensive set of attributes in order to adequately represent both binary and gray level images, to represent images captured from different scanners and cameras, and to satisfy the image requirements of diversified applications including, but not limited to, image archival/retrieval, character recognition, fingerprint classification, and mugshot identification. Figure 1 illustrates the IHead format.

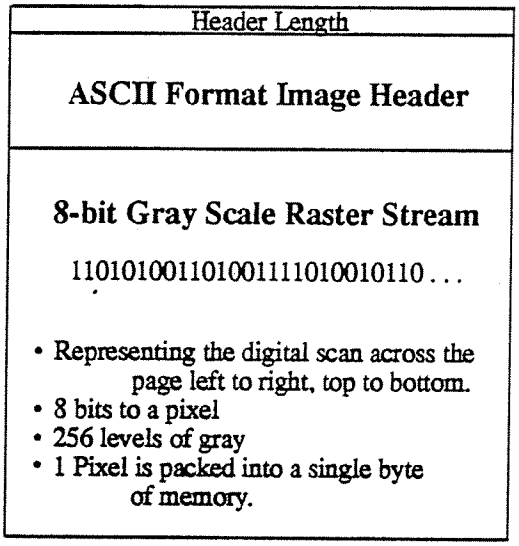

Figure 1: An illustration of the Ihead raster file format.

Since the header is represented by the ASCII character set, IHead has been successfully ported and tested on several systems including UNIX workstations and servers, DOS personal computers, and VMS mainframes. All attribute fields in the IHead structure are of fixed length with all multiple character fields null-terminated, allowing the fields to be loaded into main memory in two distinct ways. The IHead attribute fields can be parsed as individual characters and null-terminated strings, an input/output format common in the 'C' programming language, or the header can be read into main memory using record-oriented input/output. A fixed-length field containing the size in bytes of the header is prefixed to the front of an IHead image file as shown in Figure 1.

The IHead structure definition written in the 'C' programming language is listed in Figure 2. Figure 3 lists the header values from an IHead file corresponding to the structure members listed in Figure 2. This header information belongs to the database file **f00001_1.pct**. Referencing the structure members listed in Figure 2, the first attribute field of IHead is the identification field, **id**. This field uniquely identifies the image file, typically by a file name. The identification field in this example not only contains the image's file name, but also the sex and age of the individual.

The attribute field, **created**, is the date on which the image was captured or digitized. The next three fields hold the image's pixel **width, height**, and **depth**. A binary image has a pixel depth of 1 whereas a gray scale image containing 256 possible shades of gray has a pixel depth of 8. The

attribute field, **density**, contains the scan resolution of the image; in this case, 19.6850 pixels/mm (500 pixels/inch). The next two fields deal with compression.

```
/*************************************************************
      File Name: IHead.h
      Package:  NIST Internal Image Header
      Author:   Michael D. Garris
      Date:     2/08/90
*************************************************************/
/* Defines used by the ihead structure */
#define IHDR_SIZE        288     /* len of hdr record (always even bytes) */
#define SHORT_CHARS      8       /* # of ASCII chars to represent a short */
#define BUFSIZE          80      /* default buffer size */
#define DATELEN          26      /* character length of data string */

typedef struct ihead{
  char id[BUFSIZE];               /* identification/comment field */
  char created[DATELEN];          /* date created */
  char width[SHORT_CHARS];        /* pixel width of image */
  char height[SHORT_CHARS];       /* pixel height of image */
  char depth[SHORT_CHARS];        /* bits per pixel */
  char density[SHORT_CHARS];      /* pixels per inch */
  char compress[SHORT_CHARS];     /* compression code */
  char complen[SHORT_CHARS];      /* compressed data length */
  char align[SHORT_CHARS];        /* scanline multiple: 8|16|32 */
  char unitsize[SHORT_CHARS];     /* bit size of image memory units */
  char sigbit;                    /* 0->sigbit first | 1->sigbit last */
  char byte_order;                /* 0->highlow | 1->lowhigh*/
  char pix_offset[SHORT_CHARS];   /* pixel column offset */
  char whitepix[SHORT_CHARS];     /* intensity of white pixel */
  char issigned;                  /* 0->unsigned data | 1->signed data */
  char rm_cm;                     /* 0->row maj | 1->column maj */
  char tb_bt;                     /* 0->top2bottom | 1->bottom2top */
  char lr_rl;                     /* 0->left2right | 1->right2left */
  char parent[BUFSIZE];           /* parent image file */
  char par_x[SHORT_CHARS];        /* from x pixel in parent */
  char par_y[SHORT_CHARS];        /* from y pixel in parent */
}IHEAD;
```

Figure 2: The IHead 'C' programming language structure definition.

In the IHead format, images may be compressed with virtually any algorithm. Whether the image is compressed or not, the IHead is always uncompressed. This enables header interpretation and manipulation without the overhead of decompression. The **compress** field is an integer flag which signifies which compression technique, if any, has been applied to the raster image data which follows the header. If the compression code is zero, then the image data is not compressed, and the data dimensions: width, height, and depth, are sufficient to load the image into main memory. However, if the compression code is nonzero, then the **complen** field must be used in addition to the image's pixel dimensions. For example, the image described in Figure 3 has a compression

code of 6. This signifies that modified JPEG lossless compression has been applied to the image data. In order to load the compressed image data into main memory, the value in **complen** is used to determine the size of the compressed block of image data.

IMAGE FILE HEADER

Identity	: f0001_01.pct m 37
Header Size	: 288 (bytes)
Date Created	: Thu Sep 15 09:19:49 1994
Width	: 912 (pixels)
Height	: 1016 (pixels)
Bits per Pixel	: 8
Resolution	: 500 (ppi)
Compression	: 6 (code)
Compress Length	: 415293 (bytes)
Scan Alignment	: 8 (bits)
Image Data Unit	: 8 (bits)
Byte Order	: High-Low
MSBit	: First
Column Offset	: 0 (pixels)
White Pixel	: 255
Data Units	: Unsigned
Scan Order	: Row Major, Top to Bottom, Left to Right

Figure 3: The IHead values for the mugshot data file **f00001_1.pct**.

Once the compressed image data has been loaded into memory, JPEG lossless decompression can be used to produce an image which has the pixel dimensions consistent with those stored in its header. Using JPEG lossless compression and this compression scheme on the images in this database, an average compression ratio of 2.2 : 1 was achieved.

The attribute field, **align**, stores the alignment boundary to which scan lines of pixels are padded. Pixel values of 8-bit gray scale images are stored 1 pixel (or 8 bits) to a byte, so the images will automatically align to an even byte boundary.

The next three attribute fields identify data interchanging issues among heterogeneous computer architectures and displays. The **unitsize** field specifies how many contiguous bits are bundled into a single unit by the digitizer. The **sigbit** field specifies the order in which bits of significance are stored within each unit; most significant bit first or least significant bit first. The last of these three fields is the **byte_order** field. If **unitsize** is a multiple of bytes, then this field specifies the order in which bytes occur within the unit. Given these three attributes, data incompatibilities across computer hardware and data format assumptions within application software can be identified and effectively dealt with.

The **pix_offset** attribute defines a pixel displacement from the left edge of the raster image data to where a particular image's significant image information begins. The **whitepix** attribute defines the value assigned to the color white. For example, the gray scale image described in Figure 3 is gray print on a white background and the value of the white pixel is 255. This field is particularly useful to image display routines. The **issigned** field is required to specify whether the units of an image are signed or unsigned. This attribute determines whether an image with a pixel depth of 8, should have pixel values interpreted in the range of -128 to +127, or 0 to 255. The orientation of the raster scan may also vary among different digitizers. The attribute field, **rm_cm**, specifies whether the digitizer captured the image in row-major order or column-major order. Whether the scan lines of an image were accumulated from top to bottom, or bottom to top, is specified by the field, **tb_bt**, and whether left to right, or right to left, is specified by the field, **rl_lr**.

The final attributes in IHead provide a single historical link from the current image to its **parent** image. The images used in this database were mixed and renamed from their original filenames and the 'link' to the original filename was stored in the **parent** field. The **par_x** and **par_y** fields contain the origin, upper left hand corner pixel coordinate, from where the extraction took place from the parent image. These fields provide a historical thread through successive generations of images and subimages. We believe that the IHead image format contains the minimal amount of ancillary information required to successfully manage binary and gray scale images.

5.0 Database Organization and Content

NIST Special Database 18 contains a total of 3248 gray scale (8 bits/byte) mugshot images which are stored in the **data** directory (see Figure 4). The images, which use approximately 530 Megabytes of storage compressed and 1.2 Gigabytes of storage uncompressed, are distributed on an ISO-9660 formatted CD-ROM and compressed using a modified JPEG lossless compression algorithm. Decompression software and documentation are included with the mugshot data.

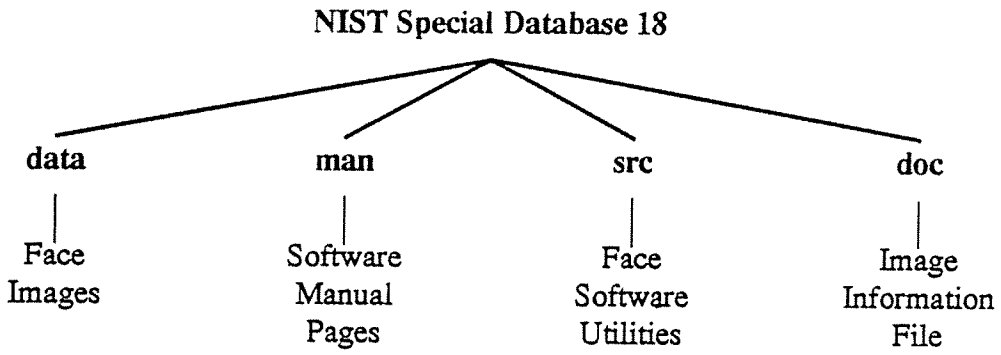

Figure 4: Top level directory tree for *NIST Special Database 18.*

The top level of the file structure contains four directories **src, man, data,** and **doc.** The code needed to decompress and use the image data is contained in the **src** directory with man pages for the source code stored in the **man** directories. Descriptions of the source code are given in section 6 of this document.

The **data** directories of discs 1-3 contain the mugshot images stored in a set of subdirectories as shown in figure 5. The directory **multiple** contains all of the cases for which there are two or more photos of an individual from the same view (front or profile). In the **multiple** directory: **fm_pm** contains all cases for which there are two or more images of both the front and profile views, **fm_p1** contains all cases with two or more images of the front view but only one image for the profile view, and **fm_p0** is all files with two or more front views and no profiles. The **single** directory is divided in much the same manner with: **f1_p1** containing all the images having only one front and one profile view, **f1_p0** is for one front view and no profile view, and **f0_p1** is no fronts and one profile. Since **f1_p1** contained 2434 images it was divided into subdirectories (**sing01-sing24**) with each containing 50 cases, except for the first (**sing01**-68 cases) and last (**sing24**-49 cases).

The file naming structure consist of a letter, a five digit number, a single digit number, and a ".pct" extension (figures 5 and 6 show some file names from the database). The letter is always an **f** (front) or **p** (profile) indicating what view of the individual is contained in the image. The profile could be a left or right profile. The five digit (for example **00001**) number immediately following the letter is a "case" sequence number; all files with the same case sequence number are different images of the same person. The final number is separated from the case sequence number by an "_" symbol. This final number is used to distinguish between multiple mugshots of the same case, with the mugshot of the person at their youngest age first (_1) and older age mugshots following

in order, by age (_2, _3, ...). If a front and profile have the same case number and the final digit is the same it means the images were taken at the same "sitting" with one pose being a front and one being a profile. Not all front views have an accompanying profile.

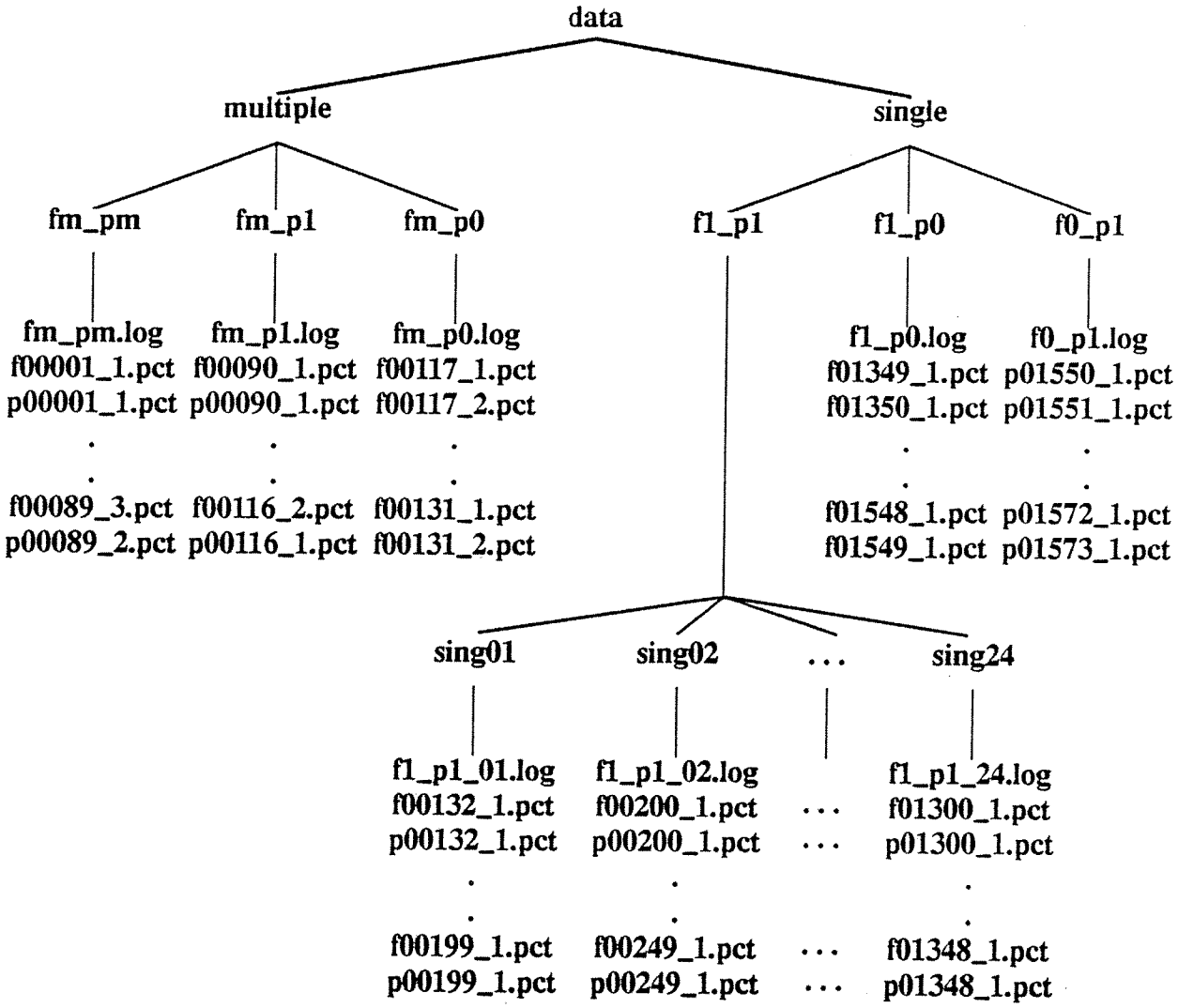

Figure 5: Arrangement of data files for *NIST Special Database 18.*

Looking at figure 5, disc 1 contains the **multiple** subdirectory and **sing01-sing04** of the **f1_p1** subdirectory contained in the **single** directory. Disc 2 contains **sing05-sing13** of the **f1_p1** subdirectory contained in the **single** directory. Disc 3 contains **sing14-sing24** of the **f1_p1** subdirectory contained in the **single** directory as well as the subdirectories **f1_p0** and **f0_p1** in the **single** directory.

The **doc** directory contains the file **info.log**, which has information about each file in the database. **Info.log** contains one line for each file in the database. The entry for each file is divided into four fields (see Figure 6). The first three fields contain the *filename*, *sex* (**m** or **f**), and *age* of the

individual respectively. If the letters **na** (not available) appear in the age field it means there wasn't enough information to calculate the age.The fourth field contains a list of filenames for all the other files, of the same view (i.e. front or profile), in the database which are images of the same person. Each filename is separated by a colon. For example, figure 6 shows that file **f00001_1.pct** has two matches in the database contained in files **f00001_2.pct** and **f00001_3.pct**. Notice in figure 6 that the fourth vertical line indicates the end of the entry for that file. A log file is also contained in each subdirectory with the same information as the **info.log** file, but only for the files in that subdirectory (see ***.log** files in figure 5)

```
f00001_1.pct | m | 37 | f00001_2.pct : f00001_3.pct |
p00001_1.pct | m | 37 | p00001_2.pct : p00001_3.pct |
f00001_2.pct | m | 41 | f00001_1.pct : f00001_3.pct |
p00001_2.pct | m | 41 | p00001_1.pct : p00001_3.pct |
f00001_3.pct | m | 42 | f00001_1.pct : f00001_2.pct |
p00001_3.pct | m | 42 | p00001_1.pct : p00001_2.pct |
```

Figure 6: Entry for first individual (case) in the **info.log** file.

The mugshot images vary in size because the original mugshot photographs vary in size from 1" to 2 1/2 inches in size. The camera scanned an image that was 1312 X 1016 pixels in size and rather than store an image that was more than 50% background (mostly uniform background) some of the background was discarded. This allowed more mugshots to be available on the database.

6.0 Software for Accessing Database

Included with the mugshot images are documentation and software written in the 'C' programming language. Four programs are included in the **src** directory: **dumpihdr, ihdr2sun, sunalign,** and **dcplljpg**. These routines are provided as an example to software developers of how IHead images can be manipulated and used. Descriptions of these programs and their subroutines are given below as well as in the included man pages located in the **man** directory. Copies of the manual pages are also included in Appendix A.

6.1 Compilation

The CD-ROM, on which *NIST Special Database 18* is provided, is a read only storage medium. The files in the **src** directory must be copied to a read-writable partition prior to compiling. After copying these files, executable binaries can be produced by invoking the UNIX utility **make** to execute the included makefile. An example of this command follows.

```
# make -f makefile.mak
```

6.2 Dumpihdr <Ihead file>

Dumpihdr is a program which reads an image's IHead data from the given file and formats the header data into a report which is printed to standard output. The report shown in Figure 3 was generated using this utility. The main routine for **dumpihdr** is found in the file **dumpihdr.c** and calls the external function **readihdr()**.

Readihdr() is a function responsible for loading an image's IHead data from a file into main memory. This routine allocates, reads, and returns the header information from an open image file in an initialized IHead structure. This function is found in the file **ihead.c**. The IHead structure definition is listed in Figure 2 and is found in the file **ihead.h**

6.3 Ihdr2sun <Ihead file>

Ihdr2sun converts an image from NIST IHead format to Sun rasterfile format. **Ihdr2sun** loads an IHead formatted image from a file into main memory and writes the raster data to a new file appending the data to a Sun rasterfile header. The main routine for this program is found in the file **ihdr2sun.c** and calls the external function **ReadIheadRaster()** which is found in the file **rasterio.c**.

ReadIheadRaster() is the procedure responsible for loading an IHead image from a file into main memory. This routine reads the image's header data returning an initialized IHead structure by calling **readihdr()**. In addition, the image's raster data is returned to the caller uncompressed. The images in this database have been 2-dimensionally compressed using a modified JPEG lossless compression algorithm, therefore **ReadIheadRaster()** invokes the external procedure **jpglldcp()** which is responsible for decompressing the raster data. Upon completion, **ReadIheadRaster()** returns an initialized IHead structure, the uncompressed raster data, the image's width and height in pixels, and pixel depth.

Jpglldcp() accepts image raster data compressed using the modified JPEG lossless compression algorithm and returns the uncompressed image raster data. **Jpglldcp()** was developed using techniques described in the WG10 "JPEG" (draft) standard [1] and adapted for use with this database. Source code for the algorithm is found in **jpglldcp.c**.

6.4 Dcplljpg <lossless JPEG compressed file>

Dcplljpg is a program which decompresses a face image file (approximately 12 seconds per image on a scientific workstation) that was compressed using the modified JPEG compression routine. The routine accepts a compressed image in NIST IHead format and writes the uncompressed image to the same filename using the NIST IHead format. The main routine is found in **dcplljpg.c** and calls the external functions **ReadIheadRaster()** (see section 6.3 for ReadIheadRaster description) and **writeihdrfile()**.

Writeihdrfile() is a routine that writes an IHead image into a file. This routine opens the passed filename and writes the given IHead structure and corresponding data to the file. **Writeihdrfile()** is found in the src file **rasterio.c**.

Acknowledgment

The author would like to acknowledge Mike Gilcrest, Charles Wilson, Mike McCabe and Remigius Onyshczak for input and assistance in this data collection.

References

[1] WG10 "JPEG", committee draft ISO/IEC CD 10198-1, "Digital Compression and Coding of Continuous-Tone Still Images," March 3, 1991.

[2] The camera used was a Kodak MegaPixel Model 1.4.

[3] M.D. Garris, "Design and Collection of a Handwriting Sample Image Database," *Social Science Computing Journal*, Vol. 10:196-214, 1992.

Appendix A: Manual Pages for Database Source Code

NAME

dcplljpg - non-standard JPEG lossless decompression for Ihead 8 bit gray scale images

SYNOPSIS

dcplljpg ihdrfile

DESCRIPTION

Dcplljpg takes an 8 bit gray scale ihead image, which was compressed using jpegcomp4, and decompresses it using techniques from the committee draft ISO/IEC CD 10198-1 for "Digital Compression and Coding of Continuous-tone Still images" with modifications to the draft image header.

NOTE: dcplljpg does not allow more than 8 bits/pixel input precision.

OPTIONS

ihdrfile

Any 8 bit gray scale ihead raster image (previously compressed using jpegcomp4).

EXAMPLES

dcplljpg foo.pct

FILES

ihead.h	NIST's raster header include file
jpeg.h	Include file for jpeg algorithm

SEE ALSO

dumpihdr(1), ihdr2sun(1), ReadIheadRaster(3), writeihdrfile(3)

DIAGNOSTICS

dcplljpg exits with a status of -1 if an error occurs.

BUGS

dcplljpg only handles gray scale images up to 8 bits per pixel precision.

NAME

dumpihdr - takes a NIST IHead image file and prints its header content to stdout

SYNOPSIS

dumpihdr ihdrfile

DESCRIPTION

Dumpihdr opens a NIST IHead rasterfile and formats and prints its header contents to stdout.

OPTIONS

ihdrfile

any NIST IHead image file name

EXAMPLES

dumpihdr foo.pct

FILES

ihead.h NIST's raster header include file

SEE ALSO

ihdr2sun(1), ReadIheadRaster(3), writeihdrfile(3), writeihdr(3), readihdr(3), printihdr(3)

DIAGNOSTICS

Dumpihdr exits with a status of -1 if opening ihdrfile fails.

BUGS

NAME

ihdr2sun - takes a NIST ihead image and converts it to a Sun rasterfile

SYNOPSIS

ihdr2sun [-o outfile] ihdrfile [mapfile]

DESCRIPTION

Ihdr2sun converts a NIST ihead rasterfile to a Sun rasterfile. If the optional argument mapfile is included on the command line and the input image is multiple bitplane, the colormap in mapfile will be inserted into the Sun rasterfile, otherwise a default colormap gray.map will be used when necessary. The Sun image file created will have the root name of ihdrfile with the extension .ras appended, unless an alternate outfile is specified.

OPTIONS

ihdrfile

any ihead raster image

mapfile

optional colormap file

EXAMPLES

ihdr2sun foo.pct gray.map

FILES

/usr/include/rasterfile.h sun's raster header include file

ihead.h NIST's raster header include file

SEE ALSO

dumpidhr(1), sunalign(1), rasterfile(5)

DIAGNOSTICS

Ihdr2sun exits with a status of -1 if opening ihdrfile fails.

BUGS

Ihdr2sun does not currently support multiple bit levels per pixel other than depth 8.

NAME

sunalign - takes a sun rasterfile and word aligns its scan-lines

SYNOPSIS

sunalign sunrasterfile

DESCRIPTION

Sunalign takes the file sunrasterfile and determines if the stored scan lines in the file require word alignment. If so, the command overwrites the image data making scan lines word aligned. This command is useful when taking clipped images from the HP Scan Jet and importing them into Frame Maker.

OPTIONS

sunrasterfile

any sun rasterfile image

EXAMPLES

sunalign foo.ras

FILES

/usr/include/rasterfile.h

sun's raster header include file

SEE ALSO

rasterfile(5)

DIAGNOSTICS

Sunalign exits with a status of -1 if opening sunrasterfile fails.

BUGS

NAME

jpglldcp - takes a JPEG lossless compressed input data buffer (with modified data header) and writes the uncompressed data to the passed output buffer

SYNOPSIS

```
void jpglldcp(indata, width, height, depth, outbuffer)
unsigned char *indata, *outbuffer;
int width, height, depth;
```

DESCRIPTION

jpglldcp() takes the input buffer indata and decompresses it writing the uncompressed data into the output buffer outbuffer with length equal to the original image dimensions given. This procedure was developed using techniques from the committe draft ISO/IEC CD 10198-1 for "Digital Compression and Coding of Continuous-tone Still Images" with modifications to the draft image header. The source is found in the source code file jpglldcp.c.

indata
- the compressed data input buffer

width
- the pixel width of the image from which the input data came

height
- the pixel height of the image from which the input data came

depth
- the pixel depth of the image from which the input data came

outbuffer
- the output buffer in which the uncompressed data is to be returned

SEE ALSO

dcpl1jpg(1), ReadIheadRaster(3), writeihdrfile(3)

BUGS

NOTE: jpglldcp will only work with gray-scale images that were compressed using a modified data header (not the standard lossless JPEG data header).

NAME

printihdr - prints an ihead structure to the passed file pointer

SYNOPSIS

```
#include <ihead.h>

printihdr(head, fp)
IHEAD *ihead;
FILE *fp;
```

DESCRIPTION

Printihdr() takes a pointer to an ihead structure and prints the ihead structure to the file pointed to by fp. The source is found in the source code file ihead.c.

fp - an open file pointer

ihead
- a pointer to an initialized ihead structure

SEE ALSO

writeihdrfile(3), writeihdr(3), readihdr(3), ReadIheadRaster(3), dumpihdr(1)

BUGS

NAME

ReadIheadRaster - loads into memory an ihead structure and corresponding image data from a file

SYNOPSIS

```
#include <ihead.h>

ReadIheadRaster(file, head, data, width, height, depth)
char *file;
IHEAD **head;
unsigned char **data;
int *width, *height, *depth;
```

DESCRIPTION

ReadIheadRaster() opens a file named file and allocates and loads into memory an ihead structure and its corresponding raster image data. If the image data is compressed, ReadIheadRaster will uncompress the data before returning the data buffer. This routine also returns several integers converted from their corresponding ASCII entries found in the header. The source is found in the source code file rasterio.c.

file - the name of the file to be read from

head - a pointer to where an ihead structure is to be allocated and loaded

data - a pointer to where the array of binary raster image data is to be allocated and loaded

width
- integer pointer containing the image's pixel width upon return

height
- integer pointer containing the image's pixel height upon return

depth
- integer pointer containing the image's Bits Per Pixel upon return

SEE ALSO

printihdr(3), readihdr(3), writeihdrfile(3), writeihdr(3), dumpihdr(1)

DIAGNOSTICS

ReadIheadRaster() exits with -1 when opening file fails.

BUGS

NAME

readihdr - allocates and reads header information into an ihead structure and returns the initialized structure

SYNOPSIS

```
#include <ihead.h>

readihdr(fp)
FILE *fp;
```

DESCRIPTION

Readihdr() takes a file pointer to an ihead structured file. Then allocates and reads the header information from the file into an ihead structure. The source is found in the source code file ihead.c.

fp - an open file pointer

SEE ALSO

ReadIheadRaster(3), writeihdrfile(3), printihdr(3), writeihdr(3), dumpihdr(1)

BUGS

NAME

writeihdr - writes an ihead structure to an open file

SYNOPSIS

```
#include <ihead.h>

writeihdr(fp, ihead)
FILE *fp;
IHEAD *ihead;
```

DESCRIPTION

Writeihdr() takes a pointer to an ihead structure and writes it to the open file pointed to by fp. The source is found in the source code file ihead.c.

fp - an open file pointer

ihead
- a pointer to an initialized ihead structure

SEE ALSO

writeihdrfile(3), printihdr(3), readihdr(3), ReadIheadRaster(3), dumpihdr(1)

BUGS

NAME

writeihdrfile - writes an ihead structure and corresponding image data to a file

SYNOPSIS

```
#include <ihead.h>

writeihdrfile(file, head, data)
char *file;
IHEAD *head;
unsigned char *data;
```

DESCRIPTION

Writeihdrfile() opens a file name file and writes an ihead structure and its corresponding image data to it. The source is found in the source code file rasterio.c.

file - the name of the file to be created

head - a pointer to an initialized ihead structure

data - the array of raster image data

SEE ALSO

writeihdr(3), printihdr(3), ReadIheadRaster(3), readihdr(3), dumpihdr(1)

DIAGNOSTICS

Writeihdrfile() exits with -1 when opening file fails.

BUGS