(1)

(14) TR-810 (11) October 1979

DAAG-53-76C-0138

Interfaces, Subroutines, and Programs for the Grinnell GMR-27 Display Processor on a PDP-11/45 with the UNIX Operating System.

(10) Robert L. Kirby, Russ Smith, Philip A. Dondes, Sanjay Ranade, Les Kitchen, and Fred Blonder

Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, Maryland 20742

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

14 TR-810 11 October. 1979 12 172

DAAG-53-76C-0138

6

Interfaces, Subroutines, and Programs for the Grinnell GMR-27 Display Processor on a PDP-11/45 with the UNIX Operating System.

10

Robert L. /Kirby, Russ/Smith,
Philip A. /Dondes, Sanjay/Ranade,
Les/Kitchen, and Fred Blonder

Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, Maryland 20742

9 Technical rept.,

ABSTRACT

15 DAAG53-76-C-φ138, VV DARPA Order-32φ6

The specialized device interfaces for the University of Maryland Computer Vision Laboratory image acquisition and display equipment extend the capabilities of a PDP-11/45 hosting the UNIX operating system. The devices include the Grinnell GMR-27 color display processor, the other Computer Vision Laboratory display and scanning equipment, and the Digi-Data TM-11/TU-16 compatible tape drive. Subroutine packages give easy access to the interfaces from user programs, allowing full use of the special features. Programs using these subroutine packages and the well-designed UNIX operating system provide a flexible and powerful environment for image processing and program development. Short descriptions of these interfaces, subroutines, and programs are given for program writers and other users.

The support of the U. S. Army Night Vision Laboratory and the Defense Advanced Project Agency under Contract DAAG-53-76C-0138 (DARPA Order 3206) is gratefully acknowledged. Lee Moore, Marshall Schaffer, Donald J. Gerson, Gyorgy Fekete, and James W. Williams contributed articles to this manual as part of a class project. Wallace S. Rutkowski also contributed program documentation. This manual was produced by them and the authors using the UNIX "nroff" text processor.

page -A-

411074 LB

CONTENTS

I. INTRODUCTION

The specialized device interfaces for the University of Maryland Computer Vision Laboratory image acquisition and display equipment extend the capabilities of a PDP-11/45 hosting the UNIX operating system. The devices include the Grinnell GMR-27 color display processor, the other Computer Vision Laboratory display and scanning equipment, and the Digi-Data TM-11/TU-16 compatible tape drive. Subroutine packages give easy access to the interfaces from user programs, allowing full use of the special features. Programs using these subroutine packages and the well-designed UNIX operating system provide a flexible and powerful environment for image processing and program development. Short descriptions of these interfaces, subroutines, and programs are given for program writers and other users.

This document has been produced with the UNIX "nroff" text processor so that it may be more easily updated. It consists of short descriptions in a format approximating that of the manual sections that are provided in the UNIX operating system user manual. References made by capital roman numerals in parentheses refer to sections within that manual. However, the normal ordering and grouping of these sections has been altered for the current presentation into interfaces, subroutines, and programs.

These descriptions are intended as cursory reference material for those who are actively using the system at the University of Maryland Computer Vision Laboratory. Tutorial and in-depth descriptions of the UNIX operating system are available elsewhere. Furthermore, these descriptions represent only an approximation to the actual behavior. Programs may change or bugs may exist. These descriptions are guidelines at best; more accurate versions may be found on-line.

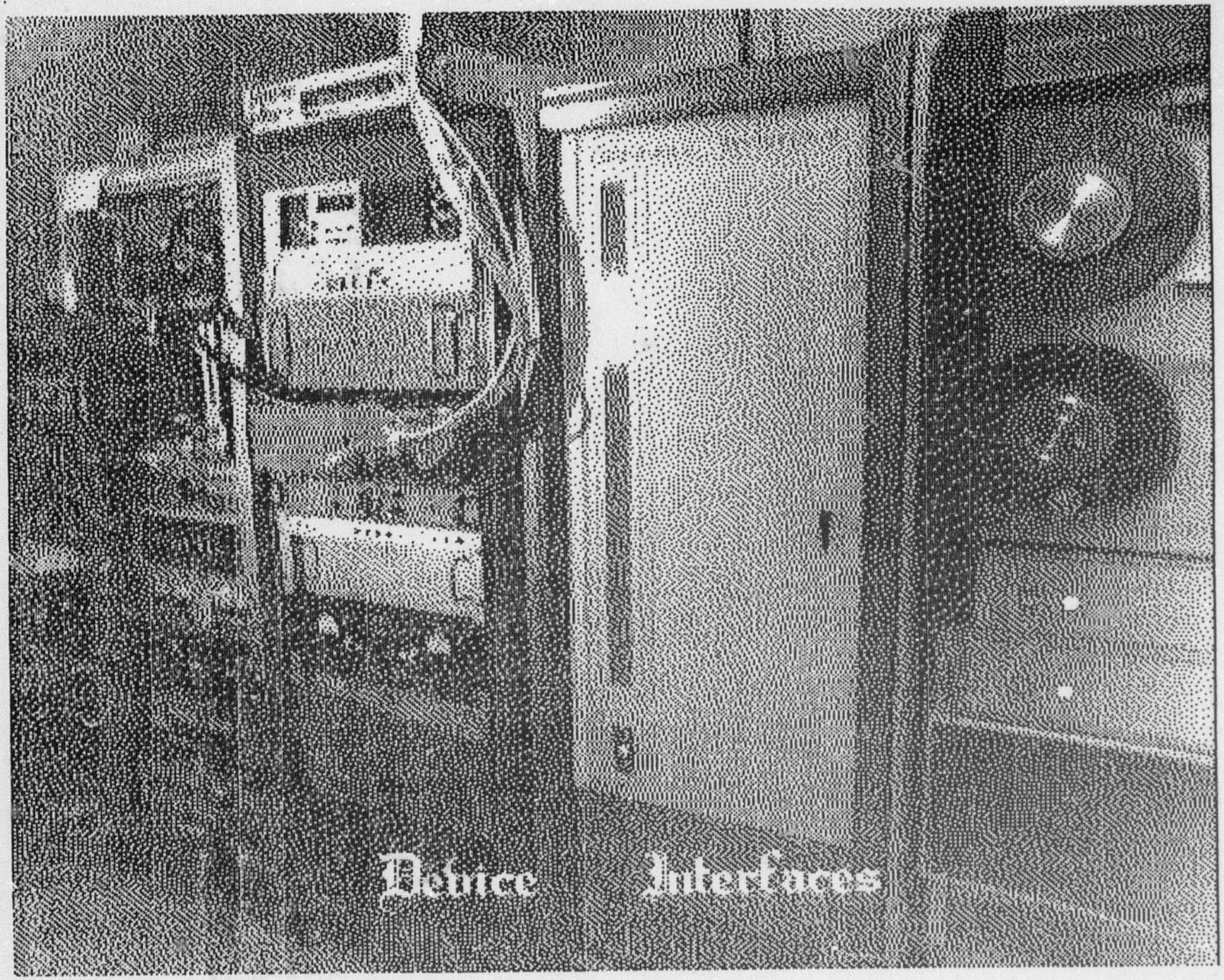
The older display equipment is connected to the CPU through a DR-11B direct memory access (DMA) interface. Under CPU control, it can produce high-quality, hard-copy 6 bit images on Polaroid film, film strips, or slides, and can display images on a black-and-white monitor.

The newer Grinnell GMR-27 color display processor also operates through a DR-11B DMA interface. The system includes the display processor and memory, a track ball and switches, color monitors, and a black-and-white TV camera. The memory consists of a 512 by 512 array of 13 bit pixels. 12 bits can display 4 bits of each of the three colors: blue, green, and red. The 13th bit displays as a white overlay. The high order 8 bits of the 12 color display bits can also be displayed as black-and-white-only grayscale images. When using the TV camera, only the bottom 480 rows are actually displayed and only 6 bits of grayscale may be acquired from any one frame. However, frame averaging can produce additional bits of input accuracy. A read/write, random-access, 4096 entry table in the display processor can display any 12 bit number in the memory as any other 12 bit number without altering the memory contents. The display processor also supports cursor positioning, cursor readback, memory readback memory writing, alphanumeric graphics, vector graphics, and internal test patterns. Channel and subchannel masks allow only selected bits of each pixel to be either read or altered.

The tape driver has been extended to correct some of the long-standing and well-known deficiencies of the original UNIX operating system. A description of this extended tape driver is included since such a large portion of image processing in this system seems to involve tape handling.

Subroutines provide a more convenient programmer interface to the hardware capabilities. The CXAP subroutine package provides a C-language callable interface that is similar to the FORTRAN callable XAP package originally written for the UNIVAC 1100 series machines. The Grinnell Application Package (GAP) allows a complete C-callable interface to the Grinnell processor. Other subroutines have also been written for specialized image processing uses. Using these subroutines, students without previous experience in either the C language or the UNIX operating system have rapidly written C programs to perform image processing. Descriptions of some of these programs have been included. Some utility programs are also described so that the novice user, after logging onto the system, may easily evoke the available image processing and graphics routines. Finally, a section describes some quadtree manipulation programs that use the Grinnell processor.

The section dividers within this manual were produced using some of the described capabilities.



Device Interfaces

NAME

dr - DR11-B Direct Memory Access Device

DESCRIPTION

This file refers to the DR11-B as used at the Computer Vision Laboratory of the University of Maryland.

The *rdr0* (or *dr0*) file is used to output pictorial information to the scan conversion memory for display purposes.

Explanation on how to set up the device to receive image data from the PDP11/45 is beyond the scope of this description and should be sought from someone familiar with the procedure.

When the file is opened, a RESET command is sent to the device. Upon closing of the file, a DISPLAY command will be sent.

This being a *display* device, the user is restricted to only writing on the file. Seeking in the forward direction is totally ignored. Seeking in the reverse direction causes a RESET command to be sent to the device (Thus resetting the device counters and disabling the scanner until the RESET button on the device itself is pushed).

Caution must be taken to set up the device switches themselves correctly. If they are incorrect, in some situations the call on the device will sleep with an un'kill'able priority until they *are* correct, thus causing untold frustration and anxiety.

To use the scanner correctly, each pixel should be left justified in the byte written out. (For example, a 64-greylevel picture should have each pixel's value shifted left 2 bits (multiplied by 4).)

About four seconds after no data has been sent to the scanner, it will display what has been sent. This does not disable the scanner from receiving more data from your program. It is only a convenience feature with no side effects.

FILES

/dev/rdr0, /dev/dr0

BUGS

The switches for the number of columns on the scan device must be set to one less than the actual number of columns sent from the PDP11/45. The picture will still all come out. Care must be taken to write out your picture to the scanner in even sized chunks, odd byte counts causing an immediate system error. This means, of course, that it is impossible to send out an image that has an odd number of rows *and* an odd number of columns.

NAME

gr - DR-11 interface to Grinnell display processor GMR-27

DESCRIPTION

Gr refers to the modified DR-11B interface to the Grinnell display processor, GMR-27. The interface supports both synchronous and asynchronous I/O as a character device only. Synchronous requests are satisfied in a limited time. Asynchronous requests may be delayed until some external condition is satified, such as the CURSOR ENTER button being pressed. In synchronous I/O, the buffer *must* begin on a word boundary and the count *must* be even. Seek calls for the file *gr* are meaningless.

Each word written to the file *gr* commands the GMR-27 to perform one operation (except when using packed bytes). Each write is done as a single DMA transfer.

The interface performs synchronous reading in a non-standard manner. The buffer used to synchronously read words from the GMR-27 must begin with command words which are first sent to the GMR-27 to enable reading. The last of these command words must be readback peripheral data (RPD), 0160000 (octal). All words following the first RPD in the buffer receive data words from the GMR-27. If no RPD command resides in the buffer or the first RPD is the last word in the buffer, *read* returns an error condition without sending any commands to the GMR-27. The byte count returned from a synchronous *read* includes the initial command words which were written to the GMR-27. In order to protect UNIX from being stalled by an improperly programmed synchronous read command performing asynchronous I/O or using nonexistant GMR-27 peripheral devices, the interface limits the duration of synchronous reading. If the GMR-27 does not complete a synchronous *read* request promptly, the interface interrupts the DMA transfer and returns a shortened *read* byte count. Such an incomplete *read* count should be considered as an error. To avoid unnecessarily tying up UNIX, processes should not use synchronous *read* calls which may need to be interrupted.

Processes may also wait for asynchronous events by reading with a byte count of exactly two. In order to intercept GMR-27 asynchronous events, the interface enables the GMR-27 "interrupt" peripheral device with a select peripheral device (SPD), 0122000 (octal), issues an RPD command, and requests a data word transfer from the GMR-27. Waiting for an asynchronous event, the GMR-27 delays returning a data word until a user either presses the *enter* button or moves the trackball with the *track* switch on. Between synchronous I/O requests, until the interface receives an "interrupt" word, the interface waits in a background mode for an asynchronous event. After an asynchronous event, all waiting processes receive the "interrupt" word. After an asynchronous event, synchronous reads can elicit the cause of the event and reset the interrupt flags within the GMR-27.

Since multiple opens are permitted, the state of the GMR-27 may unexpectedly change between the I/O requests of any one process. Also, the background asynchronous I/O processing may alter the selection of peripheral devices with an SPD after each synchronous I/O request. Thus, each I/O request is responsible for reestablishing the volatile state of the GMR-27 whenever simultaneous operations could occur. Some volatile elements of the GMR-27 state which may need to be reset include:

1) Display channel (LDC) and subchannel mask (LSM) selection,
2) Write (LWM) and update (LUM) modes,
3) Element (LER, LEA, LEB, and LEC) and line (LLR, LLA, LLB, and LLC) registers, and
4) Selection (SPD) and use (LPR, LPA, and LPD) of some GMR-27 peripherals.

The volatile GMR-27 peripheral devices include:

1) Memory readback,
2) Independent cursor locations and flags, and
3) Byte unpacking.

To avoid interference with other I/O, byte unpacking should be completed by single I/O transfers.

Other GMR-27 peripheral devices which have global effects are manipulated less often. Since all users should agree to their invocation, these devices and commands need not be reset or reissued with each I/O request:

1) Graphic digitizer (camera input),
2) Video control (color vs grayscale),
3) Video lookup table,
4) Internal self-tests,
5) Screen and line erasure, and
6) Scrolling.

With simultaneous users, by convention, each user should insure that only allocated GMR-27 display memory is altered. Mutual user consent should be obtained before changing global GMR-27 states.

FILES

/dev/gr

SEE ALSO

grdefs(VII)
Grinnell Systems, GMR-27 User's Manual.

BUGS

The DR-11B interface hardware must be modified so that stalled DMA transfers may be safely terminated. This modification, which connects one of the unassigned function bits to ATTN, must be made before reading is attempted.

The driver could establish a starting image position before each transfer (in GMR-27 registers Ec and Lc perhaps).

NAME

grdefs - Grinnell display processor GMR-27 definitions

SYNOPSIS

#*include* /usr/lib/grdefs.c

#*include* /usr/lib/grdefs.f

DESCRIPTION

The file *grdefs* provides command mnemonic definitions for the Grinnell display processor GMR-27 as implemented at the University of Maryland.

Mnemonics specify each GMR-27 command and bit patterns used to evoke options within each command. Command sequences for each peripheral summarize the available peripheral device features. The format of data words supplied by the readback peripheral data (RPD) command is provided after the RPD command of appropriate peripheral devices. A standard header gives the needed format for raster outputting rectangular images.

In the following summary, A's and D's represent addresses and data, respectively. CL is used to specify a cursor coordinate name for the independent cursor peripheral device. In CL, C is 0 or 1 to specify the first or second cursor (only two out of four may be manipulated). L is 0 or 1 to specify an element (column) or line (row) coordinate, respectively.

FILES

/usr/lib/grdefs.c	for c
/usr/lib/grdefs.f	for Fortran

SEE ALSO

Grinnell Systems, *GMR-27 User's Manual*.
gr(IV)

AUTHOR

Robert L. Kirby

BUGS

Other definition files are also in use.

COMMAND SUMMARY

Value (octal)	Mnemonic	Value (bit)	Name
00dddd	WID	0 000 DDD DDD DDD DDD	Write Image Data
01mmmm	LSM	0 001 MMM MMM MMM MMM	Load Subchannel Mask
020ddd	WGD	0 010 00x xDD DDD DDD	Write Graphic Data (left to right 8 bits)
022ddd	WAC	0 010 01x x0D DDD DDD	Write Alphanumeric Character (7-bit ASCII upper case only)
0240mm	LWM	0 010 10x xBA ZVH WCC	Load Write Mode
000200	LIGHT	0 000 000 010 000 000	Light background (reversed vs dark)
000100	ADDITV	0 000 000 001 000 000	Additive graphics
000040	ZEROW	0 000 000 000 100 000	Zero Write (must use)
000020	VECTOR	0 000 000 000 010 000	Vector graphics
000010	DHGHT	0 000 000 000 001 000	Double Height
000004	DWDTH	0 000 000 000 000 100	Double Width
000002	CURPOS	0 000 000 000 000 010	Sum for cursor position
000001	VCURSOR	0 000 000 000 000 001	Visible cursor
0260mm	LUM	0 010 11x xxx SSL LEE	Load Update Mode
000001	EC	0 000 000 000 000 001	Ea := Ec
000002	EBA	0 000 000 000 000 010	Ea := Ea + Eb
000003	ECA	0 000 000 000 000 011	Ea := Ea + Ec
000004	LC	0 000 000 000 000 100	La := Lc
000010	LBA	0 000 000 000 001 000	La := La + Lb
000014	LCA	0 000 000 000 001 100	La := La + Lc
000020	SHOME	0 000 000 000 010 000	Home scroll
000040	SDOWN	0 000 000 000 100 000	scroll down
000060	SUP	0 000 000 000 110 000	scroll up
030000	ERS	0 011 00x xxx xxx xxx	Erase (entire screen)
032000	ERL	0 011 01x xxx xxx xxx	Erase Line
034Imm	SLU	0 011 10x xxI SSL LEE	Special Location Update (see LUM for SSL LEE)
000100	SINHBT	0 000 000 001 000 000	Inhibit scroll timing
036000	EGW	0 011 11x xxx xxx xxx	Execute Graphic Write
002000	GWRITE	0 000 010 000 000 000	Execute graphic bit, W write after loading register in following
040aaa	LER	0 100 0Wx AAA AAA AAA	Load Ea Relative
044aaa	LEA	0 100 1Wx AAA AAA AAA	Load Ea
050aaa	LEB	0 101 0Wx AAA AAA AAA	Load Eb
054aaa	LEC	0 101 1Wx AAA AAA AAA	Load Ec
060aaa	LLR	0 110 0Wx AAA AAA AAA	Load La Relative
064aaa	LLA	0 110 1Wx AAA AAA AAA	Load La
070aaa	LLB	0 111 0Wx AAA AAA AAA	Load Lb
074aaa	LLC	0 111 1Wx AAA AAA AAA	Load Lc

```
10000c  LDC     1 000 xxx xxx xxx CCC   Load Display Channels
000003  IMAGECH 0 000 000 000 000 011   Image display channels
                                        bits(11-8) and (7-0)
000004  OVERLAY 0 000 000 000 000 100   Overlay display channel
                                        for white overlay

110000  NOP     1 001 xxx xxx xxx xxx   No Operation
12pppp  SPD     1 010 PPP PPP PPP PPP   Select Peripheral Device
13aaaa  LPA     1 011 AAA AAA AAA AAA   Load Peripheral Address
14dddd  LPR     1 100 DDD DDD DDD DDD   Load Peripheral Register
15dddd  LPD     1 101 DDD DDD DDD DDD   Load Peripheral Data
160000  RPD     1 110 xxx xxx xxx xxx   Readback Peripheral Data
170000  NON     1 111 xxx xxx xxx xxx   No Operation
```

PERIPHERAL DEVICE CONTROLS

Camera Digitizer control

```
120002  DIGITZ  1 010 000 000 000 010   Select Peripheral Device
1400ip  LPR     1 100 000 000 IIO PPP   II=camera selection=00
                                        (PPP<8)=shift down count
15cddd  LPD     1 101 Cxx xDD DDD DDD   Camera digitizing mode
004000  CNTUOUS 0 000 100 000 000 000   Continuous input with
                                        D>1 is averaging count,
                                        D=0 or C=0 single frame
```

Independent Cursor control

```
120004  CURSOR  1 010 000 000 000 100   Select Peripheral Device
14000d  LPR     1 100 000 000 000 ODD   Display cursors (D=1=on)
                                        Each bit for a cursor
14400d  CWHITE  1 100 100 000 000 ODD   D=1 for white, D=0 black
13000c  LPA     1 011 000 000 000 OCL   to address coordinate
                                        CL=selected cursor name
150ddd  LPD     1 101 Oxx DDD DDD DDD   Move cursor relatively
154ddd  ABSMOV  1 101 1xx DDD DDD DDD   Move cursor absolutely
160000  RPD     1 110 xxx xxx xxx xxx   Read cursor positions
                                        Reading line resets flag
OfCaaa  data    0 OOF OCL AAA AAA AAA   F=1 if enter flag is on,
                                        CL=selected cursor name
Use read count of 2 to wait for cursor flag events.
Device /dev/gr uses an SPD(122000) and RPD(160000)
to obtain the following data word:
000004  data    0 000 000 000 000 100   Cursor interrupt word
```

Special Video Control

```
120020  VCNTRL  1 010 000 000 010 000   Select Peripheral Device
14000g  LPR     1 100 000 000 000 00G   G=1 greyscale (bits 11-4)
                                        G=0 color (blue 11-8,
                                        green 7-4, red 3-0)
```

Video Lookup table

```
120040  LOOKUP  1 010 000 000 100 000   Select Peripheral Device
13aaaa  LPA     1 011 AAA AAA AAA AAA   Load table address
15dddd  LPD     1 101 DDD DDD DDD DDD   Put data in table
14000b  LPR     1 100 000 000 000 00B   B=1 bypass lookup table
160000  RPD     1 110 xxx xxx xxx xxx   Readback lookup table
10dddd  data    1 000 DDD DDD DDD DDD   Lookup table value
```

Memory Readback

```
120400   MEMORY   1 010 000 100 000 000   Select Peripheral Device
160000   RPD      1 110 xxx xxx xxx xxx   Readback memory channel
00dddd   data     0 000 DDD DDD DDD DDD   Selected memory value
```

Byte Unpacking

```
121000   UNPACK   1 010 001 000 000 000   Select Peripheral Device
001000   SWTBYT   0 000 001 000 000 000   Switch bytes of word
Below, SWTBYT default is on (for least significant byte first).
000400   ODDBYT   0 000 000 100 000 000   Ignore last byte of last
                                          word in following (Y bit)
141ddd   IBYTES   1 100 0x1 YDD DDD DDD   D=pairs of image points
145ddd   GBYTES   1 100 101 YDD DDD DDD   D=words of graphic data
147ddd   ABYTES   1 100 111 YDD DDD DDD   D=pairs of alphanumerics
DD DDD DDD pairs of bytes follow the above LPR commands.
```

Internal Self-test

```
124000   ITESTS   1 010 100 000 000 000   Select Peripheral Device
13000d   LPA      1 011 000 000 000 0DD   DD = test number - 1
```

NAME

tm - TM-11/TU-10 magtape interface with filespacing

DESCRIPTION

The files *mt[0-7] refer to the DEC TU10/TM11 magtape. File names with the same numerical suffix operate the same physical drive but with different opening and closing actions. On DIGI-DATA (trademark) drives file numbers with the high order bit (04) set refer to the same physical drive but at 1600FPI density instead of 800FPI. Thus files 0 and 4 refer to the same DIGI-DATA drive but at 800FPI and 1600FPI densities respectively.

Each standard (cooked) file on tape starts just after a file mark or the beginning of tape (BOT), consists of a series of 512 byte records, and terminates with at least one end-of-file mark (EOF). To some extent, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create monstrous record gaps. On a cooked device that is opened for writing, reading an EOF returns a 512 byte block of zeros to allow the operating systems read before write mechanism to extend tapes with small writes. Otherwise reading an EOF returns an error status from the cooked devices.

Files opened for writing whose last action was not a read write three file marks either when closing or before seeking backward. The files mt[0-7] stay at their current position when opened and when closed return to their starting position. The files bu mt[0-7] backup over the current file mark to a position just after the previous file mark or BOT when opened. When closed, they also return to their starting position. The files rw mt[0-7] stay at their current position when opened and rewind to BOT when closed. The files nrw mt[0-7] also stay at their current position when opened. When closed after reading, these files advance the tape to just past the next file mark, ready to read the next file. Hence, to skip a file, simply open the non-rewinding device for reading and then close it. After writing file marks when closing, the tape is positioned immediately after the first file mark, ready to extend the tape by writing yet another file.

The mt files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the ``raw'' interface is appropriate. The files corresponding to *mt[0-7] are named *rmt[0-7]. Each read or write call reads or writes the next physical record on the tape. In the write case the physical record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size; if the physical record is longer than the buffer, the first part of the data fills the buffer and an error is indicated. In raw tape I/O, the buffer *must* begin on a word boundary. However, the buffer length may be odd or even. Seeks are allowed on the raw device. Each physical record is treated internally as though it were exactly 512 bytes (including tape marks), so that a seek-by-blocks system call will actually seek by records. During a seek on the raw device, tape marks are counted as one block. On the cooked device, an error will be returned if the seek tries to cross a tape mark. On the raw device, an error is only returned if the seek tries to cross a double tape mark, leaving the tape positioned after the first of the pair.

When the non-rewinding raw files *nrw rmt*[*0*-*7*] have just read a file mark in reaching the current position, other than any EOF that may precede the entire file, these files do not advance any further when closed. A *seek* just before closing can redefine the current position.

An end-of-file may be written on the raw devices by specifying a write buffer length of zero. Such a file mark also counts as 512 bytes when repositioning the file.

Signals may interrupt the tape open procedure so that a processes may be immediately terminated without waiting for the drive to complete tape movement. However, the device will remain occupied until closed. Although after an interrupted opening or I/O errors from the previous close, rewind and skip forward actions do not occur when the tape is closed, the device continues with any backward prepositioning. On the raw device, a wait for tape seek operations may also be interrupted prior to the actual data transfer. In this case, the seek operation is completed but the device does not transfer data and the file pointer is not advanced.

The non-rewinding files *nrw mt*[*0*-*7*] and *nrw rmt*[*0*-*7*] are also named *smt*[*0*-*7*] and *srmt*[*0*-*7*].

DIAGNOSTICS

Except as noted above, when the cooked device tries to cross a file mark or the raw device tries to pass a pair of file marks the error number EFBIG (27) is returned to the *read* request.

Only one file corresponding to a physical drive may be open. Attempts to multiply open such files produce error ENXIO. If a physical error occurs during opening from the actions of a previous close operation or an attempt to backspace before opening, the pending open also returns error status ENXIO. In particular, an attempt to open a backup file such as *by rmt0* when the device is already positioned at the beginning of tape, is such an error. File skipping to the hardware end of tape also generates this error and backs up the tape.

FILES

/dev/mt?
/dev/rmt?
/dev/bu_mt?
/dev/bu_rmt?
/dev/rw_mt?
/dev/rw_rmt?
/dev/nrw_mt?
/dev/nrw_rmt?
/dev/smt?
/dev/srmt?

SEE ALSO

bu (I), skp (I), rewind (I), eot (I), tp (I), dd (I), seek (II)

AUTHOR

Robert L. Kirby

BUGS

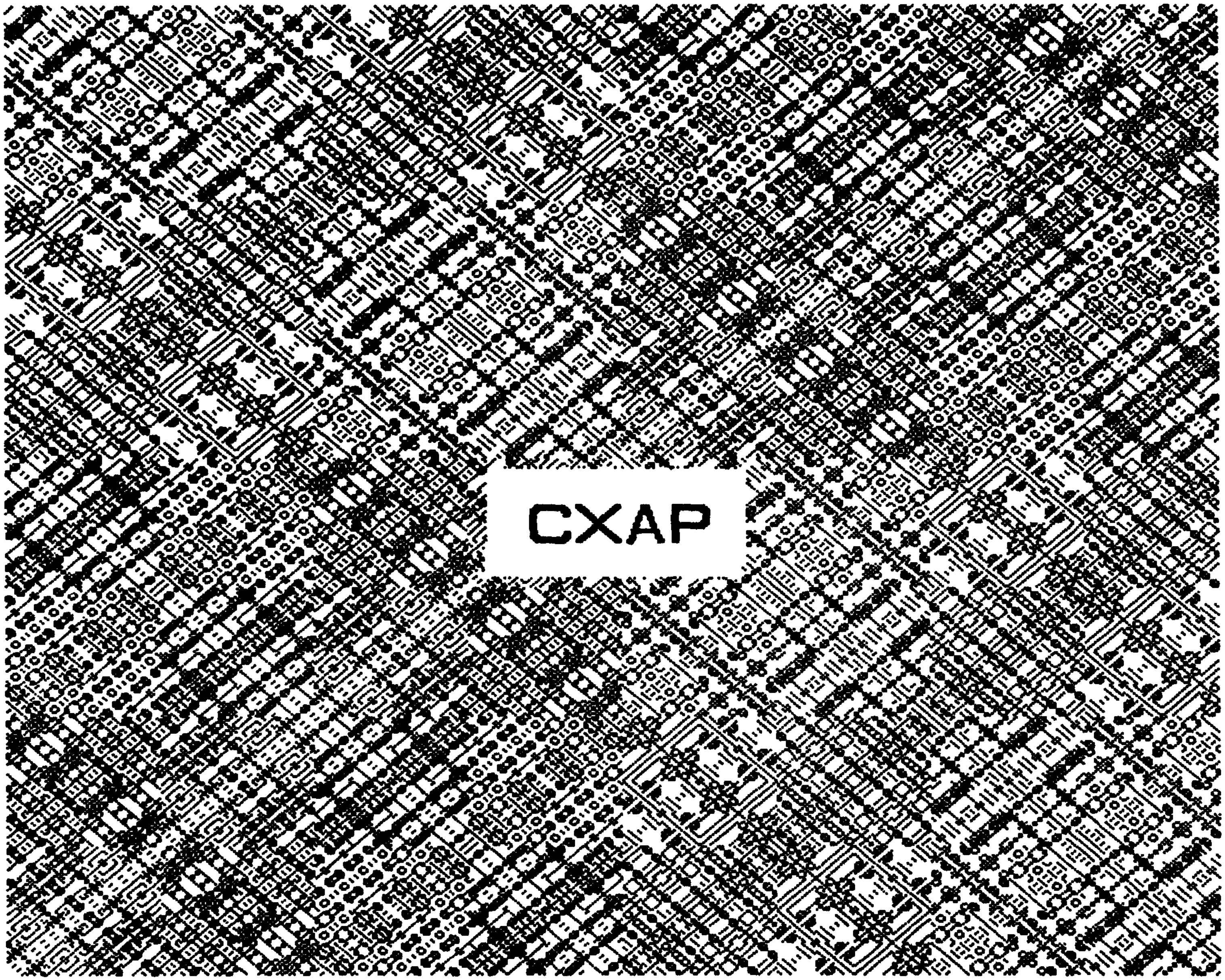
If any non-data error is encountered, it refuses to do anything more until closed. After non-data errors the tape may lose position. Such errors can be generated by using the wrong tape density or from attempting to read a virgin portion of the tape.

At the expense of being larger, this handler corrects several difficulties with the Harvard tape handler. When closing the non-rewinding files processing must waiting for the forward seek to complete. Often this creates an unkillable process that excessively ties up a terminal. In raw I/O, there should be a way to perform backward file spacing. After specifying tape repositioning with a seek on the raw files, the next I/O request locks the requesting process in core while moving the tape. This can prevent other processes from using the CPU for an unnecessarily long period. A write buffer length of 2 bytes is converted to a length of 4 bytes to avoid conflicts with the writing of file marks. However, the user process is only told that 2 bytes were written. This extended buffer must fit within the user's area. When writing a file mark, the buffer address must be on a word boundary within the user's assigned area.

As originally distributed, unmodified tape drivers round up the bytes read to an even number, do not support forward file spacing, and do not support writing end-of-files in raw mode. Furthermore, unmodified tape drivers do not support tape controllers (like DIGI-DATA) which give two logical names to the same physical drive.

The optional, Harvard stty commands are not installed to save core and discourage incompatibilities. Stty commands would be more appropriate for changing densities, parity, error handling, on-line status, or the default number of file marks written.

SUBROUTINES
SUBROUTINES


CXAP

INTRODUCTION TO CXAP

CXAP is a group of C-callable subprograms. It is modelled after the XAP USER'S MANUAL. These subprograms permit basic I/O operations on picture files.

Each picture created by CXAP is comprised of a header section and a picture section. The header section is a 6 word record (12 bytes) at the beginning of the picture file.

The picture section begins immediately after the header section. Each row of a picture, starting with the first row, is stored as one logical record in the picture section. Each picture row has the same length. All CXAP pictures are stored sequentially, where row k follows row k-1.

A CXAP picture file can be any one of five byte sizes. A picture's byte size represents the number of bits needed to store the largest picture value in a picture file. Byte sizes are powers of 2, where the number of bits per pixel can range from 1 to 16.

It is assumed that a user of CXAP has knowledge of the C programming language. In the subprograms contained in CXAP, all indexing begins at zero. All information stored in arrays is stored in row-major order, i.e., the column increments the fastest. All CXAP subprograms are functions and return one of three values, -1 on error, 0 on end-of-file and +1 on success.

The following are the subprograms in CXAP:

BREAD(VII)	read the next row of a bordered picture
COPYDN(VII)	copy a row down
COPYUP(VII)	copy a row up
HEADER(VII)	obtain the header of a picture
MWRITE(VII)	write multiple copies of a picture row
PACK(VII)	pack the contents of a picture row
PREAD(VII)	read the next n row(s) of a picture
PWRITE(VII)	write the next n row(s) of a picture
REX(VII)	read and extend a picture row
SETUPB(VII)	open a picture file with a border surrounding it
SETUPR(VII)	open a picture file for reading
SETUPW(VII)	open a picture file for writing

UNPACK(VII)	unpack the contents of a picture row
XCLOSE(VII)	close a picture file
ZWRITE(VII)	write a row of zeroes to output picture

AUTHOR

Philip A. Dondes

SEE ALSO

C Reference Manual
XAP USER'S MANUAL, CN-21

CXAP PICTURE FILE STRUCTURE

There exist two types of CXAP picture files, disk files and tape files. CXAP disk picture files are comprised of two sections, a header section and a data section. The header section is a 6 word integer vector that starts at the beginning of a picture file. The following, in C-language notation, is the definition of this vector:

VECTOR[0] - unused,
VECTOR[1] - the number of columns in the picture,
VECTOR[2] - unused,
VECTOR[3] - the number of rows in the picture,
VECTOR[4] - unused,
VECTOR[5] - the pixel size of the picture,

where VECTOR is the picture header. The description of a picture's header is very straight forward except perhaps for the fifth element. This word is referred to as a picture's pixel size or simply, as the size of a picture. (Do not mistake this to mean a picture's dimensions, i.e., the number of columns and the number of rows in a picture.) If we let SIZE represent a picture's pixel size, then 1 <= SIZE <= 5, where 2**(SIZE-1) bits comprise a pixel.

A picture's pixel size has great significance when one considers the range of values that a picture may assume. The following table describes a picture's range with respect to its pixel size.

Pixel size	Bits per pixel	Range
1	1	0 or 1
2	2	0 to 3
3	4	0 to 7
4	8	0 to 255
5	16	-32768 to 32767

The data section contains integer pictorial data. If we define NCOL as the number of columns in a picture and NROW as the number of rows in a picture, the picture section consists of NROW rows of physical data and NCOL columns of logical data. If we let PIXWD = 16/(2**(SIZE-1)), then the physical length of picture row (in bytes) is computed by the C-language expression (NCOL/PIXWD*2)+(NCOL%PIXWD>0?1:0).

A CXAP disk picture is a sequential picture stored in a packed format. Only 2**(SIZE-1) bits are stored for each pixel when a picture is written to a disk file. If BUF is a C-language integer vector dimensioned to NCOL where each element of BUF contains a pixel value, then the pixels are packed towards BUF[0] before being written to disk file. Conversely, the pixels are unpacked towards BUF[NCOL-1] just after being read from a disk file. A standard CXAP picture should be stored left to right and from top to bottom. The least significant bit of a pixel is always the low order bit, i.e., the rightmost.

CXAP tape picture files are significantly different in format from disk files. Unlike disk files, tapes files do not contain a header and therefore contain only picture data. The picture data is stored on tape as NROW records of physical data with NCOL frames of logical data. There is no packing done when a tape file is created so that if a picture's pixel size is less than five, only one byte of data is written for each pixel. Otherwise, two bytes are written. The significant bits of data will always be the low order bits of a tape frame. For pictures with a pixel size equal to five, the low order byte is written first, followed by the high order byte. Every picture will have one end-of-file mark written at the end of its last record.

NAME

bread - read the next row of a bordered picture.

SYNOPSIS

```
int bread (area,buffer,ptr,&bias)

int area[30]
int buffer[depth][rlenth]
int ptr[depth]
int bias

int depth
int iw[4]
int rlenth
```

DESCRIPTION

area	work area for CXAP
buffer	input buffer as supplied to *setupb*
ptr	vector of subscripts as supplied to *setupb*
bias	position of the first point in each row
depth	maximum number of rows kept in *buffer*
iw	user's input window
rlenth	maximum number of columns kept in *buffer*

Bread reads the next row of a bordered picture. If *iw*[3] is not finished, three alternatives are present. The next row of the picture may be read into *buffer* , a row in *buffer* may be copied upward to create the top border, or a row in *buffer* may be copied downward to create the bottom border using the internal procedures *rex*, *copyup* and *copydn*, respectively. *Rex* is also used to extend a row right and and if needed, left. The row of picture points to be processed for some row k is:

buffer[*ptr*[k]] [*bias*],...,*buffer*[*ptr*[k]] [*bias*+*iw*[-2]-1].

The set of picture rows to be processed for some picture is:

buffer[*ptr*[*depth*/ 2]],..., *buffer*[*ptr*[*depth*/ 2]+ *iw*[3]-1].

Unlike *setupr* , no priming of the input buffer need be done. After the initial call to *setupb* , the first row of the input picture can be found in row *depth*/ 2 of the input buffer.

If the user tries to read more than *iw*[3] rows, an end-of-file will be returned.

Function value is -1 on error, 0 on end-of-file and +1 on success.

FILES

/mnt/phil/cxap/func/header.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"File not to be read"	input file was set up for write-only
"Read error"	error on attempt to read from picture file

AUTHOR

Philip A. Dondes

SEE ALSO

copydn(VII)
copyup(VII)
pread(VII)
rex(VII)
setupb(VII)

BUGS

NAME

copydn - copy a row down.

SYNOPSIS

```
int copydn (area,buffer,ptr)

int area[30]
int buffer[depth][rlenth]
int ptr[depth]

int depth
int rlenth
```

DESCRIPTION

area	work area for CXAP
buffer	input buffer as supplied to *setupb*
ptr	vector of subscripts as supplied to *setupb*
depth	maximum number of rows kept in *buffer*
rlenth	maximum number of columns kept in *buffer*

Copydn copies a row in *buffer* downward to create the bottom border for a picture which was opened using *setupb*. *Buffer*[*ptr*[0] will be the recipient of the most recently reflected row when the need arises to create the bottom border of a picture.

Function value is always +1 indicating success.

FILES

/mnt/phil/cxap/func/copydn.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

AUTHOR

Philip A. Dondes

SEE ALSO

bread(VII)
setupb(VII)

BUGS

NAME

copyup - copy a row up.

SYNOPSIS

```
int copyup (area,buffer,ptr)

int area[30]
int buffer[depth][rlenth]
int ptr[depth]

int depth
int rlenth
```

DESCRIPTION

area	work area for CXAP
buffer	input buffer as supplied to *setupb*
ptr	vector of subscripts as supplied to *setupb*
depth	maximum number of rows kept in *buffer*
rlenth	maximum number of columns kept in *buffer*

Copyup copies a row in *buffer* upward to create the top border for a picture which was opened using *setupb*. *Buffer*[*ptr*[*depth*- 1] will be the recipient of the most recently reflected row when the need arises to create the top border of a picture.

Function value is always +1 indicating success.

FILES

/mnt/phil/cxap/func/copyup.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

AUTHOR

Philip A. Dondes

SEE ALSO

bread(VII)
setupb(VII)

BUGS

NAME

header - obtain the header of a picture.

SYNOPSIS

```
int header (file,hbuf)

char *file
int hbuf[6]
```

DESCRIPTION

file	CXAP picture file
hbuf	storage for header information

Header reads the first 12 bytes of *file* into *hbuf*. *File* is closed before a return is made.

The description of *hbuf* is:

```
hubf[0] = unused,
hbuf[1] = # columns in picture,
hubf[2] = unused,
hbuf[3] = # rows in picture,
hubf[4] = unused,
hbuf[5] = byte size of picture.
```

Function value is -1 on error and +1 on success.

FILES

/mnt/phil/cxap/func/header.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"Source file not opened"	user's picture file was not opened
"Source header not read"	header of picture file was not read
"Source file not closed"	user's picture file was not closed

AUTHOR

Philip A. Dondes

SEE ALSO

BUGS

NAME

ioinfo - input picture information

SYNOPSIS

```
int                                                   ioinfo
(min,msg,argc,argv,ifile,ofile,iw,ci,ri,shift,size)

int     *min
char    *msg
int     argc
char    **argv
char    *ifile
char    *ofile
int     iw[4]
int     *ci
int     *ri
int     *shift
int     *size
```

DESCRIPTION

min — minimum number of arguments for execution of program

msg — usage message (printed if insufficient number of command line parameters are present)

argc — argument count

argv — argument vector

ifile — input file for reading

ofile — output file for writing

iw — input window, where elements refer to first column, first row, # of columns and # of rows, respectively.

ci — column increment to allow sampling of input columns

ri — row increment to allow sampling of input rows

shift — bit-wise shift for pixels

size — byte size (as defined by CXAP) of picture

Ioinfo is a routine which obtains all input from the user's command line. It is used in conjunction will picture handling.

If *argc* is less than *min*, *msg* is printed to the error output file and execution stops. *msg* should be the usage line for the execution of a program. Otherwise, *iw*[*0*], *iw*[*1*], *iw*[*2*], *iw*[*3*], *ci*, *ri*, *shift* and *size* are initially set to 1,1,1024,1024,1,1,0 and 4, respectively. If *ifile* does not begin with "/dev", then the input picture file is assumed to be on disk. The header information is read from *ifile* and *iw*[*2*] *iw*[*3*] and *size* are set to the actual picture header. If *ifile* begins will "/dev", the input file is assumed to be a system device and is let alone. *ofile* is then swapped to contain the second argument of the command line. All of the remaining arguments on the command line are copied to their respective parameters, provided the arguments exist. If they do not, they retain their default values.

FILES

/mnt/phil/cxap/prog/ioinfo	source code
/mnt/phil/cxap/func/header.c	reads a picture’s header
/mnt/phil/util/cvswap.c	swaps a character vector

DIAGNOSTICS

"Usage: ..."	*program usage line*
"file not opened"	the input picture file (on disk) was not opened for a header

AUTHOR

Philip A. Dondes

SEE ALSO

CXAP(VII)

BUGS

NAME

mwrite - write multiple copies of a picture row.

SYNOPSIS

```
int mwrite (area,num,vector)

int area[30]
int num
int vector[num * rlenth]

int iw[4]
int rlenth
```

DESCRIPTION

area	work area for CXAP
num	number of rows to be written out to the picture associated with area
vector	output buffer to be copied num times
iw	user's input window
rlenth	maximum number of columns kept in user's input buffer

Mwrite writes multiple copies of a picture row. If num it less than 1, an end-of-file is immediately returned. Otherwise, pwrite is called num times with parameters area, 1 and vector.

Function value is -1 on error, 0 on end-of-file and +1 on success.

FILES

/mnt/phil/cxap/func/mwrite.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"File not to be written"	output file was set up for read-only
"Output file not written"	error when write was attempted on output file

AUTHOR

Philip A. Dondes

SEE ALSO

pwrite(VII)
setupw(VII)

BUGS

NAME

pack - pack the contents of a picture row.

SYNOPSIS

```
int pack (buffer,numpix,pixwd)

int buffer[depth][rlenth]
int numpix
int pixwd

int depth
int rlenth
```

DESCRIPTION

buffer	input buffer as supplied to *setupb* or *setupr*
numpix	number of pixels in *buffer*
pixwd	number of pixels to pack per word
depth	maximum number of rows kept in *buffer*
rlenth	maximum number of columns kept in *buffer*

Pack will pack a picture row. A picture's density is determined from the picture's byte size, where 2^(SIZE-1) is the number of bits per pixel where SIZE is the picture byte size. If *pixwd* is 0 or 1, no packing is done and control returns immediately. Otherwise, the contents of *buffer* are packed by shifting the pixels to the right side of *buffer* so that there are *pixwd* pixels in each word of *buffer*.

Function value is -1 on error and +1 on success.

FILES

/mnt/phil/cxap/func/pack.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"Illegal packing density"	pixels per word parameter is less than 0 or greater than 16

AUTHOR

Philip A. Dondes

SEE ALSO

pwrite(VII)
pread(VII)

BUGS

NAME

pread - read the next n row(s) of a picture.

SYNOPSIS

```
int pread (area,num)

int area[30]
int num

int depth
int iw[4]
int ptr[depth]
int rlenth
buffer[depth][rlenth]
```

DESCRIPTION

area	work area for CXAP
num	next *num* rows are read into *buffer*
depth	maximum number of rows kept in *buffer*
iw	user's input window
ptr	vector of subscripts as supplied to setupb or setupr
rlenth	maximum number of columns kept in *buffer*
buffer	input buffer as supplied to setupb or setupr

Pread reads the next *num* rows of the picture associated with *area* into *buffer* provided that *iw*[3] is not exceeded. Each row is written into *buffer*[*ptr*[0]] before the *ptr* vector is rotated. *Ptr* is rotated such that *ptr*[i] = *ptr*[i+1] for 0 <= i < *depth*- 1 and *ptr*[*depth*- 1] = *ptr*[0].

If *num* is less than 1 or if the more than *iw*[3] rows are read, an end-of-file will be returned.

Unlike *setupb*, *buffer* must be primed by the user. *Depth* rows of the user's window must be read before *buffer*[*ptr*[k]] (0 <= k < *depth*) will contain row k+1 of the input window.

Function value is -1 on error, 0 on end-of-file and +1 on success.

FILES

/mnt/phil/cxap/func/pread.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"File not to be read"	input file was set up for write-only
"Read error"	error on attempt to read from picture file

AUTHOR

Philip A. Dondes

SEE ALSO

setupb(VII)
setupr(VII)
unpack(VII)

BUGS

NAME

pwrite - write the next n row(s) of a picture.

SYNOPSIS

```
int pwrite (area,num,vector)

int area[30]
int num
int vector[num * rlenth]

int iw[4]
int rlenth
```

DESCRIPTION

area	work area for CXAP
num	number of rows to be written out to the picture associated with _area_
vector	storage containing picture values to be written out
iw	user's input window
rlenth	maximum number of columns kept in user's input buffer

Pwrite writes out the next _num_ rows of a picture. The pixel values to be written are:

vector[0],..., _vector_[_num_*_iw_[2]-1].

The pixels to be written are divided into _num_ groups of _iw_[2] pixels. Group k (0 < k <= _num_) represents output row k.

If _num_ is less than 1, an end-of-file is returned immediately. The user may close a picture file prematurely or exceed the _iw_[3] count without any mishap. The header in the picture will reflect the actual number of rows written, as opposed to the number originally specified in the _setupw_ call.

Function value is -1 on error, 0 on end-of-file and +1 on success.

FILES

/mnt/phil/cxap/func/pwrite.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"File not to be written"	output file was set up for read-only
"Output file not written"	error when write was attempted on output file

AUTHOR

Philip A. Dondes

SEE ALSO

pack(VII)
setupw(VII)

BUGS

NAME

rex - read and extend a picture row

SYNOPSIS

```
int rex (area,buffer,ptr)

int area[30]
int buffer[depth][rlenth]
int ptr[depth]

int depth
int rlenth
```

DESCRIPTION

area	work area for Cxap
buffer	input buffer as supplied to *setupb*
ptr	vector of subscripts as supplied to *setupb*
depth	maximum number of rows kept in *buffer*
rlenth	maximum number of columns kept in *buffer*

Rex reads the next row of the picture associated with *area* using *pread* and extends the row right and if needed, left.

Function always returns +1 indicating success.

FILES

/mnt/phil/cxap/func/setupb.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

AUTHOR

Philip A. Dondes

SEE ALSO

bread(VII)
pread(VII)
setupb(VII)

BUGS

NAME

setupb - open a picture file with a border surrounding it.

SYNOPSIS

```
int setupb (area,name,iw,shift,depth,width,ptr,buf,rlenth)

int area[30]
char *name
int iw[4]
int shift
int depth
int width
int ptr[depth]
int buf[depth][rlenth]
int rlenth
```

DESCRIPTION

area — work area for CXAP

name — input picture file name

iw — the window of *name* to read

shift — each pixel is shifted 2^ *shift* amount before being read

depth — number of rows to keep in core at any one time

width — number of columns to keep in core at any one time

ptr — a vector of subscript values. *Ptr*[0] is the subscript of the oldest (topmost) row in *buf*. *Ptr*[*depth*- 1] is the subscript of the newest (bottommost) row in *buf*.

buf — storage to hold input picture rows

rlenth — the dimensioned row length of *buf*. It must be at least *iw*[2] + *width*- 1.

Setupb opens a picture file with a border surrounding it. *Setupb* is very useful if the user wishes to have a neighborhood operation defined over the entire picture, including the border. The border is created by reflecting outward the rows and columns adjacent to the perimeter of the picture, depending on the values of *depth* and *width*. For the top and bottom borders, the columns are reflected outward before the rows are copied up or down.

The processed point for a 4x5 neighborhood is marked with an X in the following example:

```
PPPPP
PPXPP
PPPPP
PPPPP
```

Setupb primes *buf* by reading the first *depth*- 1 rows immediately and creating a border, as needed. After the first call to *bread*, the first row of the input picture can be found in *buf*[*depth*/ 2].

If *depth* is less than 2, an end-of-file is returned.

Function value is -1 on error, 0 on end-of-file and +1 on success.

FILES

/mnt/phil/cxap/func/setupb.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"Setupr error in Bread"	the function call to *setupr* was unsuccessful

The error messages that *setupr* writes also apply.

AUTHOR

Philip A. Dondes

SEE ALSO

copydn(VII)
copyup(VII)
rex(VII)
setupr(VII)

BUGS

NAME

setupr - open a picture file for reading

SYNOPSIS

```
int setupr (area,name,iw,shift,depth,ptr,buffer,rlenth)

int area[30]
char *name
int iw[4]
int shift
int depth
int ptr[depth]
int buffer[depth][rlenth]
int rlenth
```

DESCRIPTION

area	work area for CXAP
name	input picture file name
iw	the window of *name* to read
shift	each pixel is shifted 2^ *shift* amount before being read
depth	number of rows to keep in core at any one time
ptr	a vector of subscript values. *Ptr*[0] is the subscript of the oldest (topmost) row in *buffer*. *Ptr*[*depth*- 1] is the subscript of the newest (bottommost) row in *buffer*.
buffer	storage to hold input picture rows
rlenth	the dimensioned row length of *buffer*. It must be at least *iw*[2]

Setupr opens a picture file for reading. The window of the picture file which will be read on subsequent calls to *pread* is defined by the *iw* parameter, where

iw[0] is the first column of the window,
iw[1] is the first row of the window,
iw[2] is the number of columns to read and
iw[3] is the number of rows to read.

Setupr is used only for initialization of an input picture. All buffer priming must be performed by the user. If no priming is preferred, *setupb* and *bread* should be used.

Function value is -1 on error, 0 on end-of-file and +1 on success.

FILES

/mnt/phil/cxap/func/setupr.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"Input file not opened"	error result when *name* was opened
"Input header not read"	header of picture file was not read

"Window invalid" user's window (i.e., *iw* parameter) is invalid for picture file

"Seek error" seek to first pixel in user's window failed

AUTHOR

Philip A. Dondes

SEE ALSO

setupb(VII)

BUGS

NAME

setupw - open a picture file for writing

SYNOPSIS

int *setupw* (area, name, iw, shift, size)

```
int area[30]
char *name
int iw[4]
int shift
int size
```

DESCRIPTION

area	work area for CXAP
name	input picture file name
iw	the window of *name* to write
shift	each pixel is shifted 2^shift amount before being written
size	byte size of output picture, whare 2^(*size* -1) describes the number of bits for each pixel.

DESCRIPTION

Setupw opens a picture file for writing. The window of the picture file which will be written on subsequent calls to *pwrite* is defined by the *iw* parameter. *Iw*[2] is the number of columns to write and *iw*[3] is the number of rows to write. *Iw*[0] and *iw*[1] are ignored. The output picture file corresponding to *name* will have file mode 0644.

Function value is -1 on error and +1 on success.

FILES

/mnt/phil/cxap/func/setupw.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"Output file not opened"	output picture was not opened
"Illegal byte size"	byte size of output picture was invalid
"Output header not written"	header was not written to output picture

AUTHOR

Philip A. Dondes

SEE ALSO

pwrite(VII)

BUGS

NAME

unpack - unpack the contents of a picture row.

SYNOPSIS

```
int unpack (buffer,numpix,pixwd)

int buffer[depth][rlenth]
int numpix
int pixwd

int depth
int rlenth
```

DESCRIPTION

buffer input buffer as supplied to setupb or setupr
numpix number of pixels in *buffer*
pixwd number of pixels to pack per word

depth maximum number of rows kept in *buffer*
rlenth maximum number of columns kept in *buffer*

Unpack will unpack a picture row. If *pixwd* is 0 or 1, no unpacking is done and control returns immediately. Otherwise, the contents of *buffer* are unpacked by shifting the pixels to the left side of *buffer* so that there is only one pixel in each word of *buffer*.

Function value is -1 on error and +1 on success.

FILES

/mnt/phil/cxap/func/unpack.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"Illegal unpacking density"	pixels per word parameter is less than 0 or greater than 16

AUTHOR

Philip A. Dondes

SEE ALSO

pwrite(VII)
setupb(VII)
setupr(VII)

BUGS

NAME

xclose - close a picture file

SYNOPSIS

```
int xclose (area)

int area[30]
```

DESCRIPTION

area   work area for CXAP

Xclose closes a picture file. If the picture file associated with *area* was opened using *setupw*, the picture file is checked to insure its correctness. If too few or too many rows were written to the picture file, the header is changed to reflect this fact.

Function value is -1 on error, 0 on end-of-file and +1 on success.

FILES

/mnt/phil/cxap/func/xclose.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"Output file prematurely closed"	too few or too many rows written to output picture
"Error on header seek"	a seek to the beginning of the file was unsuccessful
"New header not written"	updated header was not written to picture file
"File not closed"	output picture file not closed

AUTHOR

Philip A. Dondes

SEE ALSO

BUGS

NAME

zwrite - write a row of zeroes to a picture.

SYNOPSIS

```
int zwrite (area,num)

int area[30]
int num

int iw[4]
int zbuf[1024]
```

DESCRIPTION

area — work area for CXAP

num — number of rows to be written out to the picture associated with *area*

iw — user's input window

zbuf — vector of zeroes to be written to output picture

Zwrite writes out *num* rows of zeroes to the output picture associated with *area*. *Zbuf* is initialized to zero and then a call is made to *pwrite* using *area*, *num* and *zbuf* as parameters.

Iw[2] must be not be greater than 1024 in length.

Function value is -1 on error, 0 on end-of-file and +1 on success.

FILES

/mnt/phil/cxap/func/zwrite.c	source code
/mnt/phil/cxap/area.define	definitions for CXAP
/mnt/phil/cxap.lib	object code

DIAGNOSTICS

"Too many columns in picture"	length of an output picture row exceeds 1024
"File not to be written"	output file was set up for read-only
"Output file not written"	error when write was attempted on output file

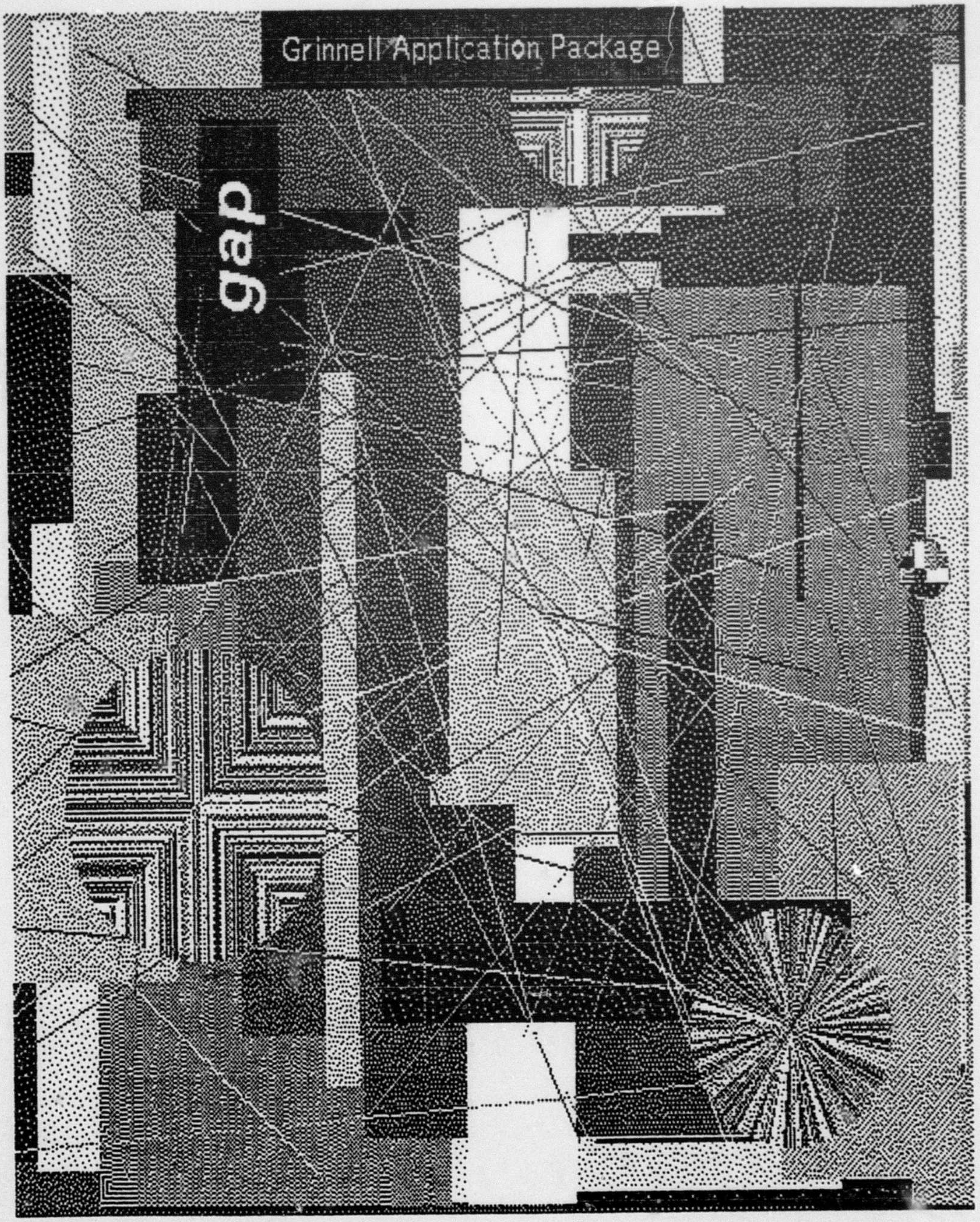
AUTHOR

Philip A. Dondes

SEE ALSO

pwrite(VII)
setupw(VII)

BUGS


Grinnell Application Package
gap

NAME

GAP --- GRINNELL APPLICATION PACKAGE

DESCRIPTION

GAP is a group of C compatible routines which simplify the interface between your C programs and the GRINNELL SYSTEMS display. Included in the package are functions to write (and read) rows, columns or points to (or from) a window on the display. With the exception of one, all the functions return a zero value when no errors occur, a negative value when an error condition is detected. The one exception is *grpnt*, which returns a 12-bit pixel value on no error. All the functions access a 16 integer buffer which is set up by the *gopen* function as a window descriptor. More than one window may be open at one time as long as each has its own descriptor (The descriptors should not be used by the user's routines (except in GAP calls) as they contain information which is vital to the GAP functions).

It should be noted that all points are referenced using the standard Cartesian coordinate system with this package. That is, columns are numbered consecutively from left to right and rows are numbered from bottom to top in the display (*not* top to bottom as with 1108 XAP). All numbering starts with *zero* (i.e. [column zero, row zero] is the lower lefthand corner of the display).

USAGE

```
cc [your-C-routines] -lg
```

SEE ALSO

cc(I), ld(I), gr(IV)
GRINNELL SYSTEMS User's Manual

NAME

header - Format of GAP image header

DESCRIPTION

The header considered standard by the GAP image handling routines consists of six words. Words 0, 2, and 4 always contain zeroes. Word 1 contains the number of columns in the image. Word 3 contains the number of rows in the image. Word 5 contains the minimum number of bytes needed for each pixel. An example of a header for a 512 column by 480 row 256 graylevel image is given below.

```
header[0] <--  zero
header[1] <--  512
header[2] <--  zero
header[3] <--  480
header[4] <--  zero
header[5] <--  1
```

This format is used due to the need of compatibility with many programming languages, including Fortran, C, and Lisp.

SEE ALSO

put (IV)

NAME

gopen --- Window creation function

USAGE

```
status = gopen(area, fcol, frow, ncol, nrow);
```

The parameters are:

area -- A 16 word array (or structure) used only by the GAP functions.

fcol,
frow -- Define the lower lefthand corner of the user's window. These are actual GRINNELL column and row numbers and must be from 0 to 511.

ncol,
nrow -- The number of columns and rows in the user's window (maximum size = 512).

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Gopen is used before all other GAP functions to set up the user's area buffer and open the GRINNELL device. The function parameters are used to define the physical window the user will be manipulating on the display. Though errors are detected relative to the user's window, no overlaps of separate windows can be discovered by the GAP functions. When many users want to use the display at one time, it is their responsibility to maintain the mutual integrity of their windows. It is therefore strongly recommended that the user input all parameters for the gopen function (with the exception of 'area') at the time of execution of the user's program.

EXAMPLE

```
int abuffer[16];
        .
        .

if(gopen(abuffer, 0, 0, 64, 64)) { /* NOTE -- Single user */
        printf("Gopen failed!\n");
        exit();
}
```

This will open the device and set up 'abuffer' to have values in it which define the user's window at the extreme lower left hand corner of the display. The columns (and rows) of the window will be numbered from 0 to 63.

NAME

gclose --- Close out a window

USAGE

```
status = gclose(area);
```

where 'area' is the user's 16 word buffer used in the gopen call.

VALUE RETURNED

zero -- no error

nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Gclose is used to close out a window when the user is finished. It will close the device and disable certain values in the 'area' buffer which have significant meaning to GAP functions.

EXAMPLE

```
int abuffer[16];
        .
        .
        .
i = gopen(abuffer, w, x, y, z);
        .
        .
        .
if(gclose(abuffer)) {
        printf("Gclose didn't work?\n");
        exit();
}
```

This shows the general way to close a window. Note that if an error is detected by gclose something very seriously wrong has occurred in your program which needs fixing...

NAME

genter --- Enter values into a window descriptor

USAGE

```
status = genter(area, channel, subchan, bakgnd, zwrite, dsize);
```

The parameters are:

area -- The buffer used in the gopen call

channel -- The channels to enable (if zero, no change)

subchan -- The subchannels to enable (if zero, no change)

bakgnd -- Select background (1 = light, >1 = dark)

zwrite -- Select zero write (1 = no write, >1 write)

dsize -- Select double size pixels

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Genter is used to change the default values for the channel, subchannel and write mode set up by the gopen call. On opening a window, gopen will select the image channels, enable all subchannels, select a dark background, enable zero writes and select single size pixels (1 point per pixel). To change these values in the area buffer, the user should use genter. Any parameters which are zero are left as is in the buffer.

Care must be taken using double size I/O. If this mode is set, the logical window size is halved.

EXAMPLE

```
int abuffer[16];
        .
        .

i = gopen(abuffer, 23, 46, 200, 200);
        .
        .
        .
if(genter(abuffer, 4, 0, 0, 0, 0)) {
        printf("Genter blew up\n");
        exit();
}
```

This user has decided to manipulate only the overlay channel. The subchannels selected are left as they are in the buffer abuffer as is the write mode.

NAME

gclear --- Selective clear of the user's window

USAGE

```
status = gclear(area, fcol, frow, ncol, nrow, subchan);
```

The parameters are:

```
        area -- The buffer used in the gopen call

        fcol,
        frow -- Define the lower lefthand corner of the subwindow
                to be cleared (relative to the user's window).
        ncol,
        nrow -- The number of columns and  rows in the subwindow.

        subchan -- The subchannels to clear
```

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Gclear is used to clear out the selected subchannels of the subwindow of the user's window. The first-column and first-row parameters sent in the call are *relative* to the lower lefthand corner of the window set up by the *gopen* call. The subchannels to be cleared are selected with the *subchan* parameter. If *subchan* is zero, the subchannels as defined in the *area* buffer are cleared. The subwindow to be cleared can be as large as the whole window or as small as one point (no larger, no smaller). Note that the window is not cleared when the *gopen* function is invoked, so it is a good idea to use the *gclear* call immediately after opening, unless you want to manipulate data that is already in your window.

EXAMPLE

```
int abuffer[16];
        .
        .
        .

i = gopen(abuffer, 23, 46, 100, 100); /* A 100 by 100 window */

if(gclear(abuffer, 0, 0, 100, 100, 0)) {
        printf("Can't clear out the window!!\n");
        exit();
}
```

This will open a 100 by 100 window at physical location (23,46). The subchannels have been defaulted to be all. The *gclear* call will then clear out the entire window. Note the relative column and row starting position for the subwindow.

NAME

gwrow -- Write out a row to a window

USAGE

```
status = gwrow(area, rbuf, rnum, rstart, npts, inc);
```

The parameters are:

area -- The buffer used in the gopen call

rbuf -- The user's integer array containing the values to be written out

rnum -- The row to be written (relative to the bottom of the window)

rstart -- The column to start writing to (relative to the left of the window)

npts -- The number of points to write out

inc -- The distance (and direction) between points

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Gwrow is used to write out a row to the user's window on the display. The row's values are transferred from the user's buffer to the row specified in the call. The column to start the writing to and the number of points to write out are also specified in the call. The inc parameter is used to write out the values a set distance apart. If inc is negative the pixels are placed from right to left in the row. It is the user's responsibility to make sure that the number of points to be written out will not violate the window's boundary. If too many points are requested an error condition will be returned and the operation will not be done.

EXAMPLE

```
int abuffer[16], thisrow[64], somewhere;
        .
        .
i = gopen(abuffer, 23, 46, 64, 64); /* A 64 by 64 window */
        .
        .
        .
if(gwrow(abuffer, thisrow, somewhere, 0, 64, 1)) {
        printf("ERROR ON ROW WRITE\n");
        exit();
}
```

This will write out 64 values in thisrow to the row somewhere. Furthermore, the output will start in the first column and the pixels will be next to each other.

NAME

gwcol -- Write out a column to a window

USAGE

status = gwcol(area, cbuf, cnum, cstart, npts, inc);

The parameters are:

area -- The buffer used in the gopen call

cbuf -- The user's integer array containing the values to be written out

cnum -- The column to be written (relative to the left of the window)

cstart -- The row to start writing to (relative to the bottom of the window)

npts -- The number of points to write out

inc -- The distance (and direction) between points

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Gwcol is used to write out a column to the user's window on the display. The column's values are transferred from the user's buffer to the column specified in the call. The row to start the writing to and the number of points to write out are also specified in the call. The inc parameter is used to write out the values a set distance apart. If inc is negative the pixels are placed from top to bottom in the column. It is the user's responsibility to make sure that the number of points to be written out will not violate the window's boundary. If too many points are requested an error condition will be returned and the operation will not be done.

EXAMPLE

```
int abuffer[16], thiscol[64], somewhere;
        .
        .
i = gopen(abuffer, 23, 46, 64, 64); /* A 64 by 64 window */
        .
        .
if(gwcol(abuffer, thiscol, somewhere, 0, 64, 1)) {
        printf("ERROR ON COLUMN WRITE\n");
        exit();
}
```

This will write out 64 values in thiscol to the column somewhere. Furthermore, the output will start on the first row and the pixels will be next to each other.

NAME

gwpnt -- Write out a point to the display

USAGE

```
status = gwpnt(area, value, cnum, rnum);
```

The parameters are:

area -- The buffer in the *gopen* call

value -- The integer value to be written out

cnum -- The window-relative column position of the pixel

rnum -- The window-relative row position of the pixel

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Gwpnt is used to write out one point to the column and row position specified in the call. The value sent should be between 0 and 4095 (the function will notice only 12 bits of the integer sent). With this function, points may be written in any order or direction one at a time. It can be used for border outlining, special symbol creation, etc.

EXAMPLE

```
int abuffer[16];
int pntval, x, y;
        .
        .
        .
i = gopen(abuffer, 23, 46, 64, 64);
        .
        .
        .
if(gwpnt(abuffer, pntval, x, y)) {
        printf("You goofed. Ha ha!\n");
        exit();
}
```

This will write out *pntval* to column *x* and row *y* of the window defined by *abuffer*.

NAME

gwvec -- Write a vector in a window

USAGE

```
status = gwvec(area, fcol, frow, lcol, lrow, type);
```

The parameters are:

area -- The buffer used in the gopen call

fcol -- The window-relative first column of the vector

frow -- The window-relative first row of the vector

lcol -- The window-relative last column of the vector

lrow -- The window-relative last row of the vector

type -- If nonzero, a solid rectangle is written

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Gwvec is used to write either a vector or a rectangle between two points in the user's window, depending on the type parameter. The vector will be written into the subchannels that are selected, those that are not selected will be zero written unless the genter command has been used to change this mode.

EXAMPLE

```
int abuffer[16], x0, y0, x1, y1;
        .
        .
i = gopen(abuffer, 23, 46, 64, 64); /* A 64 by 64 window */
        .
        .
if(gwvec(abuffer, x0, y0, x1, y1, 0)) {
        printf("Could not write the vector!\n");
        exit();
}
```

This will write out a vector between the point (x0, y0) and the point (x1, y1) in the selected subchannels. Nonselected subchannels will be cleared.

NAME

gwcur -- Write a cursor to a window

USAGE

```
status = gwcur(area, curnum, col, row, onoff, color);
```

The parameters are:

area -- The buffer used in the *gopen* call

curnum -- The cursor to write (1 or 2)

col,
row -- The window-relative position for the cursor

onoff -- If nonzero, the cursor will be visible

color -- If nonzero, the cursor will be white, else it will be black (if *onoff* is nonzero)

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Gwcur is used to manipulate the cursors in the user's window. The selected cursor will be written to window-relative position (col,row). In addition, the cursor can be turned on or off when it is positioned and the color can be selected to be black or white. When using this function, care should be taken that no other user is manipulating the same cursor.

EXAMPLE

```
int abuffer[16], mycur, x, y;
        .
        .
        .
i = gopen(abuffer, 23, 46, 64, 64);
        .
        .
        .
if(gwcur(abuffer, mycur, x, y, 1, 1)) {
        printf("Can't manipulate cursor!\n");
        exit();
}
```

This will write cursor *mycur* to window-relative position (x, y). The cursor will be visible and the color will be white.

NAME

gwstr -- Write an alphanumeric string to a window

USAGE

```
status = gwstr(area, string, fcol, frow)
```

The parameters are:

```
        area -- The buffer used in the gopen call

        string -- The string (or a pointer to a string)

        fcol,
        frow -- The position in the window for the lower lefthand
                corner of the first character of the string
```

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Gwstr is used to write a string of characters out to the user's window starting at relative location (fcol,frow) and proceeding from left to right. The standard ASCII 64 character set is used. Each character can be thought of as a 7X9 box containing some bit pattern. As such, the last column which can be the start of a character is 7 less than the number of columns in the user's window. The last row is 9 less. Characters are written into the selected subchannels of the enabled channels.

EXAMPLE

```
int abuffer[16];
        .
        .
        .
i = gopen(abuffer, 0, 0, 512, 512);
        .
        .
        .
if(gwstr(abuffer, "HI MOM!!", 250, 250)) {
        printf("It didn't say hello...\n");
        exit();
}
```

This will write out the string "HI MOM!!" approximately in the center of the user's window, starting at location (250,250)

NAME

gwtab -- Write to the lookup table

USAGE

```
status = gwtab(area, utab, start, nval);
```

The parameters are:

area -- The buffer used in the gopen call

utab -- The user's integer buffer containing the values to be written out

start -- Where in the lookup table to start

nval -- The number of values to write out

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Gwtab is used to change the values in the hardware lookup table. The values to be placed in the lookup table should be between 0 and 4095. As many as 4096 values or as few as one value may be placed in the table starting at location start and continuing for all nval values.

EXAMPLE

```
int abuffer[16], mytab[4096], i;
       .
       .
       .
i = gopen(abuffer, 0, 0, 512, 512);

for(i=0;i<4096;i++) mytab[i] = 4095 - i;

i = gwtab(abuffer, mytab, 0, 4096);
```

This will fill the lookup table locations 0 -> 4095 with the values 4095 -> 0 (i.e., the output is the inverse of the input).

NAME

grrow --- Read in a row from a window

USAGE

```
status = grrow(area, rbuf, rnum, rstart, npts, inc);
```

The parameters are:

area -- The buffer used in the *gopen* call

rbuf -- The user's integer array to receive the values to be read in

rnum -- The row to be read in (relative to the bottom of the window)

rstart -- The column to start reading from (relative to the left of the window)

npts -- The number of points to read in

inc -- The distance (and direction) between points

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Grrow is used to read in a row from the user's window. The values from the row specified are transferred from the display and sent directly to the user's row buffer. The column to start reading from and the number of points to read in are also specified in the call. The *inc* parameter is used to read in pixels which are a set distance apart. If *inc* is negative the pixels are read back from right to left from the display. It is the user's responsibility to make sure that the number of points to be read in will not violate the window's boundary. If too many points are requested an error condition will be returned and the operation will not be done.

EXAMPLE

```
int abuffer[16], thisrow[64], somewhere;
        .
        .
i = gopen(abuffer, 23, 46, 64, 64); /* A 64 by 64 window */
        .
        .
        .
if(grrow(abuffer, thisrow, somewhere, 0, 64, 1)) {
        printf("ERROR ON ROW READ\n");
        exit();
}
```

This will read in 64 values to *thisrow* from the row *somewhere*. Furthermore, the input will start from the first column and the pixels will be next to each other.

NAME

grcol --- Read in a column from a window

USAGE

```
status = grcol(area, cbuf, cnum, cstart, npts, inc);
```

The parameters are:

area -- The buffer used in the gopen call

cbuf -- The user's integer array to receive the values to be read in

cnum -- The column to be read in (relative to the left of the window)

cstart -- The row to start reading from (relative to the bottom of the window)

npts -- The number of points to read in

inc -- The distance (and direction) between points

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Grcol is used to read in a column from the user's window. The values from the column specified are transferred from the display and sent directly to the user's column buffer. The row to start reading from and the number of points to read in are also specified in the call. The inc parameter is used to read in pixels which are a set distance apart. If inc is negative the pixels are read back from top to bottom from the display. It is the user's responsibility to make sure that the number of points to be read in will not violate the window's boundary. If too many points are requested an error condition will be returned and the operation will not be done.

EXAMPLE

```
int abuffer[16], thiscol[64], somewhere;
        .
        .
i = gopen(abuffer, 23, 46, 64, 64); /* A 64 by 64 window */
        .
        .
        .
if(grrow(abuffer, thiscol, somewhere, 0, 64, 1)) {
        printf("ERROR ON COLUMN READ\n");
        exit();
}
```

This will read in 64 values to thiscol from the column somewhere. Furthermore, the input will start from the first row and the pixels will be next to each other.

NAME

grpnt --- Read in a point from a window

USAGE

```
pntval = grpnt(area, cnum, rnum); /* Returns the point */
```

The parameters are:

area -- The buffer used in the *gopen* call

cnum -- The window-relative column position of the point

rnum -- The window-relative row position of the point

VALUE RETURNED

nonnegative -- no error, point value returned
negative -- some error condition was detected (nothing done)

DESCRIPTION

Grpnt is used to read in one point from the column and row of the user's window as specified in the call. The value of the function, if no error has occurred, will be from 0 to 4095. With this function, points may be read back in any order or direction one at a time.

EXAMPLE

```
int abuffer[16];
int pntval, x, y;
        .
        .
        .
i = gopen(abuffer, 23, 46, 64 64);
        .
        .
        .
pntval = grpnt(abuffer, x, y);
if(pntval < 0) {
        printf("Neg value from grpnt\n");
        exit();
}
```

This will read in the value at column *x* and row *y*.

NAME

grcur -- Read in the positions of the cursors

USAGE

```
status = grcur(area, curbuf, asynch);
```

The parameters are:

area -- The buffer used in the gopen call

curbuf -- A four integer buffer to receive the cursor positions

asynch -- If nonzero, an asynchronous cursor read is done

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Grcur is used to read back the absolute (x,y) coordinates of the two cursors into the user's buffer in the order (x1,y1,x2,y2). If the *asynch* parameter is nonzero, *grcur* will not return until: (1) The track ball is moved and the TRACK switch is on, or, (2) The ENTER button is pushed. This mode is quite useful when a user tries to track a changing cursor position. It also lowers the load on the operating system. If *asynch* is zero, the cursor positions will be immediately returned. Note that the positions returned are *absolute*, not relative to the user's window.

EXAMPLE

```
int abuffer[16], cursors[4];
        .
        .
        .
i = gopen(abuffer, 23, 46, 64, 64);
        .
        .
        .
if(grcur(abuffer, cursors, 1)) {
        printf("Somebody blew it!\n");
        exit();
}
```

This will read in the *absolute* GRINNELL coordinates of the two cursors into the user's buffer *cursors*. The function will wait until the track ball has been moved or the ENTER button has been pushed.

NAME

grtab -- Read in the lookup table

USAGE

```
status = grtab(area, utab, start, nval);
```

The parameters are:

area -- The buffer used in the *gopen* call

utab -- The user's integer buffer to receive the values to be read out

start -- Where in the lookup table to start

nval -- The number of values to read in

VALUE RETURNED

zero -- no error
nonzero -- some error condition was detected (operation not done)

DESCRIPTION

Grtab is used to read in the values from the hardware lookup table. The values to be placed in the user's buffer will be between 0 and 4095. As many as 4096 values or as few as one value may be read from the table starting at location *start* and continuing for all *nval* values.

EXAMPLE

```
int abuffer[16], mytab[4096], i;
        .
        .
        .
i = gopen(abuffer, 0, 0, 512, 512);
        .
        .
        .
if(grtab(abuffer, mytab, 0, 4096)) {
        printf("Couldn't read the lookup table\n");
        exit();
}
```

This will copy the entire lookup table into the user's buffer *mytab*.

NAME

gcam -- Input an image from the T.V. camera

USAGE

```
status = gcam(area, nfrms, shift);
```

The parameters are:

area -- The buffer used in the gopen call

nfrms -- The number of frames to sum (nfrms < 256)

shift -- The amount to shift each frame before summing (shift should be from 0 to 7)

VALUE RETURNED

zero -- no error

nonzero -- Some error condition was detected (operation not done)

DESCRIPTION

Gcam is used to input images into the GRINNELL display memories. Using the nfrms and shift parameters a user can average as many as 64 consecutive frames (up to 255 can be summed). Single frames with no shift are input by setting nfrms to zero.

EXAMPLE

```
int abuffer[16];
          .
          .
          .
i = gopen(abuffer, 0, 0, 512, 512);      /* NOTE -- Single User */
          .
          .
          .
if(gcam(abuffer, 64, 6)) {
        printf("Couldn't input from camera\n");
        exit();
}
```

This will input and average sixty four frames from the camera. That is, each of 64 frames will be input, downshifted 6 bits (divided by 64) and added to the previous sum.

NAME

gscrl -- Scroll the image

USAGE

```
status = gscrl(area, num, updown);
```

The parameters are:

area -- The buffer used in the gopen call

num -- The number of rows to scroll

updown -- Scroll up or scroll down (zero, nonzero)

DESCRIPTION

Gscrl is used to scroll the display up or down. As many as 512 rows may be scrolled at one time. Scrolling the image could be used when a continuous strip display is wanted.

EXAMPLE

```
int abuffer[16], row[512];
        .
        .
        .
i = gopen(abuffer, 0, 0, 512, 512);
        .
        .
        .
if(gscrl(abuffer, 1, 0)) {
        printf("NO SCROLL\n");
        exit();
}
i = gwrow(abuffer, row, 0, 0, 512, 1);
```

This user scrolls the image up one row, then overwrites the first row, making a continuously upwards moving strip.

NAME

gcolor -- Turn color mode on and off

USAGE

status = gcolor(area, flag);

The parameters are:

area -- The buffer used in the *gopen* call

flag -- Used to change the mode

VALUE RETURNED

zero -- no error
nonzero -- Some error condition was detected (operation not done)

DESCRIPTION

Gcolor is used to change the display from color to black and white and back again. If *flag* is zero, the display will be color. If it is nonzero the display will be black and white.

EXAMPLE

```
int abuffer[16];
        .
        .
        .
i = gopen(abuffer,0,0,512,512);
        .
        .
        .
if(gcolor(abuffer,1)) {
        printf("No color change allowed!\n");
        exit();
}
```

This will change the displayed image to black and white.

OTHER

(Miscellaneous)

subroutines

NAME

binop - applies binary operation between two pictures.

SYNOPSIS

```
#include "binop.t"
```

DESCRIPTION

Binop contains the C source for a driver program which applies a point-wise binary operation between two picture files with GAP style headers. To construct a program:

```
#include  "defns.t"
#include  "binop.t"
char  binop( p, q )  char  p, q;
{ ...function for  computing  pointwise  binary  opera-
tion...}
```

The first file is read from standard input, the second from the file argument given on the call, EXCEPT that if the file argument is prefixed immediately with a hyphen ("-"), the order of the files is reversed when applying the binary operator. The result picture is written to standard output.

FILES

binop.t - C source for driver
defns.t - useful definitions

DIAGNOSTICS

"File argument needed!"
"Can't open file argument!"
"First file not a byte per pixel file!"
"Second file not a byte per pixel file!"
"Picture files don't match!" - not the same size
"Premature end on first file!"
"Premature end on second file!"
"Excess data remaining on first file!"
"Excess data remaining on second file!"

In the above, "first file" normally means standard input, and "second file" the file given as argument, but these roles are reversed if the "-" prefix is used on the file argument.

AUTHOR

Les Kitchen

SEE ALSO

euc(VI), max(VI), dirn(VI), unop(VII), local(VII)

BUGS

Only works for one byte per pixel files.
Read error will also cause "Premature end..." message.
Should have a facility for getting parameters, like local(VII).

NAME

errprnt - print error message and exit

SYNOPSIS

```
char *argv0 0; /* copy 0th parameter - program's name */
main(argc, argv)
char **argv;
{
        argv0 = argv[0];
        . . .

errprnt(comstr, arg);
char *comstr;
```

DESCRIPTION

Errprnt is a subroutine that printf's the calling program's name and a user supplied command string that may use an optional second argument. *Errprnt* sends to the diagnostic file 2 that is normally the user's tty.

The calling program's name is supplied through the commom variable *argv0* that should be set equal to the calling programs argument 0 by the main routine. The user supplied command string *comstr* may use one single word optional second parameter *arg* such as a string pointer or an integer. After printing *comstr*, *errprnt* prints a carriage return and exits back to the operating system using return code 1.

Versions are available for both the standard printf or the portable C library version.

DIAGNOSTICS

If argv0 is not defined in the main program, ld(I) complains.

FILES

SEE ALSO

readrow(VII), printf(III), exec(II), ld(I)

AUTHOR

Robert L. Kirby

BUGS

NAME

openfont, closefont, readchar, freechar, accesschar, height, width, r_width, baseline, name, code, leftadj - font input routines

USEAGE

```
cc <program> -lF
```

SYNOPSIS

```
int fd;
char *name;
fd = openfont(name);

char *name
closefont(name);

int fd, cd;
char c;
cd = readchar(fd, c)

int cd;
freechar(cd);

int pixel, cd, x, y;
pixel = accesschar(cd, x, y);

int cd, high;
high = height(cd)

int cd, wide;
wide = width(cd)

int cd, rwid;
rwid = r_width(cd)

int cd, base;
base = baseline(cd)

int fd;
char *nam;
nam = name(fd)

int cd, cod;
cod = code(cd)

int cd, left;
left = leftadj(cd)
```

DESCRIPTION

These routines read and access characters from font files. *Openfont* and *readchar* call the library routine *alloc* to reserve space for their respective descriptors, and return a pointer to the descriptor. *Closefont* and *freechar* clean up and then free space used by the descriptor. This is *NOT* the same as doing: free(descriptor) directly. The actual manipulation of the descriptors should be handled *only* by these subroutines.

Openfont will first examine the current directory for the named font, then the system font library. It will then append the ".fnt" suffix if necessary. The descriptor it returns is used by *closefont*, and *baseline*. A -1 return indicates that the named font file does not exist or is inaccessible.

Closefont closes the actual file descriptor and frees the font descriptor.

Readchar is given a font descriptor and an ASCII character. It will allocate core space for the character's raster pattern and read it in from the font file indicated by the descriptor. The descriptor it returns is used by *accesschar* and *freechar*.

Accesschar returns a 0 or 1 for any single pixel of the character's raster pattern. A -1 indicates an attempt to access a pixel beyond the limits of the raster.

Freechar releases the core space occupied by the character. This is *NOT* equivalent to calling the library routine *free*.

Name must be passed a font descriptor. It returns a pointer to the name of the font.

Height and *baseline* may be passed a font descriptor *or* a character descriptor. They return the height/baseline of the font/character.

Width, *r_width*, *code* and *leftadj* accept only a character descriptor. *Width* returns the width of the character, *r_width* returns the width of the character's raster (which may be greater than the defined width of the character -- meaning that the character overlaps to the right), *code* returns the ASCII code for the character, and *leftadj* returns the number of pixels by which the character should overlap the character to its left.

AUTHOR

Fred Blonder

FILES

/lib/libF.a - library where these routines live
/b/fonts - system font library

SEE ALSO

FONT(V)

DIAGNOSTICS

Several, from all the routines. They should be self explanatory, and are written to the error output. When they occur the routine detecting it will always return -1.

BUGS

Error checking is not rigorous. Obvious errors such as passing the wrong type of descriptor are caught, but not every possible combination of invalid parameters is tested for. Since the descriptors returned by the routines are pointers to structures, using them as pointers yourself could result in the structures being trashed, causing these routines to bomb the next time they are called.

NAME

font - font file format

DESCRIPTION

The format of the font files used by *pqp* is as follows. The first 512 bytes are 128 two word entries for the characters. (numbered 0 - 127) in each entry. The first word is the total character width, including any white space. The second word is the address of the character's definition farther on in the file.

Starting at byte 512 there are:

The font height in pixels. (1 word)

The distance of the font's baseline from the top of the font. (1 word)

A null terminated ASCII string containing descriptive information about the font.

The character definitions begin at byte 768. Each one consists of the following one word fields, and the raster pattern.

Character code: Essentially a pointer back into the index table at the beginning of the file.

Raster width: The number of pixels in one row of the character's raster pattern.

Raster length: The number of pixels in one column of the character's raster pattern.

Left overlap of the previous character.

Rows from top: The number of pixels at the top of the character that are left blank, i.e. the vertical offset of the raster pattern.

The size of the raster pattern in bytes.

The raster pattern: The pixels of the raster are stored starting with the upper left hand pixel and proceeding to the right across each row. Each group of eight pixels is stored in one byte, from the high-order to the low-order bits. If a row of the raster does not fit into a group of bytes evenly, the next row begins in the same byte. Only the last byte of the raster is padded with unused bits.

The files look like this:

```
 | Character #1  | Character #2  |     . . .  | Character #128|

 ----v-------v-------v-------v---------   -----v-------v----
 | Char  | Char  | Char  | Char  |      . . . | Char  | Char  |
 | Width |  Ptr  | Width |  Ptr  |            | Width |  Ptr  |
 ----^-------^-------^-------^------     -------^-------^----
  0    1    2    3    4    5    6    7           508 509 510 511


                 | Description ->

 ----v-------v-------v-------v-------v-------
 | Font  | Font e| F | O | O | B | A | R |\0 |
 | Height|Baselin|   |   |   |   |   |   |   |
 ----^-------^-------^-------^-------^-------
  512 etc.  ->


                                               | Raster ->

 ----v-------v-------v-------v-------v-------v-------v-------
 | Char  | Raster| Raster| Left  |Rows F-| Raster|   |   |. . .
 |  Code | Width | Height|Overlap|rom Top|  Size |   |   |
 ----^-------^-------^-------^-------^-------^-------^-----
  768 etc.  ->
```

AUTHOR

File format designed by Lee Moore. Documented by Fred Blonder.

FILES

/lib/fonts/*.fnt - system font library

SEE ALSO

pgp (I), descfnt (I)

NAME

local - applies local operation to a picture.

SYNOPSIS

```
#include "local.t"
```

DESCRIPTION

Local contains the C source for a driver program which applies a local operation to a picture file with GAP style header. To construct a program:

(o) #include "defns.t" (important definitions).

(i) #define the following constants:

O_HEIGHT the vertical height of the local operator in pixels;

O_WIDTH the horizontal width of the local operator in pixels;

O_BAKGND the background filler value for border points where the operator can't fit (typically zero);

O_X_CNTR & O_Y_CNTR the x & y co-ordinates of the "center" of the operator (the output of the operator is stored at the point in the output picture which corresponds to the center).

N.B. All co-ordinates are strictly Cartesian: The x co-ordinate increases from zero, left to right across the picture; and the y co-ordinate from zero, bottom to top. The picture is stored on file by rows, with the first row (y=0) at the bottom of the picture.

(ii) Declare global variables for storing operator parameters.

(iii) Include a procedure:

```
getpars( argc, argv ) int argc; char *argv[];
```

for setting of the above global variables from the command line (using standard conventions for argc & argv).

(iv) Include a function:

```
char localop( nbd )  char *nbd[ O_HEIGHT ];
```

which returns the result of the local operator. Using zero-origin Cartesian co-ordinates, relative to the neighborhood, the point (x,y) can be accessed as nbd[y][x] inside localop.

(v) #include "local.t"

The program reads the input picture from standard input and writes the result to standard output. Both input and result pictures have the same size.

FILES

local.t - C source for driver
defns.t - useful definitions

DIAGNOSTICS

"Input not a byte per pixel picture file!" - header not right

"Input picture too small for operator!" - picture smaller than neighborhood

"Premature end of input!" - picture smaller than expected from header

"Excess input data remaining!" - picture larger than expected from header

AUTHOR

Les Kitchen

SEE ALSO

sobel(VI), unop(VII), binop(VII)

BUGS

Only works for one byte per pixel files.
Read error will also cause "Premature end..." message.

NAME

readrow - read fixed length row in spite of pipe shortchanging

SYNOPSIS

```
readrow(buffer, length)
char *buffer;
int length;
```

DESCRIPTION

Readrow is a subroutine that fills *buffer* with *length* bytes from the standard input regardless of the actual input file type. Thus when reading images or matrices, *readrow* can get an entire row regardless of any pipe shortchanging. *Readrow* returns the number of bytes it was unable to read because an end of file was found. In many cases, if the number returned at the end of the file is not the number of bytes requested then the data length is incorrect.

Readrow uses *errprnt* to notify the user of read errors. *Errprnt* expects the calling program's name to be supplied through the commom variable *argv0* that should be set equal to the calling programs argument 0 by the main routine.

Readrow provides economical raster input for programs that only use one input file. Using operating system support, a program can be written as a filter that avoids using irrelevant workspaces or setup routines. For many image processing applications, all input may be handled by *readrow* alone.

DIAGNOSTICS

On physical data errors, *readrow* prints ''read error'' and exits. If *argv0* is not defined in the main program, ld(I) complains.

FILES

SEE ALSO

errprnt(VII), printf(III), exec(II), ld(I)

AUTHOR

Robert L. Kirby

BUGS

Only the standard input may be read.

NAME

unop - applies unary operation to a picture.

SYNOPSIS

```
#include "unop.t"
```

DESCRIPTION

<u>Unop</u> contains the C source for a driver program which applies a point-wise unary operation to a picture file with GAP style headers. To construct a program:

```
#include  "defns.t"
#include  "unop.t"
char  unop( p )  char  p;
{ ...function  for  computing  pointwise  unary  opera-
tion...}
```

The input picture is read from standard input, The result picture is written to standard output. Can be used for making filters

FILES

unop.t - C source for driver
defns.t - useful definitions

DIAGNOSTICS

"Input not a byte per pixel file!" - header not right
"Premature end of input!" - picture smaller than expected from header
"Excess input data remaining!" - picture larger than expected from header

AUTHOR

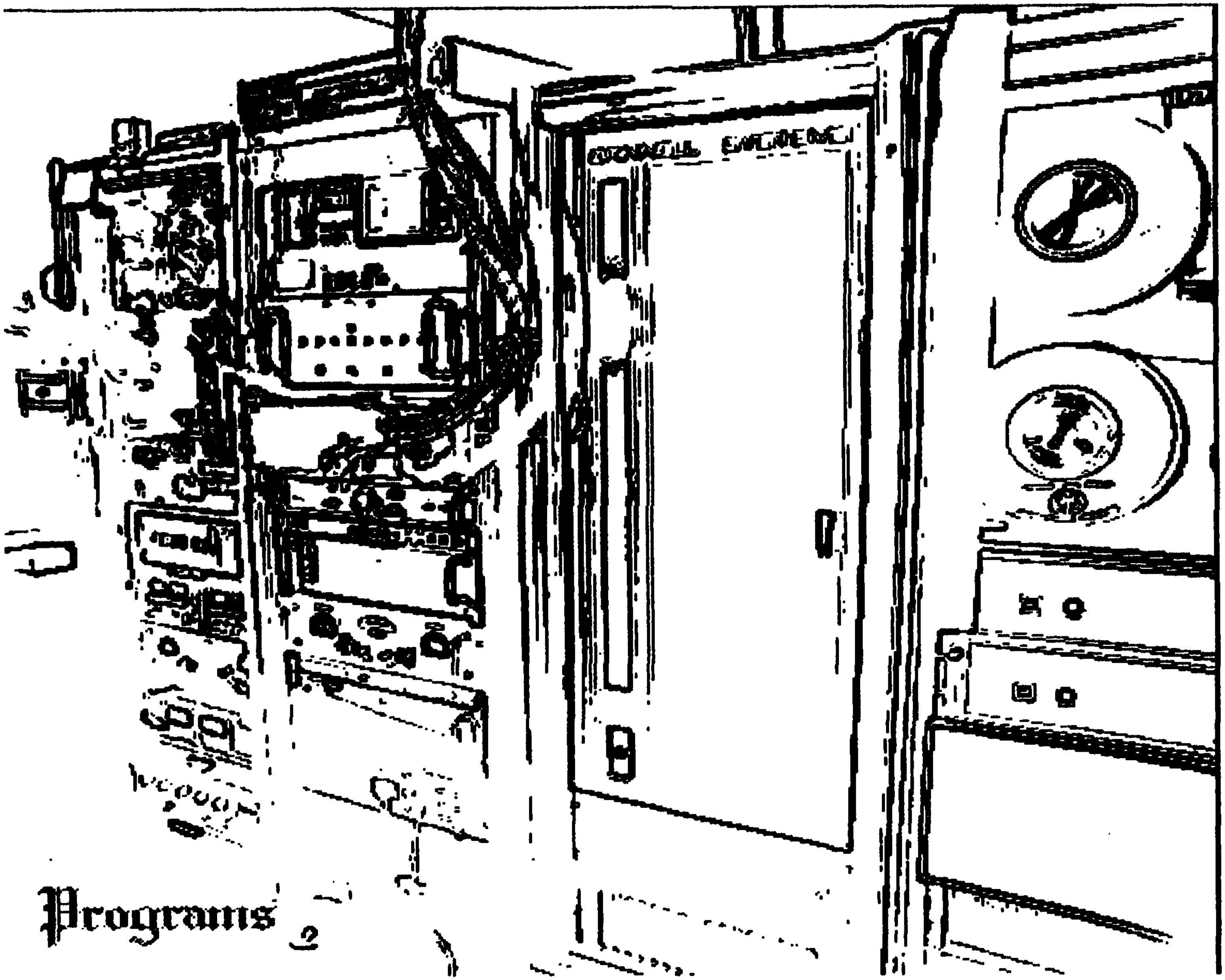
Les Kitchen

SEE ALSO

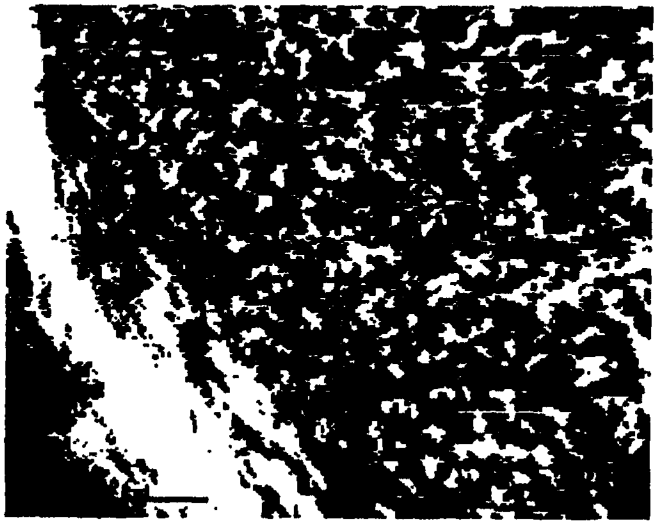
abs(VI), binop(VII), local(VII)

BUGS

Only works for one byte per pixel files.
Read error will also cause "Premature end..." message.
Should have a facility for getting parameters, like local(VII).



Programs



Programs:

Gordon J VanderBrug

Dir

Picture Creation and Modification

NAME

abs - takes absolute value of a picture file.

SYNOPSIS

abs

DESCRIPTION

Abs reads a picture file with GAP style header from standard input, takes the absolute value of every pixel, and writes the resulting picture file to standard output. May be used as a filter.

FILES

abs.c - C source code

unop.t - driver program for unary picture operators in general

defns.t - useful definitions

DIAGNOSTICS

"Input picture not a byte per pixel file!"
- header byte count not one

"Premature end of input!"
- picture smaller than expected from size given in header

"Excess input data remaining!"
- picture bigger than expected from size in header

AUTHOR

Les Kitchen

SEE ALSO

unop(VII)

BUGS

Works only for 1 byte per pixel files.

Read error will also cause "Premature end of input!" message.

NAME

biglet - print big letters using fonts

SYNOPSIS

biglet <fontname>

DESCRIPTION

Biglet reads text from the standard input, using it to select characters from <fontname> which are written to the standard output sideways, like this:

```
  **
 ****   *
******  **
**  **  **
**   *   **
**   **  **
**   **  **
 **  ** ***
**********
*********
**



              *
             **
***************
***************
 **     **
**       **
**       **
**       **
***     ***
 *********
  *******
```

Linefeeds in the input text are ignored, they will not produce spaces. Control characters may be entered by typing a ``%'' before the appropriate letter, i.e.: ``%a'' for control-a.

Biglet precedes each line of output with a control-``F'', which will put the Printronix into 8-lpi mode so that the raster pattern is more square than it would be otherwise.

AUTHOR

Fred Blonder

FILES

<fontname>.fnt - the specified font file

SEE ALSO

DESCFNT(I), TITLE(I), PGP(I), FONT(III), FONT(V)

DIAGNOSTICS

. . . are given for an unknown font, or a character that is not defined in the specified font.

BUGS

NAME

calib - University of Maryland Computer Vision Laboratory scanner calibration routine

SYNOPSIS

calib

DESCRIPTION

Calib is used to calibrate the scanner. It will construct a picture and send it out over the interface to the scan memory. The scanner should be online with the PDP11/45. The *columns* switches should be set to 375 (octal). The *rows* switches should be set to 376 (octal). The image sent is used to adjust the focus and gray scale of the Polaroids to be made. It should be compared with the standard and the controls adjusted accordingly.

FILES

/dev/rdr0

SEE ALSO

dr (IV)

NAME

color - Turn on color display mode
gray - Turn off color display mode

DESCRIPTION

Color and *gray* are used to switch the display back and forth between color and black and white mode. They do not in any way affect the stored data and may be used at any time (even if someone else is using the display). It should be noted that the Grinnell has a possible range of 256 graylevels, so 4 bits/pixel are not displayed when color mode is off.

USAGE

```
color
gray
```

EXAMPLE

```
% color
      {Turn on color mode}
% gray
      {Turn off color mode}
```

AUTHOR

Russ Smith

NAME

cm - find central moments of a shape

SYNOPSIS

_cm_ pic [I J]

DESCRIPTION

pic	64 x 64 binary GAP picture
I, J	specifies particular central moment

_cm_ finds the I, J central moments of the shape contained in the picture. If I and J are not specified, the 11, 02, 20, 21, 12 central moments and the area of the shape are found.

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

qcm(V)

BUGS

NAME

csize - changes a picture's pixel size

SYNOPSIS

csize [newsize]

DESCRIPTION

newsize newsize is the picture's new pixel size

Csize reads a picture's header from the standard input file and changes the fifth word of the header (i.e., the picture's pixel size) to newsize. The new header is then written to the standard output file. If newsize is not given, the header is unchanged. Csize then reads an integer picture of dimension NROW rows by NCOL columns, where NROW is the number of rows in the input picture and NCOL is the number of columns in the input picture, from the standard input and writes it to the standard output file.

FILES

DIAGNOSTICS

"Header not read"	first 12 bytes of picture not read
"Picture width too large"	*# of columns in input* picture exceeds input buffer length
"Illegal size"	newsize < 1 or newsize > 5
"Too much data, row count expired"	# of rows in picture exceeded
"Unexpected EOF encountered"	Eof encountered before picture completed
"Input pipe not empty"	data still resides in input pipe

AUTHOR

Philip A. Dondes

SEE ALSO

PACK(VI)
UNPACK(VI)

BUGS

NAME

descfnt - describe the contents of a font file

SYNOPSIS

```
descfnt <fontname> [ <sample-character> ]
```

DESCRIPTION

Descfnt has two modes: If it is invoked with one argument it reads the font file named by the argument, lists all the ASCII characters which are defined in the font, and prints the descriptive information stored in the font file. If it is invoked with two arguments it takes the first one as a font name -- as before -- and displays the raster of the font character corresponding to the second argument, in this form:

```
****************
****************
****************
***
***
***
***
***
***************
***************
***************
***
***
***
***
***
***
***
***
```

AUTHOR

Fred Blonder

FILES

<fontname>.fnt

SEE ALSO

BIGLET(I), TITLE(I), PGP(I), FONT(III), FONT(V)

DIAGNOSTICS

. . . are given for an unknown font, or a character that is not defined in the specified font.

NAME

diffop - produce a difference picture

SYNOPSIS

<u>diffop</u> ifile odirect nsize dir [fc] [fr] [nc] [nr]

DESCRIPTION

```
ifile     input picture file as defined by CXAP
odirect   directory where output picture will be created
nsize     neighborhood size where neighborhood  is  a  square
          region 2^nsize for nsize = 1, 2 and 3.
dir       direction for difference operation where  direction
          =  1,  2,  3  and 4 refers to horizontal, vertical,
          right diagonal and left diagonal, respectively.
fc        first column of input picture
fr        first row of input picture
nc        number of columns to process in input picture
nr        number of rows to process in input picture
```

<u>Diffop</u> creates a difference picture for a given direction. The size of the difference neighborhood may be 2x2, 4x4 or 8x8. The difference operation is defined to be the sum of the A's minus the sum of the B's, where the A's and the B's are depicted below:

```
            Right       Left
Horizontal      Vertical      Diagonal        Diagonal

AAAABBBB          AAAA        AAAA                 AAAA
AAA*BBBB          AAAA        AAAA                 AAAA
AAAABBBB          AAAA        AAAA                 AAAA
AAAABBBB          A*AA        AAA*                *AAAA
                  BBBB            BBBB        BBBB
                  BBBB            BBBB        BBBB
                  BBBB            BBBB        BBBB
                  BBBB            BBBB        BBBB
```

where the * represents the point presently being considered. The same representation holds for a neighborhood of 2x2 and 8x8.

A single CXAP output picture is created for each execution of <u>diffop</u>. The picture name will be "h", "v", "rd" or "ld" (depending upon the direction) residing under the user specified directory "odirect."

FILES

Directory p7140 exist on magnetic tape P7140 in STP format.

/p7140/diff/diffop.c	source code for difference program (includes all subprograms
/mnt/phil/cxap.lib	CXAP library

DIAGNOSTICS

"Header error" error in attempt to read input picture header

"Setupb error" error in attempt to open input picture

"Bread error" error in attempt to read input picture

"Pwrite error" error in attempt to write output picture

"Setupw error" error in attempt to open output picture

"Creating file s" file s is being created for output

AUTHOR

Philip A. Dondes

SEE ALSO

CXAP(VII)

BUGS

Diffop uses SETUPB(VII) and BREAD(VII) to avoid getting any edge reaction on the border of the picture.

The default value(s) for omitted input window dimensions is the actual dimension corresponding to the input picture.

NAME

dirn - edge gradient direction.

SYNOPSIS

dirn file2

DESCRIPTION

Dirn reads one picture from standard input, and a second picture from the file given as argument. It computes

```
atan2( y, x )
```

scaled to be in the range -128 to +127 (instead of -pi to +pi), at each point of the output picture, where x and y are the values at the corresponding points in the two input pictures. If x and y are signed edge detector outputs in x and y directions respectively, then this gives the edge gradient direction measured in 256ths of a revolution. Note that this will use all 8 bits of a byte. The result picture is written to standard output. All files have GAP style headers.

FILES

dirn.c - C source code
binop.t - driver program for binary operators in general
defns.t - useful definitions

DIAGNOSTICS

See binop(VII), from which driver they all originate.

AUTHOR

Les Kitchen

SEE ALSO

binop(VII), sobel(VI), euc(VI)

BUGS

See binop(VII).

NAME

doodle - 2-d and 3-d graphics on the Grinnell

SYNOPSIS

doodle

DESCRIPTION

Doodle is a self-contained system for creating and manipulating 2- and 3- dimensional objects and functions on the Grinnell. Objects are created and manipulated by a set of commands which allows the user to specify what part of the screen is to be used, the position of the observer, the direction of view, etc. The following commands are currently available.

Display control commands

status - prints the observer's current position, direction of view, the size and shape of the display window, and the size, shape and position of the viewport.

viewport llx lly urx ury - establishes the size and position of the viewport on the Grinnell screen. The maximum size is 4.0 by 3.75 which is the entire Grinnell screen. (llx,lly) is the lower left corner and (urx,ury) is the upper right corner, which are initially set to (0, 0) and (4.0, 3.75).

position x y z - moves the observer to the location (x,y,z). This is initially set to (6.0, 8.0, 7.5).

direction i j k - sets the vector components of the direction of view. If the observer is at point (x,y,z) and direction is set to (-x,-y,-z) then the observer is looking at the origin. The initial values are (-6.0, -8.0, -7.5).

size x y - sets the size of the display window. The 3-d space is projected onto this window, which is then projected onto the viewport. The initial window size is 5.0 by 5.0.

distance z - sets the distance between the window and the observer. The observer looks through the center of the window. Distance is initially 10.

erase - erases the entire screen, regardless of the viewport being used.

erasevp - erases the current viewport, but not its frame, if it has one.

frame - draws a frame around the current viewport.

unframe - erases the frame around the current viewport, if there is one.

Two-d commands

mov x y - moves to the point (x,y) in viewport coordinates.

drw x y - draws a line from where you are to the point (x,y) in viewport coordinates.

move x y - moves to the point (x,y) in window coordinates.

draw x y - draws a line from where you are to the point (x,y) in window coordinates.

Three-d commands

3d name1 name2 - performs the required transformations on the input file (name1) and outputs the transformed coordinates to the output file (name2). If the output file does not already exist, then it is created.

skel name - takes the output file created by *3d* and outputs a wire frame, or skeleton plot of the file.

solid name - takes the output file created by *3d* and outputs a solid plot of the file, with all hidden lines removed.

plot3 - plots three dimensional mathematical functions, and is not worth discussing in its present state.

Miscellaneous commands

prompt string - changes the prompt string.

read name - reads input commands from a file rather than from the terminal. When the entire file has been read, control returns to the terminal.

end - terminates the program. The screen is left as is.

3d input file format

Three dimensional objects are constructed out of planar polygons. A cube for instance would be six polygons, each of which is a square. A polygon is specified by listing its vertices. An edge connects each vertex in the list to the next vertex, with the last vertex connected to the first, forming a closed polygon. The polygons may have any number of edges greater than or equal to three. The separate polygons do not have to be related in any way, and they may even penetrate each other.

The input file format is as follows.

```
<file> => <polygon> * <polygon> ... * <polygon> **
<polygon> => <point> ... <point>
<point> => x y z  (three numbers, separated by at least 1
space)
```

The following is an example of the file format. The list below gives the points for two squares with endpoints (0,0,0), (1,0,0), (1,1,0), and (0,1,0), and (0,0,1), (1,0,1), (1,1,1), and (0,1,1).

```
0 0 0
1 0 0
1 1 0
0 1 0
*
0 0 1
1 0 1
1 1 1
0 1 1
**
```

The numbers in the file can be positive or negative and they may include a decimal point.

AUTHORS

Gyorgy Fekete, with extensions by James W. Williams.

FILES

/dev/gr

SEE ALSO

Principles of Interactive Computer Graphics, *1st ed*. by Newman and Sproull.

DIAGNOSTICS

Doodle tells you if a command doesn't make sense, or is not implemented yet. Error messages are also printed if *doodle* can not open or close the files it is using, or if one of the Grinnell functions can not be peformed.

BUGS

Doodle may occasionally crash for unknown reasons. The 3-d function plotting feature is not easy to use. 3-d objects displayed using the *solid* command may not penetrate the plane that passes through the observer's position and is perpendicular to the direction of view. The *skel* command handles this situation correctly.

NAME

erase - Erase the display

DESCRIPTION

Erase is used to erase the entire screen. Either the image channels, the overlay or both image and overlay can be erased.

USAGE

```
erase          {erase the entire screen (overlay and image)}
erase a        {erase the overlay}
erase a a      {erase the image}
```

EXAMPLE

```
% grid
      {Put up an alignment grid in the overlay}
% erase a
      {Erase the grid}
```

AUTHOR

Russ Smith

NAME

ersw - erase window contents

DESCRIPTION

When more than one user is manipulating data on the GRINNELL display it is bad form to use the erase command. Instead, ersw is used to erase just that portion of the screen desired. Ersw can also be used to erase selected bit groupings of the window (i.e., one can erase just the red bits, leaving the green and blue untouched).

USAGE

```
 ersw [key]
        key -- u6,u8 -- upper six or eight bits erased
               l6,l8 -- lower six or eight bits erased
               r,g,b -- red, green or blue bits erased
               a ------ all 12 bits erased (default key)
               o ------ overlay within window erased
```

EXAMPLE

```
% posw 100 100 64 64
% ersw
```

The user wants to erase just a 64 by 64 window on the GRINNELL display, leaving the rest of the display intact.

NAME

euc - euclidean combination of two pictures (L2 norm).

SYNOPSIS

euc file2

DESCRIPTION

Euc reads one picture from standard input, and a second picture from the file given as argument. It computes

$$\sqrt{x^2 + y^2}$$

at each point of the output picture, where x and y are the values at the corresponding points in the two input pictures. If the result of this expression is greater than 63, it is set to 63. The output picture is written to standard output. All files have GAP style headers.

Useful for combining the output of edge operators in x and y directions.

FILES

euc.c - C source code
binop.t - driver program for binary operators in general
defns.t - useful definitions

DIAGNOSTICS

See binop(VII), from which driver they all originate.

AUTHOR

Les Kitchen

SEE ALSO

binop(VII), sobel(VI), dirn(VI)

BUGS

See binop(VII).
Chopping at 63 may be a nuisance.

NAME

expand - expand a given picture and display it on the Grinnell

SYNOPSIS

expand pfile size factor

DESCRIPTION

pfile an n x n picture with GAP header

size an integer specifying the size of the picture to be expanded

factor an integer specifying the number of times to expand the picture

expand displays the picture on the Grinnell. The window is relative to (0,0) .

FILES

/dev/gr

DIAGNOSTICS

all deal with file access errors and are self-explanatory

AUTHOR

Sanjay Ranade

SEE ALSO

shrink(VI)

BUGS

The picture size can be obtained from the header. Extra parameters can be added to specify a general window on the Grinnell.

NAME

freeze - Freeze input from the video digitizer

DESCRIPTION

Freeze is used to freeze the current input from the video digitizer. It is usually used immediately following the *tv* command, though it can be used by itself. Used alone, *freeze* can average up to 64 frames of input or sum up to 255.

USAGE

```
freeze [#frames shift]
       #frames = 1 -> 255
       shift = 0 -> 7
```

EXAMPLE

```
% tv
       {Start input}
% freeze
       {freeze current image}

OR

% erase
       {clear screen}
% freeze 64 6
       {average ~2 seconds of video input}
```

AUTHOR

Russ Smith

NAME

getw - redisplay currently defined window

DESCRIPTION

Getw is used to redisplay a window previously defined but subsequently erased. Often after using posw to define a window a user will erase the overlay, hence making the window's location difficult (if not impossible) to discern. Getw will output the first column, first row, number of columns, and number of rows information to the user's terminal and redraw the window on the display. If an argument is given the cursors will be turned on but the overlay will not be erased.

USAGE

```
getw [o]
```

EXAMPLE

```
% posw 100 100 64 64
% hstw u6
% getw o
```

This user, after computing a histogram of a window and writing it into the overlay (thus erasing the window outline), wishes to know the true position of the window but does not want to erase the histogram.

GMAP (Author: Donald J. Gerson)

This program does false color mapping (alias color slicing) of a black and white image stored in the Grinnell by modifying the lookup table. The standard way to get an image into the Grinnell -- using the TV camera -- is to enter these commands:

```
gray
tv
freeze
```

To run gmap do:

```
color
gmap
```

A typical execution of gmap goes something like this (user input is underlined):

```
do you want to store picture (y or n)?
n
number of bits per pixel?
8
give x-base
0
give y-base
0
give size
150
```

What you enter controls the placement of the color triangle. It makes no difference whether you save the picture or not; the program doesn't ever restore it. The bits per pixel value may be any positive number. (Enter zero at your own risk!) The x-base and y-base are the location where you want the color triangle placed, and ``size'' is the size to make the triangle.

After gmap has drawn the triangle, use cursor #1 to mark pixels in the color triangle. You can either leave the track switch on to make a continuous line, or with the track switch off, use the ``enter'' button to mark single pixels. The colors you mark - and the order you mark them in - determine the mapping from grey levels to colors. For example: if you mark a red, a white, and a blue pixel, (in that order) the greyscale will be divided into three equal segments. The darkest segment will be mapped into red, the middle one into white, and the brightest into blue.

To indicate that you are done, press the ``enter'' button an extra time while the cursor remains on the last pixel you marked. Gmap will then load the mapping you have defined into the Grinnell's lookup table so you can see the result of the mapping. It then asks if you are done, to which you may answer ``no'' if you want to try a different mapping. In this case gmap starts over so you must enter the color triangle parameters again. When you quit, the current mapping remains in the lookup table so it can be used with other pictures. Doing:

```
tv
```

will run the image from the tv camera through the mapping so you can watch it in real time.

To restore the lookup table to the identity mapping do:

```
gtbld s
```

NAME

grid - Alignment grid program

DESCRIPTION

Grid is a small routine used to produce a grid of lines spaced 32 pixels apart in the overlay of the GRINNELL display. Though it was designed to aid in the correction of the aspect ratio of a television monitor being aligned, it can possibly be put to other use.

USAGE

```
grid
```

EXAMPLE

```
% grid
            {Put up the grid}
%
```

AUTHOR

Russ Smith

NAME

gt - GRINNELL memory plane test
pgt - printing version

DESCRIPTION

Gt is used to visually check out individual image bit planes. Using the lookup table and solid rectangle generation, it will reveal bit dropouts as white spots on an otherwise dark screen. *Pgt* will, in addition, plot out the results on the PRINTRONIX line printer (for hardcopy confirmation).

USAGE

```
gt [bit #]
pgt [bit #]
```

EXAMPLE

```
% pgt 7
        (test bit #7 and plot the results)
%
```

AUTHOR

Russ Smith

NAME

gtbld - GRINNELL lookup table load

DESCRIPTION

Gtbld can be used to load the GRINNELL hardware lookup table with preset values. Currently the table may be loaded with the values 0->4095 (standard), 4095->0 (complement), or 0+4095 (threshold). Quite often this program is used to remap images created from the T.V. camera for manipulation by programs running on the UNIVAC 1108 (The GRINNELL says black is zero, the 1108 (XAP), white).

USAGE

```
gtbld [s][r][t tval]
  s -- 0 -> 4095
  r -- 4095 -> 0
  t tval -- threshold at 12-bit integer value tval
```

EXAMPLE

```
% gtbld t 3000
      {all pixels valued 3000 up will display as white,
       all pixels valued 2999 down as black}
```

AUTHOR

Russ Smith

NAME

gtest - GRINNELL Internal Test program

DESCRIPTION

Gtest consecutively performs the four internal hardware tests of the GRINNELL DISPLAY SYSTEM as the keyboard <RETURN> key is pressed. It is usually used on powerup of the system to load the lookup table, reset the device registers and initialize the device memory.

USAGE

```
gtest
```

EXAMPLE

```
%  gtest <RETURN>

        Hit <RETURN> For Each Test

  <RETURN>   <-- Internal test #1 displayed
  <RETURN>   <-- Internal test #2 displayed
  <RETURN>   <-- Internal test #3 displayed
  <RETURN>   <-- Internal test #4 displayed
%
```

AUTHOR

Russ Smith

NAME

hstw - compute and display histogram of window

DESCRIPTION

Hstw will compute a histogram of the window contents and display a bar graph in the overlay. The previous contents of the overlay will be erased. The histogram can be of the color bits (red, green, blue) or of the upper or lower six or eight gray level bits.

USAGE

```
 hstw key
        key -- u6,u8 -- upper six or eight bits histogrammed
               l6,l8 -- lower six or eight bits histogrammed
               r,g,b -- red, green or blue bits histogrammed
```

EXAMPLE

```
% freeze
% posw 0 0 480 504
% hstw u6
```

The user has taken one tv frame from the video digitizer input and wishes to see a histogram of the entire image thus obtained. The digitizer produces six bit output hence the 'u6' argument.

NAME

ht - Print a semi-halftone of the display

DESCRIPTION

Ht will take the image as displayed (i.e., through the lookup table) and plot a semi-halftone of it for the PRINTRONIX line printer. The image is printed as a collection of black and white spots by an error correcting process. As such, some loss of small details should be expected. The output can be piped.

USAGE

```
ht > /dev/lp
ht ¦ opr
```

EXAMPLE

```
% scale
       {Interactive gray scale remapping}
% ht ¦ opr
       {Output piped to the printer spooler}
%
```

AUTHOR

Russ Smith

NAME

lscan - Display a cross section of an image

DESCRIPTION

Lscan will display a vertical or horizontal cross section through an image. A graylevel image can be thought of as possessing three dimensions: height, width, and brightness. This program is used to display the brightness dimension as a cross section through one of the other two dimensions. When this program is run a white line will appear on the screen. This line is moved using the TRACK ball and cursor #1 to that portion of the image through which the cross section is wanted. When the ENTER button on the track ball unit is pushed twice, the cross section will appear on the screen (in the overlay). This sequence can be repeated as often as wished. To exit hit <RUBOUT>.

USAGE

```
  lscan [length [a]]
         length -- width (or height) of cross section
         a -- a vertical line will be used
```

EXAMPLE

```
  % lscan 512
         {A horizontal full screen cross section}
    <RUBOUT>
  %
```

AUTHOR

Russ Smith

NAME

mapw - map displayed window into stored window

DESCRIPTION

Mapw is used to map the stored values of the window through the lookup table and back into the stored values. That is, it transforms the stored values as they are displayed so that they actually take on the displayed values. It is usually used after the lookup table has been changed so that the lookup table can immediately be reloaded with the standard sequence of values. Hence other concurrent users of the GRINNELL display will not have their own images displayed incorrectly for an extended period of time.

USAGE

```
mapw
```

EXAMPLE

```
% gtbld t 3000
% posw 100 100 64 64
% mapw
% gtbld s
```

This user has thresholded an image, mapped it, and reloaded the lookup table so as to not interfere with other potential users.

NAME

max - pointwise maxmum of two pictures.

SYNOPSIS

max file2

DESCRIPTION

Max reads one picture from standard input, and a second picture from the file given as argument. It computes

max(x, y)

at each point of the output picture, where x and y are the values at the corresponding points in the two input pictures. The output picture is written to standard output. All files have GAP style headers.

FILES

max.c - C source code
binop.t - driver program for binary operators in general
defns.t - useful definitions

DIAGNOSTICS

See binop(VII), from which driver they all originate.

AUTHOR

Les Kitchen

SEE ALSO

binop(VII)

BUGS

See binop(VII).

MS (Author: Lee Moore)

This program is invoked by:

```
ms <start> [ <increment> [ <color> ] ]
```

it generates a series of "munching squares" patterns on the Grinnell. The pattern uses the entire screen and its appearance is determined by the parameters on the call line. The increment and color values can be defaulted to one (1) and 4095, respectively. The color argument is an integer between zero (0) and 4095 which is the value to be placed on the screen.

The algorithm can be best explained by the following program fragment:

```
n := start;
i := increment;

FOR i := 0 TO lastx DO BEGIN
        FOR x := 0 to lastx DO BEGIN
                y := x XOR n;
                putpoint(x,y) END
        n := n + i END
```

where the routine "putpoint" puts a point at (x,y) on the display. Different values for start and increment produce different patterns. Larger increments tend to produce finer grain patterns, especially if the values are prime.

Once the above fragment completes its execution, the program will place black points instead of colored ones and then repeats. When this is complete, the whole process is started over again.

NAME

mul - scale a picture by a constant

SYNOPSIS

mul [constant]

DESCRIPTION

Mul reads a picture's header from the standard input file and echoes it to the standard output file. *Csize* then reads an integer picture of dimension NROW rows by NCOL columns, where NROW is the number of rows in the input picture and NCOL is the number of columns in the input picture, from the standard input, multiplies every pixel by *constant* and writes it to the standard output file. *constant* is 1 if unspecified.

FILES

DIAGNOSTICS

"Header not read"	first 12 bytes of picture not read
"Picture width too large"	# of columns in input picture exceeds input buffer length
"Illegal size"	*newsize* < 1 or *newsize* > 5
"Too much data, row count expired"	# of rows in picture exceeded
"Unexpected EOF encountered"	eof encountered before picture completed
"Input pipe not empty"	data still resides in input pipe

AUTHOR

Philip A. Dondes

SEE ALSO

pack(VI)
unpack(VI)

BUGS

NAME

munch - Munching squares demonstration program

DESCRIPTION

Munch is a shell program which starts up three munching squares routines in the three primary colors. It makes for an impressive demonstration of the speed at which individual points can be written. It's also pretty. The programs will be stopped when the RUBOUT button is pushed.

USAGE

```
munch
```

EXAMPLE

```
% munch
  r#
  g#
  b#
       {r#, g#, b# -- The process numbers for each color}
        <RUBOUT>
       {the user kills the processes}
```

AUTHOR

Russ Smith

``PAINT BY NUMBERS'' (Author: Fred Blonder)

The ``paint by numbers'' system consists of four seperate programs. These are: draw which allows you to input line drawings into the Grinnell system using the track ball and your terminal keyboard, binin which reads the line drawing from the Grinnell into a Unix file as a binary picture, connect which runs the (four neighbor) connected component labeling algorithm on the binary picture file, producing a labeled picture file, and paint which allows you to use the track ball to mix colors and use them to fill in any component in the labeled image, displaying the result on the Grinnell. A description of each of these programs follows.

DRAW

Draw takes no arguments, and uses the entire screen of the Grinnell. When it is run, cursor #1 should be turned on, cursor #2 off, and the track switch should be on. You use the track ball to move cursor #1 to where you want to start a line, and then you can draw the line either freehand - with the track ball - or have the Grinnell draw a straight vector.

To do it freehand, type ``d'' on the keyboard. This puts the program in the ``pen down'' condition, where pixels are turned on wherever you move cursor #1. Type ``u'' to lift the ``pen''.

To draw a straight vector type ``m''. This drops cursor #2 at the current location of cursor #1. Move cursor #1 to the other end of the desired line and type ``l''. This places a vector between the cursors.

To aid in drawing geometric figures, the coordinates of both cursors are displayed on the overlay channel in the upper left hand corner of the screen, along with the pen (up/down) position. Since a single pixel can connect two regions, you must take care to enclose all the regions in the picture.

BININ

This program copies a rectangular window from the Grinnell - treating it as a binary picture - into an arbitrary Unix file. The window is selected by positioning the Grinnell's cursors at any two of its diagonally opposite corners before running the program. Binin is called as follows:

```
binin -ci <filename>
```

The *c* option causes the program to examine the cursors to determine the window; the default is to read in the entire screen. The *i* option is necessary to cause the program to invert the picture as it is read in because the *connect* program likes it that way. <Filename> may be the name of any writeable Unix file, but for traditional reasons it should begin with ``/tmp/''. The file will be created if it does not exist.

CONNECT

This program doesn't use the Grinnell. It runs the connected component labeling algorithm on its input picture, producing a component labeled output picture. It is called like this:

```
connect <binary input picture> <labeled output picture>
```

where <binary input picture> will normally be the file you just created using *binin*. The same comment applies to the name you give <labeled output picture> as to the file name given to *binin*. *Connect* prints out some information as it runs to let you know what is happening.

PAINT

This is the fun and exciting part. To work properly cursor #1 should be turned on, and #2 off. *Paint* is called like this:

```
paint <labeled output picture>
```

It displays the picture as a line drawing in the lower left hand corner of the Grinnell's screen, and displays a palette in the upper right hand corner.

To mix colors the cursor must be positioned within the block of the palette containing the three primary color bars. You can either leave the track switch off, move the cursor over any color bar to the desired length of the bar and hit the *enter* button, or you may leave the track switch on, and position the cursor over any color bar and move it up or down while the color bar changes length along with it. The currently selected color is continuously displayed in a rectangle at the top of the palette.

To color in a region of the picture with the currently selected color, move the cursor (with the track switch off) to any point within the region, and press *enter*. No more commands may be given to the program until that region has been colored in. The same color may be used to color any number of regions without needing to reselect the color. A region may be recolored any number of times; the previous color in the region will be overwritten completely.

When you are done painting, exit by pressing the <u>rubout</u> key on your terminal. The palette will be erased, and the borders between the components will be erased and then filled in, by blending the colors of the adjacent regions.

NAME

pgp - print text on the Printronix using fonts

SYNOPSIS

pgp <fontname> <output-file>

DESCRIPTION

Pgp reads text from its standard input, and the font file associated with *fontname*. It produces an output file containing the text, set in the specified font, along with the neccessary control codes to put the Printronix into plot mode. This file may be sent directly to the printer, or edited and concatenated with other similar files first.

If a file called: *fontname* doesn't exist *pgp* appends ``.fnt'' to the name and tries to find it again.

Many of the fonts have printing characters defined for several of the ASCII control codes which may be bothersome to type from a terminal. These codes can be entered by typing ``%a'' for control-A, ``%b'' for control-B, &c. A ``%%'' is interpreted as a single ``%''.

AUTHOR

Original version: Lee Moore
Current version: Fred Blonder

FILES

<fontname>.fnt

SEE ALSO

DESCFNT(I), TITLE(I), BIGLET(I), FONT(III), FONT(V)

DIAGNOSTICS

. . . are given for an unknown font, or a character that is not defined in the specified font.

BUGS

If the output line length is greater than what will fit on the Printronix, the Printronix will not go into plot mode, and the file prints as garbage. This is a bug in *pgp* and not a problem with the Printronix.

NAME

phist - Computer Vision Lab Histogram Printing Routine

SYNOPSIS

phist [-] xappic1 xappic2 ... xappicn

DESCRIPTION

Phist will print the histogram for each PDP/XAP formatted picture file given as an argument. If the dash is present, only a summary is printed on the standard output. If the dash is not present, actual bar graphs will be plotted on the PRINTRONIX line printer.

FILES

/dev/lp

BUGS

NAME

posw - Position a window (use overlay)
oposw - Position a window (nonoverlay)

DESCRIPTION

Posw (oposw) is used to position the two cursors so as to define a rectangular window on the Grinnell display. Cursor #1 defines the lower left corner, cursor #2 defines the upper right corner. If posw is used, the window will be displayed in the overlay (and anything else in the overlay will be erased); if oposw is used, the two cursors will be turned on but the overlay will remain untouched. This routine is to be used before all other window oriented programs.

USAGE

```
 posw [[fc fr] nc nr]
 oposw [[fc fr] nc nr]
        fc,fr - first column and row of window
        nc,nr - number of columns and rows
```

If no arguments are given the window will be completely dynamic in both size and position. If the number of columns and rows is given, only the position of the window will be dynamic. If all arguments are given, the window will be immediately positioned (static position and size). The TRACK ball unit is used to position the window and change its size. When the window is correctly positioned, pushing the ENTER button on the TRACK ball unit twice will fix the window in place.

EXAMPLE

```
 %  posw
        {user wants to interactively change
        position and size}
 %  posw 64 64
        {user wants to interactively position a
        64 by 64 window}
 %  posw 100 100 64 64
        {user knows exactly where to position a
        64 by 64 window.
```

NAME

put - Put an image on the Grinnell display

SYNOPSIS

put [fc fr] < file
put8 [fc fr] < file

DESCRIPTION

Put is used to place an image on the Grinnell display. The image should have a correct header (GAP). Both *put* and *put8* can put 12-bit images (full color). *Put* is used for images having six bit gray level ranges. *Put8* is used for images having eight bit gray level ranges. With either version of *put* the column and row for the lower lefthand corner of the image may be specified. If the column and row are not given a cursor will be displayed. This marks the lower lefthand corner of the image. It may be positioned using the TRACK ball unit. When the cursor is correctly positioned, push the <RUBOUT> key to actually put the image.

FILES

/dev/gr

SEE ALSO

header (V)

NAME

savw - read window contents off display

DESCRIPTION

Savw reads the contents of the display and writes data to the standard output. Either image data or the overlay may be read off in any raster scan direction (left to right, right to left, bottom to top, etc.). Arguments to the program select that portion of the data to be saved as well as the direction of the scan. A 12-byte header containing number of columns, number of rows, and bytes per pixel information will precede the data unless the '-' is used with the mode argument.

USAGE

```
 savw key [[-]mode] > file
 savw key [[-]mode] > device
 savw key [[-]mode] | process
         key -- u6,u8 -- upper six or eight bits saved
                l6,l8 -- lower six or eight bits saved
                r,g,b -- red, green or blue bits saved
                a ------ all 12 bits saved
                o ------ overlay saved

         If the '-' is present, no header is output

         mode -- lb,lt,rb,rt,bl,br,tl,tr
                 l -- left to right
                 r -- right to left
                 b -- bottom to top
                 t -- top to bottom
                 i.e., 'lt' would be a standard tv scan
                 'lb' is the default scan
```

EXAMPLE

```
% posw 0 0 510 510
% savw u8 -lt > /dev/rdr0
```

This user is sending a gray level image from the GRINNELL display to the scanout device (in order to make a poloroid photo). Note that no header is sent and that a standard tv scan (left to right, top to bottom) is used.

NAME

scale - Interactive grayscale mapping

DESCRIPTION

Scale is used to interactively change the hardware lookup table graylevel mapping. The x-axis of the monitor screen is considered to represent the possible range of displayed (through the lookup table) pixel values. The y-axis is considered to represent the actual stored value range. Hence if one were to plot the mapping function of the standard lookup table values a 45 degree line starting at the origin and proceeding to point (511,511) would result. Using the TRACK ball and cursor #1 this function can be interactively changed (i.e., the line can be redrawn) allowing any graylevel mapping such as multiple thresholds, contrast stretching, etc.

USAGE

```
scale
```

EXAMPLE

```
% scale
        {Interactive function mapping}
        <RUBOUT>
%
```

AUTHOR

Russ Smith

NAME

shrink - progressively shrink a picture to produce a 'pyramid'

SYNOPSIS

shrink pfile ftemp

DESCRIPTION

pfile a 64 x 64 picture with the standard 12-byte header

ftemp a filename template. If this is 'ftemp', the files created would be 'ftemp1','ftemp2'......'ftemp6'.

shrink produces 6 files corresponding to each level of the pyramid. 'ftemp1' is 32 x 32 , 'ftemp2' is 16 x 16 .... etc. A pixel in a level n file is the average of four corresponding pixels in the level (n-1) file.

FILES

ftemp1,ftemp2.....ftemp6 picture files created

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

expand(VI)

BUGS

At the moment the program only shrinks 64 x 64 GAP pictures.

NAME

sobel - applies Sobel edge operator to a picture.

SYNOPSIS

sobel d

DESCRIPTION

Sobel reads a picture file from standard input, applies the Sobel operator, and writes the result to standard output. Both pictures have GAP style headers, and the output picture is the same size as the input picture. If the first character of the direction parameter d is an 'x', then the operator is applied in the x direction, if a 'y', then it is applied in the y direction. For a 3 by 3 neighborhood:

```
ABC
DEF
GHI
```

the x direction operator produces

```
(C + 2*F + I - A - 2*D - G) / 4;
```

the y direction operator produces

```
(A + 2*B + C - G - 2*H - I) / 4.
```

(Note that x increases left to right across the picture, y increases bottom to top. The picture is read row by row, bottom to top.) The outer border of the output picture will be all zeroes, since the Sobel operator cannot be applied right up against the edge.

FILES

sobel.c - C source code for Sobel operator
local.t - driver program for local operators in general
defns.t - useful definitions

DIAGNOSTICS

"Input picture not a byte per pixel file!"
- header byte count not one

"Input picture too small for operator!"
- smaller than 3x3

"Premature end of input!"
- picture smaller than expected from size given in header

"Excess input data remaining!"
- picture bigger than expected from size in header

"Direction argument missing!"

"Direction must be x or y!"

AUTHOR

Les Kitchen

SEE ALSO

local(VII), euc(VI), dirn(VI)

BUGS

Works only for 1 byte per pixel files.
Read error will also cause "Premature end of input!" message.

NAME

stretch - Graylevel range stretching

DESCRIPTION

Stretch changes the lookup table values in order to produce a 64 graylevel displayed image from a less than 64 level stored image. The user inputs to the program the lowest and highest stored values of the image. These will then become pure black and pure white in the stretched image. Note that only the displayed image will be affected. The stored image remains the same (see MAPW).

USAGE

```
stretch low high
       low -- low value of stored image
       high -- high value of stored image
```

EXAMPLE

```
% stretch 7 43
       (7 ==> 0, 43 ==> 63, values in between suitably changed)
%
```

AUTHOR

Russ Smith

NAME

tempera - allows one to "paint" on Grinnell using trackball as "brush"

SYNOPSIS

tempera

DESCRIPTION

TEMPERA is a program that simulates tempera painting. The screen of the Grinnell monitor is the "canvas;" the "paintbrush" is the trackball device. The artist may direct a "palette" to appear on the screen, when it is desired to electronically mix another color of "paint." At the beginning of the program, the artist may choose either subtractive color mixing (primary colors: cyan, magenta, yellow) or additive color mixing (primary colors: red, green, blue) or a "premixed" palette (a faster way of choosing colors), to be used throughout the painting as the basis for mixing paints. For a simulation of tempera painting, the artist should specify the subtractive color mixing method, since tempera paints are opaque pigments that absorb (i.e. subtract) light. The color-mixing algorithm the program employs blends colors to produce the same color as would result if real paints were mixed.

The artist switches between the "palette" mode and the "painting" mode by pressing the ENTER button on the trackball twice. When the program reads the same position twice in a row, it takes this as an indication that a mode change is desired.

While in the "palette" mode, use the trackball in this way: turn the TRACK switch on the trackball OFF. To select a color, superimpose the cursor on the desired color and press the ENTER button once. To select a control option (one of the four boxes with writing), position the cursor inside the box you are choosing, and press the ENTER button once.

While in the "painting" mode, the ends (width) of the "paintbrush" are defined by the positions of the two cursors. To adjust width of brush, turn TRACK switch off. Turn either of the two cursors off (they are controlled by the toggle switches labeled "1" and "2" on the trackball.) Move trackball to reposition the cursors, and then turn the cursors back on. To adjust the position of the brush (to move the brush without painting): turn TRACK switch off. Spin the trackball to move the "brush" to desired place. Turn TRACK switch on. Moving the trackball now will start painting at the cursors.

AUTHOR

Marshall Schaffer.

FILES

Source program: cartridge #16 -- /a/class/marshal.a
temporary file will be created under present working directory (=pwd): "pwd"/quarter

SEE ALSO

gr(IV)

DIAGNOSTICS

none

NAME

thresh - Interactive threshold program

DESCRIPTION

Using the TRACK ball unit the user can select the most pleasing threshold of an image with this program. The horizontal screen position of cursor #1 is used to select the current threshold. By moving this cursor to the left or right the threshold will move down or up the grayscale accordingly. The keyboard <RUBOUT> key will terminate the program.

USAGE

```
thresh [[b] w]
       b -- value to use for 'black' (0 is default)
       w -- value to use for 'white' (4095 is default)
```

EXAMPLE

```
% thresh
      {cursor moved to most pleasing threshold}
   <RUBOUT>

   Eight bit threshold = XXXX

   Six bit threshold = XXXX
```

AUTHOR

Russ Smith

NAME

title - type text onto the Grinnell using fonts

SYNOPSIS

```
title <fontname> [-]
```

DESCRIPTION

Title allows you to type on the screen of the Grinnell using a font which is stored in standard Unix font file format. The initial font to use is specified by the argument. If the file is not found, title appends ``.fnt'' to the name and looks again. If this doesn't work the system font library is searched. The color used is initially set to white. The font may be changed at any time by typing control-C and answering the prompt. The color may be changed in the same manner by typing control-P.

Placement of the characters is controlled by cursor #1 which may be moved with the track ball, or the four arrow keys surrounding the ``*home*'' key on the Datamedia keyboards. The position of the cursor is updated after each character, according to the character's width. A carriage-return moves the cursor to the beginning of the next line. *Title* normally puts the cursor somewhere in the upper left hand corner of the screen when it starts; the exact position depends on the size of the font. The optional ``-'' argument causes *title* to leave the cursor wherever it finds it.

ASCII control characters that have printing characters defined for them in the font may be typed directly if *title* doesn't interpret them specially. They can also be entered by preceding the equivalent printing character with a percent sign. (i. e., ``%a'' for control-A.) To enter a single percent sign, type two of them.

The program may be exited by typing break, control-D, or rubout.

AUTHOR

Lee Moore. Modified by Fred Blonder.

FILES

*.fnt - font file
/b/fonts - system font library
/dev/gr - Grinnell

SEE ALSO

font(III), font(V), pgp(I), biglet(I), descfnt(I)

DIAGNOSTICS

You will be told if title can't access the font file you specify, or if you type a character that is not defined in that particular font.

BUGS

Some fonts don't put the cursor in the expected position.

The overlay cannot be used directly.

Space and tab may generate unexpected cursor movements or display characters. In particular, typing space may place the cursor at the left hand margin. Thus, only the ``arrow'' keys may be used for cursor placement.

NAME

tprint - Print out a thresholded display

DESCRIPTION

Tprint is used to make a hard copy of the displayed image after it has been thresholded (i.e., through the lookup table) or of the overlay for the PRINTRONIX line printer. Depending on the number of arguments the resulting print will be of the thresholded image, the complement of the thresholded image or the complement of the overlay. Only the overlay may be printed without thresholding, the results being undefined for a non-thresholded image (see *ht*). The output may be piped.

USAGE

```
tprint [a [a]] > /dev/lp
tprint [a [a]] ¦ opr
       no arguments - print thresholded image
       1 argument - print complement of thresholded image
       2 arguments - print complement of overlay
```

EXAMPLE

```
% thresh
       {Interactive thresholding of image takes place}
  <RUBOUT>
% tprint ¦ opr
       {Thresholded image printed on PRINTRONIX}
%
```

AUTHOR

Russ Smith

NAME

track - tracks regions in Grinnell pictures.

SYNOPSIS

track

DESCRIPTION

Track traces the borders of thresholded regions in a Grinnell image. The component to be tracked is selected manually by positioning cursor 1 inside a region and typing a threshold value. *Track* scans right from the cursor position until a below threshold point is found. Tracking begins at this point and continues counterclockwise around the boundary of the region. Border points are written to the Grinnell overlay so tracking may be followed visually. As many borders as desired may be tracked by repositioning the cursor and entering a threshold. Each border tracked outputs a string of integers seperated by blanks. The first two integers are the X Y coordinates of the starting point. Following the start point is the four-neighbor chain code for the border. A -1 terminates the string for each border. Entering a negative threshold terminates *track* and causes an additional -1 to be printed as an end marker. The chain code directions are shown in the diagram below.

```
      1
      ^
      !
2<--+-->0
      !
      v
      3
```

FILES

/dev/gr

AUTHOR

Wallace S. Rutkowski

BUGS

The cursor must be positioned at a point whose gray level is at or above the threshold.

NAME

tv - Turn on video digitizer

DESCRIPTION

Tv is used to turn on and start input into the GRINNELL memory from the video digitizer. In usual practice the digitizer's input comes from the T.V. camera, but video disk and tape units can also be used as input devices. *Tv* will continue inputting data at the standard frame rate (30 frames/second) until *freeze* is used. Because *tv* configures the GRINNELL hardware registers in a certain way, it should always be followed by a *freeze*.

USAGE

```
tv
```

EXAMPLE

```
%  tv
          {Input from t.v. camera, tape or disk}
% freeze
          {Freeze the current frame}
%
```

AUTHOR

Russ Smith

NAME

vex - Random color vector generator (demonstration program)

DESCRIPTION

Vex is a demonstration program which will generate randomly oriented and colored vectors (or solid rectangles) on the GRINNELL display. The endpoints and the colors of the vectors are determined using the system random number generator.

USAGE

```
 vex [r]
        r -- generate rectangles, not vectors
```

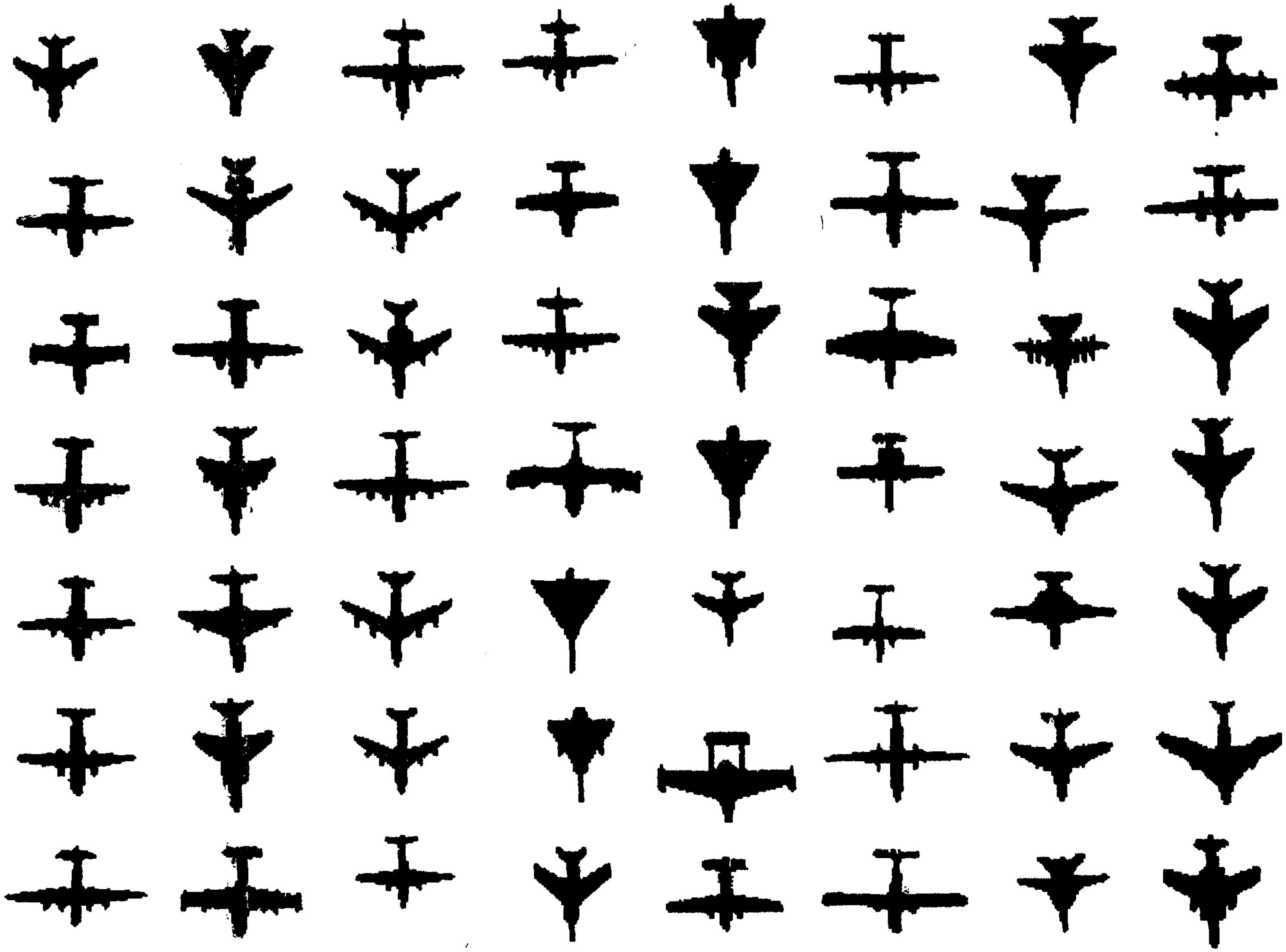
EXAMPLE

```
 % vex & vex r &
```

This user is running both versions of the vex program simultaneously. It makes an interesting display.

AUTHOR

Russ Smith



File Handling

NAME

bu, bu0, bu4 - back up magtape

SYNOPSIS

bu [-*special*] [-*N*] [*count*]

DESCRIPTION

Bu backs up magtape to the file before the current one. If a previous file exists, the tape is left positioned just after the end-of-file (EOF) or beginning of tape that begins the file. If the tape was already at the beginning of tape, then *bu* prints an error message.

Alternative names for *bu* such as *bu4* provide a different default tape device according to the last digit of the name.

The options either specify a non-default backup device or provide for multiple file backup:

-special backs up the given special file instead of default ''/dev/bu_rmt0''.

-N backs up magtape raw device ''/dev/bu_rmtN'' where the one digit octal number N gives the unit number (default 0). If N is omitted 4 is used.

count backs up over count file marks instead of the default of one. A leading zero specifies an octal number. If only a zero is given, *bu* takes no action.

DIAGNOSTICS

Complains if the beginning of tape is reached before the file count is exhausted or if the tape drive is offline or already occupied. If interrupted, *bu* displays a count of files that have been backed up over.

FILES

/dev/bu_rmt?

SEE ALSO

tm(IV), rewind(I)

AUTHOR

Robert L. Kirby

BUGS

The backing-up version of the tape driver must be installed.

NAME

chead - change header of a picture file.

SYNOPSIS

chead picture [nc] [nr] [size]

DESCRIPTION

picture	CXAP picture
nc	number of columns in picture
nr	number of rows in picture
size	size of picture, where each pixel has 2 ^ (size) bits

Chead changes the header of a picture. The header of *picture* is initially read and is replaced with *nc*, *nr*, and *size*. If an argument is not specified, the corresponding parameter in the picture header is unchanged.

nc and *nr* must be greater than zero and 1 <= *size* <= 5.

FILES

/mnt/phil/cxap/prog/chead.c source code

DIAGNOSTICS

Many, all of which should be self-explanatory.

AUTHOR

Philip A. Dondes

SEE ALSO

HEADER(VII)

BUGS

NAME

dosfa - convert DOS formatted ASCII to UNIX standard ASCII

SYNOPSIS

dosfa

DESCRIPTION

Dosfa is a filter that converts DOS formatted ASCII input to UNIX standard ASCII output. The format resembles that used by DEC operating systems RT-11, DOS/BATCH-11, and RSX-11. After eliminating any parity bit, each carriage return (015) line feed (012) pair is converted to a single line feed (012). NULL (0) characters are also eliminated. All other characters are passed through as is. _Dosfa_ eliminates the unnecessary carriage returns read by _rdostape_ from tapes produced by DOS.

DIAGNOSTICS

none

FILES

SEE ALSO

rdostape(VI), tm(IV), DEC's DOS/BATCH Handbook

AUTHOR

Robert L. Kirby

BUGS

NAME

dosfb - convert DOS formatted binary

SYNOPSIS

dosfb

DESCRIPTION

Dosfb is a filter that converts DOS formatted binary input to raw data output eliminating the formatting information. This input format produced by DEC operating systems RT-11, DOS/BATCH-11, and RSX-11 can be read by _rdostape_. _Dostape_ can write in this format to allow for the accurate retrieval of raw data from DOS format tapes.

DIAGNOSTICS

A new formatted record begins with the first byte containing 1. Other bytes between records are ignored. The checksum information is not verified. If the final record is incomplete, it is dumped as is.

FILES

SEE ALSO

dostape(VI), rdostape(VI), tm(IV), DEC's DOS/BATCH Handbook

AUTHOR

Robert L. Kirby

BUGS

Checksum errors and non-zero inter-record bytes should be reported.

NAME

dostape - write DEC DOS-PIP Format Tape

SYNOPSIS

dostape [-*special*] [-*N*] [-*n* alias] [-*uN*] [-*gN*] [-*pN*] [-*a*] [-*b*] [-*fN*] [-*hN*] [file] ...

DESCRIPTION

Dostape writes files in DEC's DOS/BATCH-11 Peripheral Interchange Program (PIP) magtape format. The format written by *dostape* may be read by DEC operating systems RT-11, DOS/BATCH-11, and RSX-11. The output tape must have been prepositioned using the Harvard non-rewinding magtape device. *Dostape* individually writes files converting each name into a sometimes abreviated DOS equivalent. If there are no file parameters, *dostape* reads its standard input using the default name "UNIX.OUT". *Dostape* normally leaves the tape positioned just after the file mark for the last file written, in position for a subsequent *dostape* invocation or for marking with two additional file marks which constitute the DOS end-of-tape convention.

Each DOS-PIP file on magtape consists of a 14 byte header record, 512-byte data records, and one file mark. The first three words of the header consist of a radix-50 encoded name and extension using up to 6 characters for the name and 3 characters for the extension. *Dostape* places only the final part of each path name into the header omitting any previous directory names and slashes, treating lower case a-z as upper case A-Z. From this, *dostape* uses the first 6 filename characters before the last dot(.), if any, for the encoded file name and the first three characters after the last dot as the extension. If the final part contains no dot, the first three characters in excess of the first six become the extension. The next two bytes are the user and group owner numbers, the next word is the protection code, and the last two words are the Julian date based on 1970 computed from the file modify date. Logical records may extend across the 512-byte physical tape records. *Dostape* pads the last physical record with NULLs, i. e. zero bytes, up to 512 bytes.

Options, except aliasing, apply to all the following files but may be reset between files:

-special writes to given special file instead of default ``/dev/nrw_rmt0''.

-N write to non-rewinding magtape raw device ``/dev/nrw_rmtN'' where the octal number N gives the unit number (default 0).

-n alias treat the next argument as an alias in the the next header in place of the actual file name.

-uN place user ID N in header. If N is omitted, the file owner is used (default).

-gN place user group ID N in header. If N is omitted, the file group owner is used (default).

-pN place octal DOS protection code N in header. If N is omitted, use the default 0233.

-a produce DOS formatted ASCII output by inserting a carriage return (015) before each line feed (012).

-b produce DOS binary unformatted output (default) that copies bytes as is. The last record may be null padded.

-fN produce DOS formatted binary output using input logical record size N. If N is omitted, use 8192 bytes. Prefixes each logical record with the format word 000001 and a record byte count that includes the prefix words. Suffixes each record with an additive one byte checksum and NULL padding to a word boundary. The final logical record may be foreshortened. The padding does not preclude the exact recovery of formatted binary data.

-hN write one formatted binary header record of length N before the other formatted binary records. This record is followed by zero padding to fill out a physical record. If N is omitted, no header record is written.

For example

```
     dostape   -4 /tmp/verylongname  -a  -g1  /tmp/short.c.s
               -h6 -f255 -n fakename.pic /tmp/pic
```

writes three files to device /dev/nrw_rmt4. The first unformatted file is called ``VERYLO.NGN'' in the tape header. The second formatted ASCII file called ``SHORT.S'' precedes each line feed with a carriage return. The third formatted binary file called ``FAKENA.PIC'' uses the data in /tmp/pic to produce a 6 byte logical record followed by 255 byte logical records appropriate to images with 6 byte headers. The last two files use group 1 instead of the original file group owner.

DIAGNOSTICS

Physcial data I/O errors generate appropriate messages. Expansion troubles indicate inadequate buffer space. Illegal radix-50 characters are converted to the ``unused'' character (035).

FILES

/dev/nrw_rmt? (default output device)
unix.out (default tape header name)

SEE ALSO

ldostape(VI), rdostape(VI), tm(IV), DEC's DOS/BATCH Handbook

AUTHORS

Russ Smith
Robert L. Kirby

NAME

eot, eot0, eot4 - advance magtape to end of tape mark

SYNOPSIS

eot [-*special*] [-[*N*]] [*count*]

DESCRIPTION

Eot advances the tape to the next software end of tape mark that consists of two file marks. When using a standard tape name, *eot* then moves the tape backward to be immediately after the first of the two marks. From this position the tape can be extended with another file. Otherwise the tape remains positioned immediately after both marks. A count of the number of file marks passed is displayed.

Alternative names for *eot* such as *eot4* provide a different default tape device according to the last digit of the name.

The options either specify a non-default device or limit the number of file marks that may be passed:

-special advances the given special file instead of default ''/dev/nrw_rmt0''. No backup is performed after advancing.

-N advances the magtape raw device ''/dev/nrw_rmtN'' where the one digit octal number N gives the unit number (default 0). Afterward the tape is backed up to between the file marks using back up device ''/dev/bu_rmtN''. If N is omitted 4 is used.

count limits the number of file marks passed while searching for the software end-of-tape. A leading zero specifies an octal number. If only a zero is given, *eot* takes no action.

DIAGNOSTICS

Complains if the tape drive is offline, already occupied, or if reading the first record of any file being passed generates a hardware error. If interrupted, *eot* displays a count of file marks to be passed before stopping.

FILES

/dev/nrw_rmt? /dev/bu_rmt?

SEE ALSO

tm(IV), bu(I), skp(I), rewind(I)

AUTHOR

Robert L. Kirby

BUGS

The backing-up version of the tape driver must be installed

If the first record of any file passed is longer than 44544 bytes, a hardware record length error occurs.

NAME

ldostape - list directory for DEC DOS-PIP Format Tape

SYNOPSIS

ldostape [-*special*] [-*N*] [-*cN*] [-*uN*] [-*gN*]
[file] ...

DESCRIPTION

Ldostape lists a directory for magtape in DEC's DOS/BATCH-11 Peripheral Interchange Program (PIP) format starting at the current tape position. The format resembles that used by DEC operating systems RT-11, DOS/BATCH-11, and RSX-11. For each file, *ldostape* converts the radix-50 filename in the header into six ASCII characters followed by a dot(.) and three characters for the extension. Alphas are converted to lower case. The radix-50 codes 0, 033, 034, and 035 are converted into blank( ), dash(-), dot(.), and question mark(?) respectively. The extension is followed by a user identification code (UIC) in brackets, the octal protection code in angle brackets, and a decimal version of the Julian date based on 1970.

The tape is left positioned after the tape mark for the last file read or immediately after the double file mark at end-of-tape (EOT).

The options either specify a non-default input device or limit the files to be included in the directory listing:

-special search the given special file instead of default ''/dev/nrw_rmt0''.

-N search non-rewinding magtape raw device ''/dev/nrw_rmtN'' where the one digit octal number N gives the unit number (default 0).

-cN only list information for the next N files.

-uN only print information for files with user ID N in the header skipping files with other IDs.

-gN only print information for files with user group ID N in the header skipping files with other IDs.

DIAGNOSTICS

Physical I/O data errors generate appropriate messages.

FILES

/dev/nrw-rmt?

SEE ALSO

dostape(VI), rdostape(VI), tm(IV), DEC's DOS/BATCH Handbook

AUTHOR

Robert L. Kirby

NAME

pack - packs an unpacked picture.

SYNOPSIS

pack [sub3]

DESCRIPTION

sub3 if present, 3 is subtracted from the input picture's size before the output picture's header is written

PACK reads a picture's header from the standard input file and echoes it to the standard output file. PACK then reads the unpacked picture from the standard input file one row at a time, packs the row and writes it to the standard output file.

The input picture must be comprised of two sections, a header section and a data section. The header is a six word record of which two types exist. Type 1 specifies as a picture's size the number of bytes which comprise a pixel, whereas type 2 specifies the number of bits a pixel is to use. To obtain a type 1 header, the sub3 argument should be given. This subtracts 3 from the input picture's size, which should previously have been 4 or 5, and thereby effectively creating a picture with a type 1 header. Note that if the input picture's size is already 1 or 2, it is treated as either a binary or a 2-bits per pixel picture. The following is the description of the header:

word 0	unused,
word 1	# of columns in picture,
word 2	unused,
word 3	# of rows in picture,
word 4	unused,
word 5	size of picture, where each pixel is represented by 1 or 2 bytes, as in type 1, or where each pixel is represented by 2^(SIZE-1) bits, as in the type 2, where SIZE is the picture's size. 1<=SIZE<=5.

The data section is a n x m word stream of integer pictorial data, where n is the number of rows and m is the number of columns in the picture. Both unsigned and signed integer representation may be used. If we let PIXWD = 16/(2^(SIZE-1)), where SIZE relates to the type 2 header format, represent the number of pixels per word in the packed format, the length of an output row (in bytes) is calculated by the C-Language expression (m/PIXWD)*2)+(m%PIXWD>0?1:0).

The number of columns an input row may have is MAXLEN, where MAXLEN is currently 512.

FILES

DIAGNOSTICS

"Header not read"	first 12 bytes of picture not read
"Picture width too large"	# of columns exceeds MAXLEN
"Illegal size"	SIZE<=0 or SIZE>=5
"Too much data, row count expired"	# of rows in picture exceeded
"Unexpected EOF encountered"	eof encountered before picture completed
"Input pipe not empty"	data still resides in input pipe

AUTHOR

Philip A. Dondes

SEE ALSO

UNPACK(VI)

BUGS

NAME

rdostape - read DEC DOS-PIP Format Tape

SYNOPSIS

rdostape [-*special*] [-*N*] [-*sN*] [-*uN*] [-*gN*]
[file] ...

DESCRIPTION

Rdostape reads files in DEC's DOS/BATCH-11 Peripheral Interchange Program (PIP) magtape format. If parameters specify a particular file name or UIC, the tape is first searched forward until a suitable file header is found or a double file mark (EOT) is reached. If the appropriate file is found, *rdostape* bypasses the header record and passes the remaining records to the standard output as is, regardless of the length of the records (up to 44544 bytes) or the format used to write them. If no file is specified, the file at the current tape position is read, omitting the header record. After a file is transfered, the tape is left positioned after the file mark that ends the file. If no satisfactory file is found, the tape is positioned just after the double file mark that defines end of tape (EOT).

Rdostape converts file names into a sometimes abreviated DOS equivalent. The first three words of the DOS-PIP header consist of a radix-50 encoded name and extension using up to 6 character for the name and 3 characters for the extension. *Rdostape* examines only the final part of each path name omitting any previous directory names and slashes, treating lower case a-z as upper case A-Z. From this, *rdostape* uses the first 6 filename characters before the last dot(.), if any, to compare with the encoded file name and the first three characters after the last dot as the extension. If the final part contains no dot, the first three characters in excess of the first six become the extension. The next two bytes are the user and group owner numbers. The format used may also be read by DEC operating systems RT-11, DOS/BATCH-11, and RSX-11.

The options specify either a particular file or non-default input device:

-special read from given special file instead of default ``/dev/nrw_rmt0''.

-N read from non-rewinding magtape raw device ``/dev/nrw_rmtN'' where the one digit octal number N gives the unit number (default 0).

-uN search for user ID N in header skipping files with other IDs.

-gN search for user group ID N in header skipping files with other IDs.

-sN seek past N physical records before reading tape. If N is zero or omitted, the first record read will be the first in the file even if it is the header record. Thus by not specifying a particular file, any format data may be read as is.

DIAGNOSTICS

Physical I/O data errors generate appropriate messages. Illegal radix-50 characters are converted to the ``unused'' character (035).

FILES

/dev/nrw-rmt?

SEE ALSO

dostape(VI), ldostape(VI), dosfa(VI), dosfb(VI), tm(IV), DEC's DOS/BATCH Handbook

AUTHOR

Robert L. Kirby

BUGS

Should be able to output to files other than standard output.

NAME

rewind - rewind magtape

SYNOPSIS

rewind [*special*] [*N*] [-]

DESCRIPTION

Rewind rewinds magtape back to the beginning of tape. *Rewind* tries to rewind using both the old and new tape drivers.

Alternative names for *rewind* such as *rewind4* provide a different default tape device according to the last digit of the name.

The options specify a non-default backup devices:

special rewinds the given special file instead of default ``/dev/rw_rmt0'' and then ``/dev/rmt0''.

N backs up magtape raw device ``/dev/rw_rmtN'' where the one digit octal number N gives the unit number (default 0).

- backs up magtape raw device ``/dev/rw_rmt4''.

DIAGNOSTICS

Complains if the tape drive is offline or already occupied. I/O errors from the previous close are ignored.

FILES

/dev/rw_rmt? /dev/rmt?

SEE ALSO

tm(IV), bu(I)

AUTHOR

Robert L. Kirby

BUGS

NAME

skp - skip tape files

SYNOPSIS

<u>skp0</u> #-of-files
<u>skp4</u> #-of-files

DESCRIPTION

Either <u>skp0</u> for 800bpi tapes or <u>skp4</u> for 1600bpi tapes will skip over the number of tape EOF marks (i.e. files) specified on the command line.

FILES

all the special tape devices (/dev/srmt0, /dev/srmt4, etc.)

SEE ALSO

tm (IV)

NAME

unpack - unpacks a packed picture.

SYNOPSIS

unpack [add3]

DESCRIPTION

add3 if present, 3 is added to the input picture's size before the output picture's header is written

UNPACK reads a picture's header from the standard input file, adds 3 to the picture's size if add3 is present, and writes it to the standard output file. UNPACK then reads the packed picture from the standard input file one row at a time, unpacks each pixel into a word, and writes the row to the standard output file.

The input picture must be comprised of two sections, a header section and a data section. The header is a six word record of which two types exist. Type 1 specifies as a picture's size the number of bytes which comprise a pixel, whereas type 2 specifies the number of bits a pixel is to use. When unpacking a picture with a type 1 header the add3 argument should be given so that 3 will added to the picture's size. This in effect treats the input picture as a size 4 or 5 picture and thereby treating each pixel as 8 or 16 bits, respectively. The following is the description of the header:

word 0	unused,
word 1	# of columns in picture,
word 2	unused,
word 3	# of rows in picture,
word 4	unused,
word 5	size of picture, where each pixel is represented by 1 or 2 bytes, as in type 1, or where each pixel is represented by 2^(SIZE-1) bits, as in the type 2, where SIZE is the picture's size. 1<=SIZE<=5.

The data section is n rows of packed pictorial data. If we let m be the number of columns in the unpacked picture format and PIXWD = 16/(2^(SIZE-1)), where SIZE relates to the type 2 header format, be the number of pixels per word in the packed format, the length of a packed row (in bytes) is calculated by the C-Language expression (m/PIXWD)*2)+(m%PIXWD>0?1:0). The lengths of each input row must all be equal and needless to say, the number of rows and columns in the picture must be the same as described by the picture's header.

The number of columns an input row may have is MAXLEN, where MAXLEN is currently 512.

FILES

DIAGNOSTICS

"Header not read"	first 12 bytes of picture not read
"Picture width too large"	# of columns exceeds MAXLEN
"Illegal size"	SIZE<=0 or SIZE>=5
"Too much data, row count expired"	# of rows in picture exceeded
"Unexpected EOF encountered"	eof encountered before picture completed
"Input pipe not empty"	data still resides in input pipe

AUTHOR

Philip A. Dondes

SEE ALSO

PACK(VI)

BUGS

NAME

xapin - read a picture file from magnetic tape

SYNOPSIS

xapin tape file [fc] [fr] [nc] [nr] [ci] [ri]

DESCRIPTION

tape — input tape file, either 9-800 bpi or 9-1600 bpi (special tape files are recommended)

file — output disk file (will be created if not already present)

fc — first column with which to begin reading input picture

fr — first row with which to begin reading input picture

nc — number of columns to read from input picture

nr — number of rows to read from input picture

ci — increment between columns (allows for sampling of columns)

ri — increment between rows (allows sampling of rows)

Xapin reads a raw-formatted picture from tape devices /dev/srmt? or /dev/rmt?. The resulting output picture will be located in *file* in the conventional CXAP format.

All arguments which are not given are defaulted. *fc*, *fr*, *ci*, and *ri* are defaulted to 1. *nc* is set the length of the first picture row on the input tape if left defaulted. It is crucial that all records on the input tape be the same length because no checking is performed to ensure this *nr* is defaulted to 1024.

Each pixel will be read from the input tape as one byte of information and converted to integer as defined by C. The picture byte size will be 4, indicating 8 bits per pixel.

If the default or user supplied values for *nc* and *nr* exceed the user window, they are changed to reflect the actual size of the input picture.

All end-of-file marks terminate *xapin* normally, with some parameters (as mentioned above) being modified as needed. Errors terminate *xapin* abnormally resulting in the output of an error message and the status of the last executed function call.

FILES

/mnt/phil/cxap/prog/xapin.c — source code

/mnt/phil/cxap/prog/ioinfo.c — inputs information for program

DIAGNOSTICS

Many, all of which should be self-explanatory.

AUTHOR

Philip A. Dondes

SEE ALSO

CXAP(VII)
IOINFO(VII)

BUGS

NAME

xapout - write a picture to magnetic tape

SYNOPSIS

xapout file tape [fc] [fr] [nc] [nr] [shift]

DESCRIPTION

file CXAP picture file to be written to tape

tape output tape file, either 9-800 bpi or 9-1600 bpi (special tape files are recommended)

fc first column with which to begin reading input picture

fr first row with which to begin reading input picture

nc number of columns to read from input picture

nr number of rows to read from input picture

shift each pixel is shifted 2^shift amount before being written

Xapout writes a CXAP picture from disk to either tape device /dev/srmt? or /dev/rmt?.

All arguments which are not given are defaulted. *fc* and *fr* are defaulted to 1. *nc* and *nr* are defaulted to the actual dimensions of the input picture. *shift* is zero if not specified.

Each pixel in *file* is written as one byte of information if the size of the picture associated with *file* is not equal to 5. Otherwise, two bytes will be written for each pixel.

FILES

/mnt/phil/cxap/prog/xapout.c	source code
/mnt/phil/cxap/prog/ioinfo.c	inputs information for program

DIAGNOSTICS

Many, all of which should be self-explanatory.

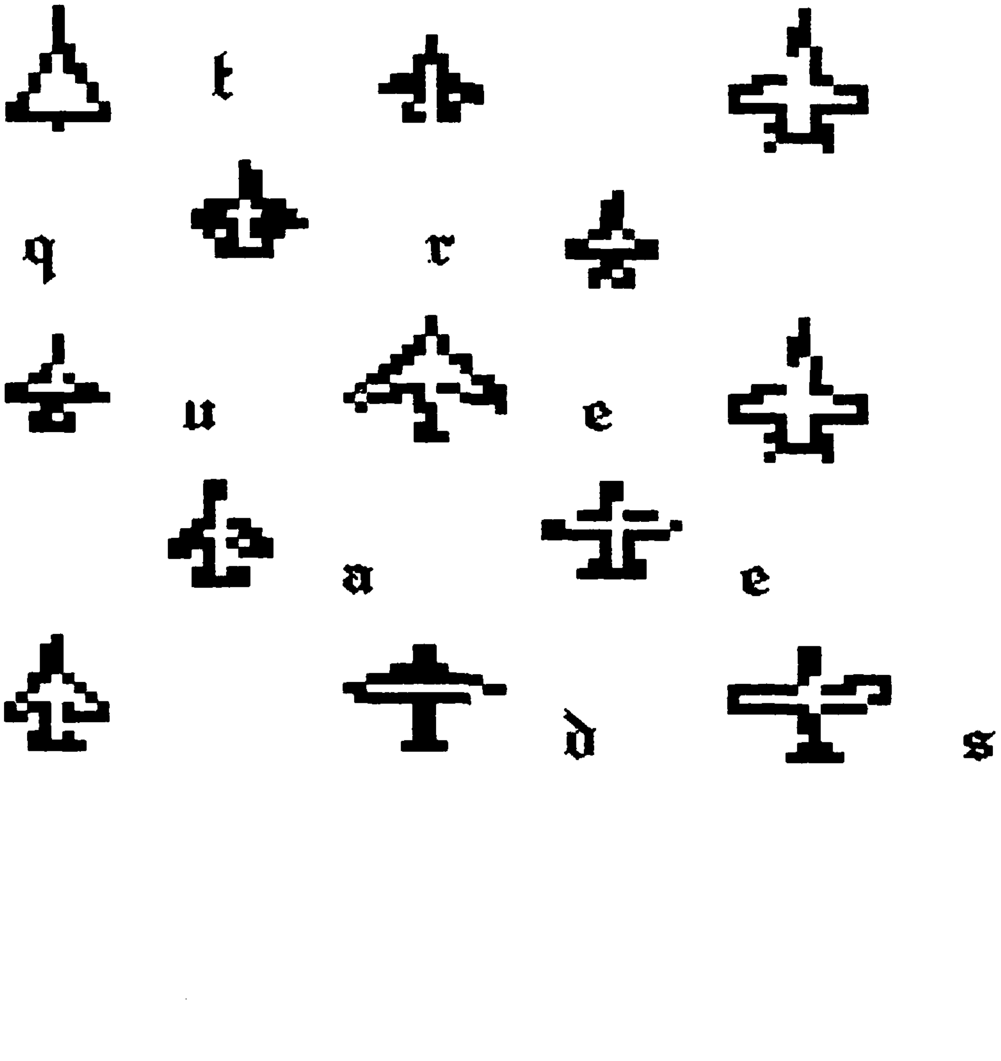
AUTHOR

Philip A. Dondes

SEE ALSO

CXAP(VII)
IOINFO(VII)

BUGS


t
q
r
u
e
a
e
d
s

NAME

quadtrees - description of quadtree structure

DESCRIPTION

A set of programs is available to create, display and compute certain properties of "quadtrees" representing 64 x 64 GAP pictures. Quadtrees handled by these programs are completely "portable", i.e. they can be passed around the system in the same way as files. New operations on quadtrees can be programmed easily by keeping to the basic data structure described below and using some of the routines provided.

A quadtree consists of linked nodes, where each node represents a quadrant of a binary image which is either black, white, or "gray". A node corresponding to a gray quadrant has four son nodes. This kind of structure can be represented by a linked list of cells, each corresponding to a node in the tree.

The cell format used here is as follows : Word 0 contains the average gray level of the quadrant. Thus a value 0 or 63 in this word indicates that this is a terminal or "leaf" node (all black or white) and has no sons. Word 1 contains the level of the node in the tree, the root being level 0. Word 2 contains a pointer to the "father" of the node. Pointers are addresses in the machine's memory corresponding to Word 0 of the node cell. In the case of the root, this pointer points to Word 0 of the root cell. In the case of a gray node, Words 3-6 contain a pointer to each of the 4 sons of this node. The order of the sons is NW , NE , SE , SW. In the case of leaf nodes, these words are zero.

If the quadtree is to be roped, the node cell size is increased by 8, and these additional words contain pointers to the 8 neighbors of the node. Neighbors are only defined for black nodes. If N is a black node, a neighbor of N is defined as another black node which lies adjacent to it and touches one of its corners. Neighbor 1 of N, for example, is a black node which is adjacent to the North side of N and which touches its NW corner. Neighbor 2 is a black node which is adjacent to N 's North side and which touches its NE corner.....neighbor 8 is a black node which is adjacent to N's West side and which touches its NW corner. When no neighbor exists the corresponding pointer is set to -1.

Each quadtree file contains a fixed header of 5 words. Word 1 contains the number of disc blocks required by the tree file. Words 2 and 3 contain the values of the black and white gray levels respectively. These are defaulted to 0 and 63 , but can easily be changed to compress the gray level range. Such a compression would have the effect of causing nodes which are 'mostly' black to be regarded as black and those which are mostly white to be regarded as white. Word 4 contains the size of each cell in the tree. Word 5 is unused at present.

AUTHOR

Sanjay Ranade

SEE ALSO

GAP(VI), QMAKE(V), QDISP(V) etc.

BUGS

If the image is noisy, the size of the quadtree will be large. If this size exceeds about 12K, no tree will be made. Unfortunately, at the moment, trees are not optimally constructed and each node requires the same amount of space. Alternative definitions of a neighbor are possible but have not been implemented.

NAME

qrope - rope a quadtree

SYNOPSIS

qrope tree

DESCRIPTION

tree quadtree file created with the 'r' option by QMAKE VI

qrope ropes a quadtree created by QMAKE. Black leaf nodes now contain pointers to their adjacent neighbor nodes.

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees (V)

BUGS

NAME

qp - profile a quadtree

SYNOPSIS

qp tree

DESCRIPTION

tree   quadtree file

qp prints out the number of black,white,gray and marked nodes in the tree and the storage required by each type. Marked nodes are nodes which have been been selected by some other program on the basis of a particular property which they possess (e.g. maximal nodes).

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees (V)

BUGS

NAME

qcg - find centroid of shape represented by quadtree

SYNOPSIS

qcg tree [bg]

DESCRIPTION

tree	quadtree file created by QMAKE(VI)
b	flag to specify inside approximation to shape
g	flag to specify outside approximation to shape

qcg finds the centroids of successive approximations to the shape represented by the quadtree. The 'b' option specifies that black nodes up to a particular level be used for the approximations. The 'g' option specifies black nodes up to a particular level and gray nodes at that level.

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees (V)

BUGS

NAME

qcm - find central moments of a shape represented by a quadtree

SYNOPSIS

qcm tree [bg] I J

DESCRIPTION

tree quadtree file created option by QMAKE VI
b flag to specify inside approximation to shape
g flag to specify outside approximation to shape
I,J specifies particular central moment to be found

qcm finds the I,J central moment of successive approximations to the shape represented by the quadtree. The 'b' option specifies that black nodes up to a particular level be used for the approximations. The 'g' option specifies black nodes up to a particular level and gray nodes at that level.

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees (V), CM (VI)

BUGS

NAME

qdisp - displays a quadtree on the Grinnell display

SYNOPSIS

qdisp tree x y [m]

DESCRIPTION

tree	quadtree file
x, y	integers specifying the base of the 64 x 64 Grinnell window in which the quadtree is to be displayed.
m	optional. If specified, only the marked nodes will be displayed.

qdisp displays the black nodes of a quadtree. The optional argument m is specified to display marked nodes only. This is useful for examining maximal nodes, nodes at particular levels, etc.

FILES

/dev/gr

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees (V)

BUGS

NAME

qcom - complement a quadtree

SYNOPSIS

qcom tree1 tree2

DESCRIPTION

tree1	existing quadtree file
tree2	complemented tree to be created

qcom complements a quadtree , i.e. black nodes in tree1 will be white nodes in tree2 etc

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees (V)

BUGS

Tree2 will have the same roping as tree1. This means that if subsequent processing needs roping, tree2 must be re-roped.

NAME

qdispr - row by row quadtree display

SYNOPSIS

qdispr tree file [m]

DESCRIPTION

tree quadtree file

file file to which the 64 x 64 picture will be output in GAP format

m if specified, only the marked nodes will be displayed.

qdispr displays the black nodes of a quadtree. The optional argument m is specified to display marked nodes only. Useful for examining maximal nodes, etc. The picture is output with the GAP header.

FILES

/dev/gr

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees(V)

BUGS

The algorithm used does repeated tree traversal and is therefore quite inefficient.

NAME

qdump - dump specified number of quadtree node cells

SYNOPSIS

qdump tree

DESCRIPTION

tree	quadtree file
ncells	number of cells to dump
stcell	the cell number from which to start the dump

qdump prints the address of each node cell followed oy its contents. The address printed out is relative to the start address of the root cell (i.e. address 0).

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees (V)

BUGS

NAME

qmake - make quadtree from a binary 64 x 64 GAP picture

SYNOPSIS

qmake pic tree [rbw]

DESCRIPTION

pic a 64 x 64 binary image in GAP format
tree quadtree file which will be created
r will enable the tree to be roped
b black gray level value. If not specified, defaulted to 0.
w white gray level value. If not specified, defaulted to 63.

qmake makes a quadtree from the specified picture and prints out the number of white, black and gray nodes in the tree and the storage required by it. The 'r' option will increase the size of each node to accommodate roping information.

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees(V)

BUGS

If the image is noisy, the size of the quadtree will be large. If this size exceeds about 12K, no tree will be made.

NAME

ql - mark all nodes of a quadtree which are at the specified level

SYNOPSIS

ql tree n

DESCRIPTION

tree	quadtree file
n	an integer in the range 0 - 6

ql marks all the black nodes at level n of the quadtree. A mark is simply a flag set at the node cell.

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees (V)

BUGS

NAME

qmax - mark maximal nodes of a quadtree

SYNOPSIS

<u>qmax</u> tree n

DESCRIPTION

```
tree    quadtree file
n       an integer in the range 0 - 5
```

<u>qmax</u> marks the maximal nodes of a quadtree. If n is specified, the definition of a maximal node is extended, so that a node is maximal if it has no adjacent node n sizes greater than it.

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees (V)

BUGS

NAME

qout - collapse a quadtree

SYNOPSIS

qout tree file

DESCRIPTION

tree	quadtree file
file	file containing the collapsed tree .

qout outputs a collapsed version of the quadtree. The format of the output file is ' int file[20][3] ' . The first two words of a row contain the coordinates of the bottom left corner of the node. The third word contains the level of the node in the tree.

FILES

DIAGNOSTICS

a few, all are clear and self-explanatory.

AUTHOR

Sanjay Ranade

SEE ALSO

quadtrees (V)

BUGS

The table size is limited to 20, but can easily be changed if required.

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER TR-810	2. GOVT ACCESSION NO. AD-A086 098	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Interfaces, Subroutines, and Programs for the Grinnell GMR-27 Display Processor on a PDP-11/45 with the UNIX Operating System		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER TR-810
7. AUTHOR(s) Robert L. Kirby, Russ Smith, Philip A. Dondes, Sanjay Ranade, Les Kitchen, Fred Blonder		8. CONTRACT OR GRANT NUMBER(s) DAAG-53-76C-0138
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Center University of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Night Vision Laboratory Fort Belvoir, VA 22060		12. REPORT DATE October 1979
		13. NUMBER OF PAGES 167
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing	PDP-11/45
Displays	UNIX
Software	Grinnell GMR-27
Programs	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The specialized device interfaces for the University of Maryland Computer Vision Laboratory image acquisition and display equipment extend the capabilities of a PDP-11/45 hosting the UNIX operating system. The devices include the Grinnell GMR-27 color display processor, the other Computer Vision Laboratory display and scanning equipment, and the Digi-Data TM-11/TU-16 compatible tape drive. Subroutine packages give easy access to the interfaces from user programs, allowing full use of the special features. Programs using these subroutine packages and the well-designed UNIX operating system provide a flexible and powerful environment for image processing and program development. Short descriptions of these interfaces, subroutines, and programs are given for program writers and other users.