The DARPA/DMA Image Understanding Testbed
USER'S MANUAL

Version 1.1

Technical Note 277

January 1984

By: Andrew J. Hanson, Senior Computer Scientist

Artificial Intelligence Center
Computer Science and Technology Division

Approved for public release
Distribution unlimited

SRI Project 1009

The work reported herein was supported by the Defense
Advanced Research Projects Agency under Contract No.
MDA903-79-C-0588.

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486
The DARPA/DMA Image Understanding Testbed
USER’S MANUAL

Version 1.1

by Andrew J. Hanson
Artificial Intelligence Center
SRI International
Preface

The primary purpose of the Image Understanding (IU) Testbed is to provide a means for transferring technology from the DARPA-sponsored IU research program to DIA and other organizations in the defense community.

The approach taken to achieve this purpose has two components:

1. The establishment of a uniform environment that will be as compatible as possible with the environments of research centers at universities participating in the IU program. Thus, organizations obtaining copies of the Testbed can receive a flow of new results derived from ongoing research.

2. The acquisition, integration, testing, and evaluation of selected scene analysis techniques that represent mature examples of generic areas of research activity. These contributions from participants in the IU program will allow organizations with Testbed copies to immediately begin investigating potential applications of IU technology to problems in automated cartography and other areas of scene analysis.

The IU Testbed project was carried out under DARPA Contract No. MDA903-79-C-0588. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States government.

This document presents a user's view of the IU Testbed and the facilities it provides. Many talented people, both at SRI and at each of the contributing institutions, must be acknowledged for their part in bringing the Testbed into existence. Special recognition is due to David Kashtan and Kenneth Laws for their essential contributions to the environment described here.

Andrew J. Hanson
Testbed Coordinator
Artificial Intelligence Center
SRI International
Abstract

This manual is intended to help users of the Image Understanding Testbed system understand the structure and major features of the environment. Separate chapters are devoted to getting started as a new Testbed user, the UNIX, FRANZ LISP, and EMACS programming systems, Testbed applications programs, and Testbed utility systems. Appendices contain descriptions of demonstrations, picture and graphics utilities, the file system structure, and some elementary programming examples.
# Table of Contents

Preface .................................................................................................................. i 

Abstract ............................................................................................................... ii 

1. Introduction ..................................................................................................... 1 
   1.1. Documentation ......................................................................................... 1 
   1.2. Hardware Configuration .......................................................................... 2 
   1.3. Operating System Software .................................................................... 4 
   1.4. Languages ............................................................................................... 4 

2. Getting Started on the Testbed ...................................................................... 5 

3. UNIX and the C Programming Environment .................................................. 9 
   3.1. Introduction to UNIX ............................................................................. 9 
   3.2. Where to find UNIX system and Testbed files ...................................... 11 
   3.3. SYNOPSIS OF UNIX COMMANDS .................................................... 12 

4. The Testbed FRANZ LISP Environment ....................................................... 21 
   4.1. STARTING AND STOPPING FRANZ LISP ......................................... 21 
   4.2. UNIX FACILITIES ............................................................................... 21 
   4.3. Where to find LISP files ......................................................................... 22 
   4.4. The LEDIT System ................................................................................. 23 
   4.5. Testbed Demonstration Programs .......................................................... 24 
   4.6. Environment Utilities ............................................................................. 25 
       4.6.1. General Utilities ............................................................................. 25 
       4.6.2. EUNICE/VMS Utilities ................................................................. 27 

5. The Testbed EMACS Editor System ............................................................... 39 
   5.1. Where to find EMACS files .................................................................... 39 
   5.2. The EMACS Editor ............................................................................... 39 
   5.3. EMACS Personalization ......................................................................... 39 
   5.4. C programming in EMACS .................................................................... 39 
   5.5. INFO Utility .......................................................................................... 39 
   5.6. RMAIL Utility ....................................................................................... 39 
   5.7. FRANZ LISP Programming in EMACS ............................................... 39 

6. Testbed Applications Programs ...................................................................... 39 
   6.1. Overview of Applications Software Contributed to Testbed .................. 39 
   6.2. Carnegie-Mellon University Contributions ......................................... 39 
   6.3. University of Maryland Contributions ............................................... 42 
   6.4. MIT Contributions ............................................................................... 42 
   6.5. University of Rochester Contributions ................................................. 43 
   6.6. SRI Contributions ................................................................................ 43
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7. Stanford University Contributions</td>
<td>45</td>
</tr>
<tr>
<td>6.8. University of Southern California Contributions</td>
<td>46</td>
</tr>
<tr>
<td>7. Testbed Utility Programs</td>
<td>47</td>
</tr>
<tr>
<td>7.1. Image Manipulation Utilities</td>
<td>47</td>
</tr>
<tr>
<td>7.2. Image Display Utilities</td>
<td>47</td>
</tr>
<tr>
<td>7.3. User Libraries</td>
<td>48</td>
</tr>
<tr>
<td>7.4. Miscellaneous Packages</td>
<td>49</td>
</tr>
<tr>
<td>8. Guidelines for Programming in the Testbed Environment</td>
<td>51</td>
</tr>
<tr>
<td>8.1. Programming Style Guidelines</td>
<td>51</td>
</tr>
<tr>
<td>8.2. Program Installation Guidelines</td>
<td>52</td>
</tr>
<tr>
<td>Appendices</td>
<td></td>
</tr>
<tr>
<td>A. A Sampler of Testbed Demonstrations</td>
<td>55</td>
</tr>
<tr>
<td>B. The Testbed Picture File Access and Format</td>
<td>65</td>
</tr>
<tr>
<td>B.1. Picture Data Access</td>
<td>65</td>
</tr>
<tr>
<td>B.2. Picture File Format</td>
<td>65</td>
</tr>
<tr>
<td>B.2.1. Picture header data from /nu/tb/include/tbhdr.h</td>
<td>66</td>
</tr>
<tr>
<td>B.2.2. Octal Dump of Typical Picture Header</td>
<td>66</td>
</tr>
<tr>
<td>C. Graphics System Summary</td>
<td>69</td>
</tr>
<tr>
<td>D. A Tour of the Testbed File system</td>
<td>73</td>
</tr>
<tr>
<td>E. Programming Examples Using the Testbed Picture and Graphics Systems</td>
<td>79</td>
</tr>
</tbody>
</table>
Section 1

Introduction

The Image Understanding Testbed is a system of hardware and software that is designed to facilitate the integration, testing, and evaluation of implemented research concepts in machine vision. The system was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Defense Mapping Agency (DMA) and was developed at the Artificial Intelligence Center of SRI International.

This manual is designed both as an introduction to the Testbed for new users of the system and as a reference work for current users. Section 1 is an introduction to the background and general features of the system. Section 2 provides the basic information needed by a new user. Section 3 describes the Testbed UNIX\(^1\) programming environment, while Section 4 is concerned with the FRANZ LISP environment. Section 5 is devoted to the EMACS editor and several Testbed user systems that function in the context of EMACS. Section 6 gives an overview of the major applications programs running on the Testbed, while Section 7 describes a number of the available Testbed utility programs. Section 8 presents some brief guidelines for programmers who wish to develop programs to be integrated into the Testbed system.

Appendix A contains instructions for typical demonstrations of the programs contributed to the Testbed. Appendix B describes the Testbed picture file format, and Appendix C summarizes the graphics system. Appendix D gives an overview of the Testbed file system; Appendix E contains a number of C programming examples which demonstrate the use of the Testbed picture access and graphics libraries.

The remainder of this section will consist of overviews of the Testbed system, including:

* System documentation
* Hardware configuration
* Software configuration

1.1. Documentation

This document, "The DARPA/DMA Image Understanding Testbed USER'S MANUAL," presents a user's view of the Testbed. It outlines the general structure of the system and describes the use of major facilities. Complementary information is available in a UNIX-style document, the "IU Testbed Programmer's Manual," which collects the man pages describing Testbed programs, libraries, and files.

Some users may also wish to read the companion document, "The DARPA/DMA Image Understanding Testbed SYSTEM MANAGER'S MANUAL," which contains information relevant to system implementation and management issues. Users of systems that run the EUNICE/VMS emulation of the UNIX operating system may wish to examine the supplementary information contained in another management-oriented

\(^1\)UNIX is a trademark of Bell Laboratories.
Introduction

document, "Managing the IU Testbed under EUNICE/VMS."

The following reports evaluating major contributed software systems are currently available:

**GHOUGH**
The GHOUGH Generalized Hough Transform Package: Description And Evaluation

**PHOENIX**
The PHOENIX Image Segmentation Package: Description And Evaluation

**RELAX**
The RELAX Image Relaxation System: Description And Evaluation

The CMU graphics and picture access systems are described separately in the following documents:

CMU002, "Grinnell Display Software Support."
CMU003, "CMU Image Format and Paging System."
CMU004, "Image File Naming Conventions."

The EMACS editor system is described in another separate document, "UNIX EMACS," by James Gosling at CMU.

FRANZ LISP, the C language, and the features of the UNIX operating system and utilities are described in the "UNIX Programmer's Manual: Volumes 1, 2a, 2b, and 2c" available from the University of California at Berkeley Computing Services Library.

1.2. Hardware Configuration

The principal element of the Testbed hardware configuration is a DEC VAX-11/780 central processing unit and its peripherals. The SRI Testbed VAX is a four-megabyte system with one tape drive, four 300MB disk drives, one 414MB drive, and 32 teletype lines. The VAX interfaces to a variety of terminals and pointing devices via the teletype lines. Graphics capabilities are provided by a Grinnell display system. Other peripherals include a Versatec 11-inch printer/plotter with 200-point/inch resolution, which functions as a phototypesetter, and an Optronics C-4100 color image scanner with resolution selectable from 12.5 to 400 microns. The system also supports an ARPANET network link and an ETHERNET network connecting the VAX to a set of Lisp Machines. Other sites modeled after the SRI Testbed may have slightly different configurations.

The SRI Grinnell display system has a resolution of 512x512 with 32 bit-planes arranged in four 8-bit channels. Four overlay planes and four independent hardware cursors are available. Special features include individual zoom, pan, and color lookup tables on each group of 8 bit planes.

We note that it is desirable for terminals used with the SRI Testbed EMACS editor system to have a special meta key that toggles the eighth bit of the ASCII character set as well as a full set of screen control features.

---

*VAX is a trademark of the Digital Equipment Corporation.*
Introduction

Additional site-specific equipment that may be available is not described here.

---

**BLOCK DIAGRAM OF SRI IMAGE UNDERSTANDING TESTBED SYSTEM**

**VAX-11/780 COMPUTER**
- 4 MByte memory
- 32 teletype lines
- Four 300 MB removable disk drives
- One 414MB Winchester disk drive
- Tape drive

  | Grinnell display
  | 512 x 512 x 32 memory planes
  | Four overlays
  | Four independent cursors
  | Zoom and pan

  | Optronics image scanner
  | Transparent film resolution 12.5 to 400 microns
  | Opaque print resolution 100 microns
  | Variable optical density
  | RGB color filters

  | Versatec printer/plotter
  | 11-inch width
  | 200 points/inch resolution

  | Networks
  | ARPANET
  | ETHERNET to Lisp Machines

  | User devices
  | Text terminals
  | Mouse
  | Digitizing tablet
  | Digitizing table
Introduction

1.3. Operating System Software

The Image Understanding Testbed system may be run either under the UNIX\textsuperscript{3} operating system or under the VAX/VMS\textsuperscript{4} operating system. A "32V" or higher ("System III" or "System V") UNIX license is required to operate the Testbed under either system, while a EUNICE\textsuperscript{5} license is required in addition to run the Testbed under VAX/VMS.

Under the UNIX operating system, the Testbed currently uses the Berkeley VAX/UNIX 4.1c BSD system software distribution with support for the IP/TCP networking protocols. The 4.2 BSD system will be supported soon. Device drivers are available from Berkeley for the Versatec printer/plotter and for ARPANET devices; a driver for the Grinnell display system has been supplied by CXU. At present, no Optronics scanner driver is available under UNIX.

Under the VAX/VMS operating system, UNIX is emulated using the EUNICE system. This combination of operating-system support permits compatibility with both UNIX environments and other VMS/EUNICE environments. Device drivers are currently available under VMS for ARPANET devices, the Grinnell display system, the Versatec printer/plotter, and the Optronics image scanner. See the document "Managing the IU Testbed under EUNICE/VMS" for further details about running the Testbed under EUNICE.

1.4. Languages

The principal high-level programming languages of the Testbed are UNIX C and FRANZ LISP. The MAINSAIL algol-like language is available under both UNIX and VMS but is currently used only on the SRI EUNICE/VMS Testbed system. FORTRAN and PASCAL language compilers are available under both UNIX and VMS but are not used in any contributed software. On EUNICE/VMS systems, the DEC C language compiler can in principle be used instead of the UNIX C compiler for some applications; although the DEC C compiler generates exceptionally efficient code, substantial changes may be required to compile and run code written originally for a UNIX C system.

The dominant applications language on the Testbed is UNIX C. Extensive operating system facilities such as the make system, ctags, vgrind, and special features of the EMACS and vi editors are all designed to provide special assistance to the C programmer. We recommend that users be familiar with programming in C.

\textsuperscript{3}UNIX is a trademark of Bell Laboratories.
\textsuperscript{4}VAX/VMS is a trademark of the Digital Equipment Corporation.
\textsuperscript{5}EUNICE is a proprietary software product of SRI International.
Section 2

Getting Started on the Testbed

A new user on the Image Understanding Testbed system will need to know a few elementary facts in order to begin working with the system. Here we list the basic steps that need to be carried out, files that need to be created, and so forth. The detailed information required to interact with and understand the system is provided in later sections.

**IN THE BEGINNING.** A new user must first have the system manager carry out the following steps:

- Assign a user ID number, group, and home directory name to the new user;
- Create a home directory;
- Enter the user name and password into the account authorization system.

Next, the user should log onto the system and, with the system manager’s assistance, either create the following files, or copy the sample files from the /nu/testbed home directory:

**.login**
This file is a shell script that is run at user login time to set up a UNIX environment.

**.cshrc**
This sets up aliases and path names that the user wishes to be present in every C-shell environment that he may create. At least a search path declaration is customarily provided in this file.

**.bispro**
This file is loaded whenever a raw FRANZ LISP system is started up.

**emacspinit [EUNICE/VMS]; .emacs.pro [UNIX]**
The user’s EMACS editor environment may be customized by declaring personal key-bindings and function definitions in this file.

**login.com [EUNICE/VMS only]**
Systems running under VMS execute the login.com file at VMS login time; users may use this file to customize their VMS environment and to set some features of the EUNICE/VMS UNIX emulation mode. Invocation of the VMS command shell starts up the C-shell, runs the .login file, and places the user in an emulated UNIX environment.

**PICTURE DATA.** A representative library of picture data is stored in the /nu/tb/pic directory. Stereo and multiple images, color images, and monochrome images are available. All these images are stored in the standard Testbed picture format (see Appendix B) and are accessed via the picture access functions described in the pictioth man page. Several simple commands are available at the C-shell level to aid in examining the picture data files:

**show**
(takes the names of 1, 2, or 3 picture files and displays them on the default
Getting Started on the Testbed

display device (normally the Grinnell) as monochrome, anaglyphic stereo, or red-green-blue color images, respectively.

describe
reads the header in a named picture file and prints out the contents on the user terminal.
erase
erases the default display.
overlay
displays the named file as a 1-bit overlay on the default display (note: this works only with Grinnell device drivers designed to support this capability).

DEMONSTRATIONS. Appendix A contains instructions for running a number of simple demonstrations of Testbed programs. Representative demonstrations can be run by looking in the /pub/testbed/demo directory, connecting to a subdirectory, and invoking the demo shell file in that subdirectory. Brief program descriptions are given in Section 5 of this manual and in the man page collection. The "TU Testbed Programmer's Manual." More details about demonstrating and testing selected Testbed programs are available in the software evaluation reports.

PROGRAMMING. To use the Testbed as a programming environment, the user should become familiar with the UNIX system environment, the C language, an editor such as EMACS or vi, and possibly other language systems such as FRANZ LISP or MAINSAI. A user without prior preferences is strongly advised to begin by studying the EMACS system because of its exceptional versatility. Then one can become familiar with the techniques of C programming from the UNIX C-shell environment; the C programming tools of the EMACS environment may also prove helpful.

TESTBED LIBRARIES. Testbed library routines are organized into a hierarchy of three libraries contained in /pub/tb/lib. The libraries are linked in the following order if all are required:

visionlib.a imagelib.a sublib.a

The first two of these libraries have parallel versions that support the linking of native programs from CMU. (Note: convrtcmu.sh is a shell file illustrating the translation between Testbed names for certain utilities and the old CMU names.)

visionlib.a [CMU - cmuvsmlib.a]
This is the highest level library, and must have imagelib.a and sublib.a linked after it. High-level machine vision functions, histogramming, and so forth are found here.

imagelib.a [CMU - cmuimglib.a]
This is the picture access and graphics drawing library. All routines having to do with manipulating pixels in image arrays and drawing graphics on the display are found here. sublib.a must be linked in after this library.

Picture access functions: Access to digitized imagery is provided by the Testbed picture access functions documented in pictib and picti2b. A higher-level package supporting multiband image access is documented in the imgtib man page.

Graphics access functions: Graphics functions are currently accessible through a series of functions based on the CMU Grinnell graphics package. A quick summary of the functions is provided in Appendix C. Full
Getting Started on the Testbed

documentation is available in separate manuals from CMU. Higher level sys-
tems for carrying out image display are described in the man pages *frmlib*
and *imgfrmlib*.

**sublib.a**
This is the lowest level set of libraries, and should be linked last. Some appli-
cations will require only this library. Among the subsystems supported are:

**arglib:** These routines aid the user in parsing text strings that are either
passed as arguments invoked from the C-shell or requested interactively.

**printerr:** This is a system supporting hierarchical error-catchxing and report-
ing.

**ci command interpreter:** The *ci* command interpreter is a CMU-contributed
utility that enables user subroutines to be invoked interactively. The user
employs a library of argument-parsing utilities to provide defaults and valid-
dity checks before the command-line arguments are sent to the subroutines
themselves.

**icp command interpreter:** The *icp* command interpreter is an SRI-
contributed utility that supports more general treatment of arguments and
local variables than *ci*. In particular, *icp* invokes system and user subrou-
tines directly, while *ci* must have an argument-parsing interface written
especially for each routine.

The above information should be sufficient to get a beginning user acquainted with the
Testbed system. The following sections and Appendices contain additional details
about the programming environment, program systems, library routines, organization,
and use of the Testbed system. For UNIX man-style documentation, see the accom-
panying "IU Testbed Programmer's Manual."
Getting Started on the Testbed
Section 3

UNIX and the C Programming Environment

The Berkeley 4.1 BSD VAX/UNIX system is the core of the Testbed programming environment. An upgrade to 4.2 BSD is expected soon. The Berkeley C-shell provides a powerful command interpreter for invoking main programs and shell scripts. The standard UNIX system, programs, subroutines, and general features are described in the set of manuals entitled "UNIX Programmer's Manual: Volumes 1, 2a, 2b, and 2c" available from the University of California at Berkeley (UCB) Computing Services Library.

3.1. Introduction to UNIX

UNIX novices are advised to read several chapters of Volume 2, in the following order:

Chapter 3 - UNIX for Beginners
Chapter 2 - The UNIX Time-Sharing System
Chapter 1 - 7th Edition UNIX — Summary
Chapter 40 - An Introduction to the C-shell.

Among the most frequently used UNIX commands are:

- `pwd` - show current working directory
- `ls` - list the files contained in the current directory
- `cd` - connect to new directory
- `cat` - type out a file
- `more` - type out files by single screenfuls
- `cp` - copy a file
- `mv` - move a file to another name or place
- `rm` - delete (remove) a file
- `dirs, pushd, popd` - display, push or pop the directory stack
- `find` - general utility for locating files with given names or types
- `alias` - generate an abbreviation for a command
- `which` - tell which program in the search path will be run.

On-line documentation of the UNIX system is supported by a number of commands. On the Testbed, some of the informational programs have been extended to handle local Testbed information and to search local Testbed directories before the system-wide directories. The Testbed directory `/tu/tb/bin` should be placed early in the user's search path if these enhanced programs are to be invoked. The Testbed includes special versions of:

- `apropos` - lists routines whose synopses include a given key word
- `man` - runs a manual page through `nroff` for display on the terminal.
- `whatis` - lists the synopsis of the indicated routine
- `whereis` - shows the locations of all files pertinent to a command

To get a hard copy of a given manual section, one connects to the appropriate `man` directory, e.g. `/usr/man/man3` or `/tu/tb/man/man3`, finds the full name of the
UNIX and the C Programming Environment

troff source file for the desired man entry, e.g. string.3, and invokes a command like
	vtroff -man string.3

C programmers should be familiar with a number of utilities that are available on the
system to aid in program maintenance. Among these are:

ctags
The ctags facility enables the user to compile a tag table telling the loca-
tions of C subroutines within a collection of files. When used in conjunction
with the tag-table-interpreting capabilities of the vi or EMACS editors, the
tag table enables the user to edit routines by name without specifying
explicit file names or locations within files.

vgrind
The vgrind facility takes a standard C program and formats it for a photo-
typesetter. Pages are labeled and titled clearly, certain reserved words of
the C language are printed in boldface, and the entire text is indented to
emphasize program structure. (To keep the program from mistaking int
int baz() for a subroutine whose name is int, one must place the subroutine
type declarations on a separate line from the name.)

make
The make facility is a powerful mechanism for compiling and updating
complex systems. C programmers should arrange systems of programs so
that updates may be carried out by single make commands utilizing
makefile files in appropriate directories. The use of make is fairly complex
and is fully documented in the standard UNIX manuals; we refer the
reader to the detailed description in volume 2A of the UNIX Programmer's
Manual. On the Testbed, both /at/u/source and /at/u/lib have master
makefiles that will compile all the subdirectories. Each subdirectory has
its own makefile that will compile all the local routines and install them in
the appropriate place. To install only a single new file, the command make
FILES=filename.c may be invoked. [Users should beware that make may
fork a large number of subprocesses and will fail to finish normally if for
some reason the user's allocated number of processes is exceeded. On
EUNICE/VMS systems, the number of allowed processes can be altered
dynamically, while on UNIX systems the system parameters must be
reconfigured. One can circumvent the problem by remarking the elements
of the deepest hierarchies individually instead of using the master make
file.]

On the Testbed, there are various other utilities which may be useful:

crr
This generates a cross-reference listing for a C program.

indent
This is a utility to format C programs so that they are nicely indented
when printed out in ASCII. If ASCII printout is not required, the vgrind util-
ity is probably a better choice.

Users should be aware of the following features of the UNIX system relating to
input/output:

stdin, stdout, stderr
User processes under UNIX normally have three open I/O streams for stan-
dard input, output, and error logging. The integer file descriptors for
these streams ordinarily have the values 0, 1, and 2, respectively. This fact
is useful for developing programs which receive input piped from
UNIX and the C Programming Environment

Preceding programs. For more information, see stdio.h and /usr/man/man3/stdio.3s.

Limit on open files
UNIX normally limits the user to at most 20 simultaneously open I/O streams, including the three standard streams. [Under EUNICE/VMS, this limit is raised to a maximum of 32 streams, but may be limited to a smaller number by the user's VMS account privileges.]

3.2. Where to find UNIX system and Testbed files

The basic files making up the UNIX system include the following (see /usr/man/man7/lib.7):

/usr/src/cmd
Source files for all UNIX commands. These commands should for the most part be documented in /usr/man/man1.

/usr/src/libc
Source files for the standard C library in /lib/libc.a.

/usr/src/lib
Source files for all other libraries in /lib.

/bin
Some of the most frequently used executable programs.

/usr/bin
Other important executable programs.

/usr/ucb
Executable programs identifiable as developed at UCB.

/lib
Standard object module libraries and some C utilities.

/usr/lib
Other object module libraries, utilities, typesetter data and fonts.

/usr/include
Standard #include files for C programs.

/usr/man
The subdirectories manN contain the Nth chapters of the "UNIX Programmer's Manual" pages.

/usr/doc
Text formatted manuals for UNIX systems, e.g., the C manual.

/etc
Essential system data and maintenance utilities.

/tmp
Temporary files such as those generated by compilers.

/usr/tmp
More temporary files.
UNIX and the C Programming Environment

The basic files making up the UNIX-based TestBed system are given below. A more finely detailed list is provided in Appendix D, "A Tour of the TestBed File System," which is based on the file /tt/td/maa/man17/tbhead.7.

/tt/td/src
Source files for all TestBed main programs. These programs are documented in /tt/td/maa/man1.

/tt/td/lib
Source files and archives for all TestBed library routines. The major archive libraries are visionlib.a, imagelib.a, and sublib.a. Linking these three libraries in the order given provides access to all TestBed utility routines. Testbed library routines are documented in /tt/td/maa/man3 under the names of the sublibraries containing them.

/tt/td/bin
Executable files for all TestBed main programs.

/tt/td/td/tnclude
#include files for TestBed programs.

/tt/td/maa
The subdirectories manN contain the Nth chapters of the "Testbed Programmer's Manual" pages.

/tt/td/doc
Sources for formatted TestBed text files.

/tt/td/docsrc
troff sources for TestBed documents such as this one.

3.3. SYNOPSIS OF UNIX COMMANDS

In the following set of tables, we give an overview of the available UNIX and C-shell commands broken into operational categories. For complete documentation, the user should of course refer to the UNIX manuals.

UNIX and csh Command Language Functions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>Create or print synonyms.</td>
</tr>
<tr>
<td>at</td>
<td>Execute commands at a later time.</td>
</tr>
<tr>
<td>basename</td>
<td>Strip filename affixes.</td>
</tr>
<tr>
<td>cd</td>
<td>Change working directory.</td>
</tr>
<tr>
<td>chsh</td>
<td>Change default login shell.</td>
</tr>
<tr>
<td>csh</td>
<td>A shell (command interpreter) with C-like syntax.</td>
</tr>
<tr>
<td>echo</td>
<td>Echo arguments.</td>
</tr>
<tr>
<td>eval</td>
<td>Execute argument as a command.</td>
</tr>
<tr>
<td>exec</td>
<td>Substitute specified process for current one.</td>
</tr>
<tr>
<td>expr</td>
<td>Evaluate arguments as an expression.</td>
</tr>
<tr>
<td>false</td>
<td>Provide an unsuccessful execution status code.</td>
</tr>
<tr>
<td>foreach</td>
<td>Iterate over the elements of a list.</td>
</tr>
<tr>
<td>gets</td>
<td>Get a string from standard input.</td>
</tr>
<tr>
<td>glob</td>
<td>Expand file names.</td>
</tr>
</tbody>
</table>
UNIX and the C Programming Environment

goto Branch to a label.
hashstat Print effectiveness of file name hashing.
history Display the session history.
if Execute one of two statements.
kill Terminate a process with extreme prejudice.
lastcomm Show last commands executed in reverse order.
login Sign on.
nice Run a command at low priority.
nokup Ignore hangup and terminate signals.
notify Specify the label for an interrupt goto.
passwd Change login password.
pwd Working directory name.
rehash Recreate the file name hash table.
renice Alter priority of running process by changing nice.
repeat Execute a specified number of times.
set Set the value of a shell variable.
setenv Set shell environment variables.
source Execute commands from another source.
su Substitute user id temporarily.
switch Branch to one of several labels.
tee Pipe fitting.
test Condition command.
time Time a command.
true Provide a successful execution status code.
umask Set the default file access permission mask.
unalias Delete an alias.
unhash Discontinue hash table use.
unset Delete a shell variable.
wait Await completion of process.
while Iterate while a condition holds.
whoami Print effective current user id.
yes Repeatedly print "y" or an argument until terminated.

Additional sh Shell Command Functions

case Execute one of several statements.
export Make a shell variable global.
for Iterate over the elements of a list.
read Read a line of input.
readonly Mark a shell variable as read-only.
sh Shell command language.
shift Delete the first element of an argument list.
times Print time consumed by the current shell.
trap Specify an interrupt handler.

Information Functions

apropos Locate commands by keyword lookup.
find Find files.
learn Computer aided instruction about UNIX.
look Find lines in a sorted list.
UNIX and the C Programming Environment

**man** Run off section of UNIX manual.

**units** Conversion program.

**whatis** Describe what a command is.

**whereis** Locate source, binary, and or manual for program.

**which** Locate a program file including aliases and paths.

**Editors**

**ed** Text editor.

**edit** Text editor (variant of ex for casual users).

**ex** Text editor.

**sed** Stream editor.

**vi** Screen-oriented (visual) display editor based on ex.

**Text Formatting**

**checkeq** Check equations for unbalanced delimiters.

**checknr** Check nroff/troff files for common errors.

**col** Filter reverse line feeds.

**colcrt** Filter nroff output for CRT previewing.

**colrm** Remove columns from a file.

**deroff** Remove nroff, troff, tbl, and eqn constructs.

**egrep** Search a file for a regular expression.

**eqn** Typeset mathematics.

**expand** Expand tabs to spaces.

**fgrep** Search a file for a string.

**fmt** Simple text formatter.

**fold** Fold long lines for finite width output device.

**grep** Search a file for a pattern.

**lookbib** Look up a bibliographic reference.

**neqn** Mathematics preprocessor for terminals.

**nroff** Text formatting for terminals.

**ptd** Phototypesetter interpreter.

**refer** Find and insert literature references in documents.

**rev** Reverse lines of a file.

**soelim** Eliminate .so's from nroff input.

**spell** Find spelling errors.

**spellin** Add words to a spelling list.

**spellout** See if words are in the spelling list.

**ssp** Limit to single spacing.

**tbl** Format tables for nroff or troff.

**tc** Phototypesetter simulator.

**tk** Paginator for the Tektronix 4014.

**troff** Text formatting and typesetting.

**ul** Do underlining.

**vfontinfo** Inspect and print information about UNIX fonts.

**vgrind** Grind nice listings of C programs for the versatec.

**vtroff** Troff to the Versatec.

**Text Manipulation**
UNIX and the C Programming Environment

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp</td>
<td>Compare two files.</td>
</tr>
<tr>
<td>comm</td>
<td>Select or reject lines common to two sorted files.</td>
</tr>
<tr>
<td>ctags</td>
<td>Maintain a tags file for a C program.</td>
</tr>
<tr>
<td>diction</td>
<td>Find and print wordy sentences.</td>
</tr>
<tr>
<td>diff</td>
<td>Differential file comparator.</td>
</tr>
<tr>
<td>diff3</td>
<td>3-way differential file comparison.</td>
</tr>
<tr>
<td>explain</td>
<td>Interactive thesaurus for diction phrases.</td>
</tr>
<tr>
<td>file</td>
<td>Determine file type.</td>
</tr>
<tr>
<td>join</td>
<td>Merge two files line by line.</td>
</tr>
<tr>
<td>num</td>
<td>Number lines.</td>
</tr>
<tr>
<td>od</td>
<td>Octal or formatted dump.</td>
</tr>
<tr>
<td>ptx</td>
<td>Permutated indexed.</td>
</tr>
<tr>
<td>sort</td>
<td>Sort or merge files.</td>
</tr>
<tr>
<td>style</td>
<td>Analyzes surface characteristics of document style.</td>
</tr>
<tr>
<td>tr</td>
<td>Translate characters.</td>
</tr>
<tr>
<td>uniq</td>
<td>Report repeated lines in a file.</td>
</tr>
<tr>
<td>wc</td>
<td>Word count.</td>
</tr>
<tr>
<td>what</td>
<td>Show what object module versions were used in a file.</td>
</tr>
</tbody>
</table>

Numeric Manipulation

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apl</td>
<td>An apl interpreter.</td>
</tr>
<tr>
<td>awk</td>
<td>Pattern scanning and processing language.</td>
</tr>
<tr>
<td>bc</td>
<td>Arbitrary-precision arithmetic language.</td>
</tr>
<tr>
<td>dc</td>
<td>Desk calculator.</td>
</tr>
</tbody>
</table>

Language Aids

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adb</td>
<td>Debugger.</td>
</tr>
<tr>
<td>as</td>
<td>Assembler.</td>
</tr>
<tr>
<td>cb</td>
<td>C program beautifier.</td>
</tr>
<tr>
<td>cc</td>
<td>C compiler.</td>
</tr>
<tr>
<td>eff</td>
<td>Extended Fortran language.</td>
</tr>
<tr>
<td>error</td>
<td>Analyze and disperse compiler error messages.</td>
</tr>
<tr>
<td>eyacc</td>
<td>Modified yacc with improved error recovery.</td>
</tr>
<tr>
<td>f77</td>
<td>Fortran 77 compiler.</td>
</tr>
<tr>
<td>ld</td>
<td>Link editor.</td>
</tr>
<tr>
<td>lex</td>
<td>Generator of lexical analysis programs.</td>
</tr>
<tr>
<td>lint</td>
<td>A C program verifier.</td>
</tr>
<tr>
<td>lisp</td>
<td>Lisp interpreter.</td>
</tr>
<tr>
<td>liszt</td>
<td>Compile a Franz Lisp program.</td>
</tr>
<tr>
<td>lxref</td>
<td>LISP cross reference program.</td>
</tr>
<tr>
<td>m4</td>
<td>Macro processor.</td>
</tr>
<tr>
<td>make</td>
<td>Maintain program groups.</td>
</tr>
<tr>
<td>mkstr</td>
<td>Create an error message file by massaging C source.</td>
</tr>
<tr>
<td>pc</td>
<td>Pascal compiler.</td>
</tr>
<tr>
<td>pi</td>
<td>Pascal interpreter code translator.</td>
</tr>
<tr>
<td>pix</td>
<td>Pascal interpreter and executor.</td>
</tr>
<tr>
<td>prof</td>
<td>Display profile data.</td>
</tr>
<tr>
<td>px</td>
<td>Pascal interpreter.</td>
</tr>
<tr>
<td>pxp</td>
<td>Pascal execution profiler.</td>
</tr>
</tbody>
</table>
UNIX and the C Programming Environment

pxref  Pascal cross-reference program.
ratfor  Rational Fortran dialect.
sdb    Symbolic debugger.
strings Find the printable strings in a binary file.
struct Structure Fortran programs.
xstr   Extract shared strings from C programs.
yacc   Yet another compiler-compiler.

Graphics

cifplot CIF graphic format interpreter and plotter.
graph  Draw a graph.
plot   Graphics filters.
spline Interpolate smooth curve.

Programmer's Aides

cal     Print calendar.
calendar Reminder service.
leave   Remind you when you have to leave.
script  Make typescript of terminal session.

File and Directory Maintenance

cat     Catenate and print.
cat    Cat compressed files.
chmod   Change mode.
chown   Change owner.
compact Compress files.
cp      Copy.
crypt   Encode/decode.
dd      Convert and copy a file.
head    Give first few lines.
ln      Make links.
ls      List contents of directory.
mkdir   Make a directory.
mv      Move or rename files.
pr      Print file.
print   Pr to the line printer.
rm      Remove (unlink) files.
rmkdir  Remove an empty directory.
see     See what a file has in it.
split   Split a file into pieces.
sum     Sum and count blocks in a file.
tail    Deliver the last part of a file.
uncompress Uncompress files.
vpr     Versatec printer/plotter spooler.
vprint  Pr to the Versatec.
UNIX and the C Programming Environment

Archive and Object Library Maintenance

ar    Archive and library maintainer.
arcv  Convert archives to new format.
arf   Archiver for floppy.
lorder Find ordering relation for an object library.
rm    Print name list.
rnlib  Randomize an archive for rapid loading.
size  Size of an object file.
strip Remove symbols and relocation bits.
symorder Rearrange name list.
tar   Tape archiver.
tp    Manipulate tape archive.
tsrt  Topological sort.

Device Manipulation

clear Clear terminal screen.
fcopy Copy floppy disk.
getty Set terminal mode.
lock Reserve a terminal.
mmore Type files a page at a time on terminal screen.
reset Reset the teletype bits to a sensible state.
rewind Rewind tape drive.
stty Set terminal options.
tabs Set terminal tabs.
tset Set terminal modes.

Networking and Communication

biff Sets up notification of arriving mail.
binmail Alternate mail system.
call Ring a telephone.
cu Call UNIX.
delivermail Basic mail transmissions program.
enroll Initiate secret mail service.
from Who is my mail from?.
mail Send and receive mail.
makekey Generate an encryption key.
mesg Permit or deny messages.
msgs System messages and junk mail program.
net Execute a command on a remote machine.
netcp Remote copy of files through the net.
netlog Print the last few lines of the network log file.
netlogin Provide login name and password for a remote machine.
netpr Use a remote lineprinter through the net.
netmail Read mail on a remote machine over the network.
netq Print contents of network queue.
etrm Remove a command from the network queue.
etrmtroff Troff to the phototypesetter over the network.
newalias Rebuild the mail alias database.
UNIX and the C Programming Environment

prmail Print out mail in the post office.
uuclean Uucp spool directory clean-up.
uucp UNIX to UNIX copy.
uudiff Directory comparison between machines.
uuencode Encode a binary file for transmission by mail.
uudecode Decode a binary file for transmission by mail.
uulog Maintain a log of all network transactions.
uusend Send a file to a remote host.
uux UNIX to UNIX command execution.
wali Write to all users.
write Write to another user.
xget Read secret mail.
xsend Send secret mail.

System Maintenance

ac Login accounting.
accton Store accounting information for all processes.
analyze Virtual UNIX postmortem crash analyzer.
catman Create manual pages and update apropos database.
crond Clock daemon.
chfn Change full name of user.
chgrp Change group.
cli Clear i-node.
date Print and set the date.
dcheck File system directory consistency check.
ddate Print dump dates from the dump tables.
df Disk free.
dmesg Collect system diagnostic messages to form error log.
dn Summarize disk usage.
dump Incremental file system dump.
dumpdir Print the names of files on a dump tape.
finger User information look-up program.
fscik File system consistency check and interactive repair.
halt Halt the processor.
ichck File system storage consistency check.
init Process control initialization.
iosstat Report I/O statistics.
last Indicate last logins of users and telnetypes.
lpq Prints the line printer queue.
lpri Line printer spooler.
lpri Remove a file from the line printer queue.
mkfs Construct a file system.
mklost+found Make a lost+found directory for fsck.
mknod Build special file.
mount Mount a file system.
nccheck Generate names from i-numbers.
newgrp Log in to a new group.
printenv Print out the environment.
ps Process status.
pstat Print system facts.
quot Summarize file system ownership.
rc Command script for automatic reboot.
### UNIX and the C Programming Environment

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reboot</td>
<td>UNIX bootstrapping procedure.</td>
</tr>
<tr>
<td>restor</td>
<td>Incremental file system restore.</td>
</tr>
<tr>
<td>sa</td>
<td>System accounting.</td>
</tr>
<tr>
<td>sleep</td>
<td>Suspend execution for an interval.</td>
</tr>
<tr>
<td>swapon</td>
<td>Specify an additional device for paging and swapping.</td>
</tr>
<tr>
<td>sync</td>
<td>Update the super block.</td>
</tr>
<tr>
<td>touch</td>
<td>Update date last modified of a file.</td>
</tr>
<tr>
<td>trman</td>
<td>Translate version 6 manual macros to version 7 macros.</td>
</tr>
<tr>
<td>tty</td>
<td>Get terminal name.</td>
</tr>
<tr>
<td>umount</td>
<td>Dismount a file system.</td>
</tr>
<tr>
<td>update</td>
<td>Periodically update the super block.</td>
</tr>
<tr>
<td>uptime</td>
<td>Show how long system has been up.</td>
</tr>
<tr>
<td>users</td>
<td>Compact list of users who are on the system.</td>
</tr>
<tr>
<td>vipw</td>
<td>Edit the password file with vi.</td>
</tr>
<tr>
<td>vmstat</td>
<td>Report virtual memory statistics.</td>
</tr>
<tr>
<td>vpac</td>
<td>Print Versatec accounting information.</td>
</tr>
<tr>
<td>vpq</td>
<td>Prints the line printer queue.</td>
</tr>
<tr>
<td>vprm</td>
<td>Remove a file from the Versatec queue.</td>
</tr>
<tr>
<td>who</td>
<td>Who is on and what they are doing.</td>
</tr>
<tr>
<td>who</td>
<td>Who is on the system.</td>
</tr>
</tbody>
</table>
UNIX and the C Programming Environment
Section 4

The Testbed FRANZ LISP Environment

FRANZ LISP is a dialect of MACLISP that has been implemented in C and is designed to interact naturally with the Berkeley UNIX operating system. Complete documentation of the FRANZ LISP system is available in "The FRANZ LISP Manual," a University of California at Berkeley document by J.K. Foderaro and K.L. Sklvester. (A copy of this manual is included on Volume 2c of the "UNIX Programmer's Manual.")

In this Section, we describe a selection of utilities and program systems that have been developed for use on the Testbed and that can be loaded into a FRANZ LISP image to construct a customized Testbed LISP user environment.

4.1. STARTING AND STOPPING FRANZ LISP

The standard FRANZ LISP 38.79 program system is /usr/uucb/lisp, and is invoked by the command

```
  lisp
```

provided that /usr/uucb is in your search path. Upon startup, lisp will search for the file .lisprc in the user's home directory and will load it if present. Either typing -Z or invoking the function (exit) will cause an exit from FRANZ LISP.

The user may at any time suspend a currently executing FRANZ LISP function, examine its state, and resume execution with the following sequence of actions:

```
  ~C
```

; control-C takes you out of LISP to a "break" loop.

```
<1>: < do anything you wish >
<1>: (return) ; this resumes execution of the original function.
```

The Testbed FRANZ LISP in /iu/tb/bin/tblishp has a number of special features and utilities preloaded which may be of interest to some users. The Testbed image is invoked by

```
  tblisp
```

provided that /iu/tb/bin is in your search path. The -Z exit method is disabled to prevent accidental destruction of one's LISP. To exit from tblisp, enter the command (exit) or, for convenience on SRI Datamedia keyboards, (~).

The raw FRANZ LISP upon which tblisp is built is not /usr/uucb/lisp, but /iu/tb/bin/patched-lisp, which has a special patch to enable control-E interrupts for LEDIT (see below).

4.2. UNIX FACILITIES

A C-shell may be invoked from FRANZ LISP as a subprocess using the (shell)
The Testbed FRANZ LISP Environment

command. Access to individual system commands and programs is provided by
the (exec), (fork), (syscall), and (*process) commands.

When using such systems as (*process), the user is warned that a (wait) com-
mmand must be invoked to avoid hung processes, and that any named pipes
opened as side-effects must be released using (close pipe-name) to avoid using
up the fixed number of 20 I/O ports that LISP can support under UNIX [under
EUNICE/VMS, the number may be increased to 32].

Subroutines written in UNIX C, PASCAL, and FORTRAN may be linked directly into
FRANZ LISP using (cfa1). Note that subroutine entry points must be preceded
by one underscore, e.g. (cfa1 '_foo 'lisp-name 'e) will link in a C subroutine
named foo() in its C source file. Users should also beware that FRANZ LISP itself
is written in C, and that many subroutine names internal to FRANZ are present
in the symbol table: if one accidently loads a user C routine with a conflicting
name, the program may fail.

There are a number of internal LISP functions which are in fact useful for com-
municating to user C programs. Here we mention only one, inewstr(), which
passes a legal LISP string object from C back to LISP. It is used as follows:

```
char *ttstring()
{
    extern char *inewstr(); /* This converts a string to lisp space */
    return(inewstr(ttyname(1)));
}
```

When the function _ttstring is then (cfa1)'d into LISP, (ttstring) it will return
a LISP string as its value. Other examples may be found by reading Section 4.6
of the FRANZ LISP manual and examining the LISP source code.

**NOTE:** Under EUNICE/VMS, the liszt -f filename.1 example in Section 12.6 of the
FRANZ LISP manual will fail, since the resulting bootstrap file is meaningful only
to the UNIX kernel. To get the same effect, compile the desired file to give a "o"
file, and then invoke

```
/usr/ubc/lisp -f filename
```

This loads the function into LISP without having to store an entire LISP image.

4.3. Where to find LISP files

Fundamental files making up the FRANZ LISP system are the following:

```
/usr/src/cmd/lisp/
   This contains the FRANZ LISP source code. Subdirectories include
   franz/ - FRANZ LISP itself
   liszt/ - the FRANZ LISP LISZT compiler
   lispib/ - a selection of FRANZ LISP utilities such as step, etc.

/usr/lib/lisp/
   Object files for /usr/src/cmd/lisp/lispltb/ are kept here.

/usr/ib/lisp/
   Source files for systems of Testbed LISP applications programs. Sub-
   directories include:
   src/ - LISP source files.
   csrc/ - C programs needed for special applications
```

22
The Testbed FRANZ LISP Environment

*graphics* - graphics utilities accessible from LISP.
*help* - help files for Testbed demonstration menus.
*lib* - aliased to *lisplib* (see below).

*lisplib* (alias of *nu/tb/lisp/lib*)
This is a library of C and LISP utilities developed independently for the Testbed FRANZ LISP environment. Included are routines that enable single-character teletype I/O and support the EUNICE/VMS ledit system.

The Testbed program *tblisp* contains the basic Testbed LISP environment; source files, a *makefile* for compiling the system, and the file *setup.l*, which is used to construct the *tblisp* system from the raw *nu/tb/bin/patched-lisp*, are found in *nu/tb/lisp*.

4.4. The LEDIT System  [This section refers only to the LEDIT system that runs on EUNICE/VMS versions of the Testbed; the facilities described here cannot be supported on UNIX at this time. For additional background, see "Managing the IU Testbed under EUNICE/VMS." Similar UNIX facilities have been constructed by a number of organizations.]

LEDIT is an extremely useful mechanism for editing LISP functions in the EMACS editor and then loading the revised functions into a running LISP. To set up the LEDIT system, one must carry out the following steps:

* Start up EMACS from VMS
* Load the */emacslib/ledit.ml* macro package into EMACS
* Suspend the EMACS with *(paws-emacs)*
* Start up a LISP from VMS
* Load the *lisplib/ledit.l* package into LISP.

Many of these steps may be accomplished automatically by judicious use of one's *emacsrc* file and a LISP such as *tblisp* which already has the LEDIT facilities loaded and saved in the LISP image.

One must bear in mind that the LEDIT system consists of two distinct parts, the FRANZ LISP portion of the system and the EMACS portion of the system. Further details on the LISP portion of the system are contained in this section, while the reader is referred to the LEDIT portion of the EMACS chapter for additional details concerning the EMACS side of the system.

The following function loads into LISP the interface to the EMACS editor:

```
(cfasl '/lisplib/lispsemacs '.emacs 'edit')
```

The function *(edit)* will then activate a previously existing EMACS process; this editor must have been started up and suspended before you entered LISP.

One must also invoke

```
(load '/lisplib/ledit)
```

to define the *(ledit)* function. Assuming one already has a suspended LEDIT EMACS invoked from VMS and has a LISP running with LEDIT functions loaded, one uses the package as follows:

23
The Testbed FRANZ LISP Environment

(ledit) or "control-E" suspends LISP and starts up the suspended EMACS.
(ledit function-name pp) loads the function "function-name" into EMACS.

Functions and files can then be edited as usual in EMACS. When ready to resume
LISP, the user invokes one of the several available EMACS functions described in
the following chapter. One has the option of resuming LISP directly with (paws-
emacs), loading the current function using (zap-defun-to-lisp), or loading a
series of functions which have been marked using (ledit-save-defun) for reload-
ing during the editing session. If (zap-defun-to-lisp) is invoked with the mark at
the beginning of a file, all the functions in the entire file are loaded into LISP.

4.5. Testbed Demonstration Programs

A variety of programs are available and loaded into tblisp to help support user
programming and demonstrations. Among the files in /usr/lbl/lisp/src/ that are
included in tblisp and which may be of general interest are:

utilities.l
This is a utility system including the routines:
(print-list item-list) ; item-list is a list of arbitrary atoms
(print-tabbed-list item-list)
(print-numbered-list item-list)
(read-line open-file-port)
(get-single-char char-list prompt) ; only accepts characters in the list
(get-file-list directory-string)

top.l
This contains a customized Testbed top level system that is invoked in
tblisp by rebinding top-level to testbed-top-level upon entry. (Users
wishing to use this same mechanism in their own LISP image may wish
to save the function definition of top-level somewhere so that it can be
recovered when wanted.) The prompt and actions taken in the course
of evaluating an S-expression (e.g., echoing commands to a trail file)
can be further customized by modifying this basic file.

menu.l
This Testbed Menu system is a general recursive menu caller that is
invoked most simply by
(menu (make-menu-plist item1 item2 item3 ...))
Examine the result of (make-menu-plist) to see what how a menu is
composed. Among the menu utility routines available are:
(menu menu-plist)
(make-menu-plist arg-list)
(select-file directory)
(**push-menu-stack** name)
(**pop-menu-stack**)  (**clear-menu-stack**)
(**print-menu-stack**)  (**print-menu-stack**)

menudata.l
This contains a basic set of menus for the (menu) program, including
an empty menu that may be used as a template for user menus. The
empty menu can also be passed as an argument to menu, using (menu
empty-menu), to examine basic features of the system.
The Testbed FRANZ LISP Environment

picdbms.l
This is a preliminary version of a Testbed picture database support system. The functions available here are primitive but potentially useful. The list of currently supported keywords for imagery is contained in the special variable **key-list**. Among the keys that are supported are the words listed in parentheses below:

IMAGE TYPE AND MULTIPLICITY: (bw color stereo multiple)
SCENE DOMAIN TYPE: (indoor cultural natural)
CONTENT CHARACTERISTICS: (point linear area)
VIEWPOINT: (aerial ground).

Other types of data can be supported as the need arises.

The special data type picture-list is a list beginning with the directory name and followed by a list of images.

For use with the picdbms system, a picture directory must have a pic.dat file that has been formatted by hand in the picdbms format or has been created by the make-pic.dat utility. Among the functions available are the following:

(db_help) - types the help file in /atu/tb/lisp/help/picdbms.hlp
(key-loop picture-list) - loops through the elements in a picture-list
(get-key-list directory picture) - extracts all keywords from /pic.dat
(get-pic-key-list pic-list key-list) - returns a list of pictures having ANY key
(get-pic-allkey-list pic-list key-list) - returns those pictures having ALL keys
(pic-loop pic-list) - displays in sequence all pictures in pic-list

(make-pic.dat dir-name pic-name img-name data-file) - creates data file with the name passed in "data-file", e.g., pic.dat.
(all-pic.dat directory) - compiles pic.dat files for a whole directory

(show-any-pic pic-name-string) - displays the specified picture on the default display.

demodata.l
This is large menu of Testbed demonstrations. It includes the utility
(send-commands-on-cr cmd-list to-process-port),
which is used to allow pauses between command lines being piped to the demonstration process.

4.6. Environment Utilities

A number of useful special utilities are available in the Testbed LISP library. Most of the utilities listed here are loaded into tblisp, and can be loaded into any user system, by loading /atu/tb/lisp/csrc/setup.l.

The Utilities listed here include those used by the EUNICE/VMS Testbed system to supply capabilities unavailable under UNIX 4.1BSD. We list below a selection of these utilities and the manner in which they are customarily loaded into LISP.
The Testbed FRANZ LISP Environment

4.6.1. General Utilities

**chars-in-buffer-maknum**

The function `chars-in-buffer-port` defined below returns the number of characters left in a pipe or in a file text line. "-1" is returned at the end of a pipe or upon reading the `<if>` at the end of a file text line. To check for the end of a file, test for `(tyl) = -1. One cannot use this test to check for the end of a piped message. If the pipe is empty, (tyl) will wait forever; this is why one requires this special function to test for the end of a pipe message. The routines are loaded as follows:

```
(cfasl '/lisplib/rbfsizer.o 'linfmt 'chars-in-buffer-maknum 'f)
(defun chars-in-buffer (port)
  (chars-in-buffer-maknum (maknum port)))
```

**getwd, ttyname, cbreak, raw, echo**

These are system utility functions that interrogate UNIX environment variables, return LISP-readable values, and enable special teletype I/O functions. Loading is as follows:

```
(chdir '/iu/tb/lisp/csrc)
(cfasl 'ttylisp.o 'ttyname "ttyname" "function")
  ; TWO underscores precede name:
  ; (ttyname) returns a LISP string with the current teletype name.

(getaddress 'getwd 'getwd 'f)
  ; TWO underscores precede name:
  ; (getwd) returns a LISP string with the current directory name.

(getaddress 'cbreak 'char-mode-on 's)
(getaddress 'ucbreak 'char-mode-off 's)
  ; (char-mode-on) enables single-character teletype I/O;
  ; ctrl-C is still enabled for normal process interruption.

(getaddress 'raw 'raw-mode-on 's)
(getaddress 'unraw 'raw-mode-off 's)
  ; (raw-mode-on) enables raw teletype I/O; the user must
  ; handle ctrl-C, DEL, ctrl-Y, and so forth.

(getaddress 'echo 'echo-mode-on 's)
(getaddress 'noecho 'echo-mode-off 's)
  ; (echo-mode-on) and (echo-mode-off) turn character
  ; echoing at the terminal on and off.
```

**Picture display utilities**

The standard Testbed picture display system can be loaded to enable the display of picture files on the standard display as follows:

```
(chdir '/iu/tb/lisp/csrc)
(cfasl 'showpic.o
  'showpic
  'show-pic
  's
  "/iu/tb/lib/imagelib.a /iu/tb/lib/sublib.a-im")

(getaddress 'genshowpic 'gen-show-pic 's)
```
The Testbed FRANZ LISP Environment

; Requires x, y, xorigin, yorigin to be specified
(getaddress 'showcolorpic 'show-color-pic 's)
; Display color picture
(getaddress 'genshowcolorpic 'gen-show-color-pic 's)
; Allows x, y, xorigin, yorigin.

4.8.2. EUNICE/VMS Utilities
dummy-address, dummy-value

These are dummy variable manipulation routines loaded to avoid problems when using SYSCALL 8 to set up a VMS mailbox; LISP storage can otherwise be clobbered by a non-LISP value appearing in the wrong place.

(cfasl '/lisplib/dummy.o 'dummy_address 'dummy-address 'i)
; Define (dummy-address n), 0 <= n <= 9
(getaddress 'dummy_value 'dummy-value 'i)

Event flag handling

These routines allow users to set and use VMS Event Flags and are very useful as signals to control processes competing for resources. They use the VMS Event Flag system routines SY$AscEf, SY$SetEf, SY$CirEf, and SY$WaitFr. The returned values are:

WASCLR = 1 if the event flag is clear, EF = 0,
    indicating the Channel IN USE.
WASSET = 9 if the event flag is set, EF = 1,
    indicating the CHANNEL IS AVAILABLE. (The logic is that
    you wait until EF = 1 to wake up your process.)

The event flag cluster (32 flags) named "dpylock" is compiled into efn.s.c. It is assigned the number "64", but this number is not compiled in, and so must be supplied to the calling routines. To load the routines, use:

(cfasl '/lisplib/efns.o 'assoc_dpylock 'assoc_dpylock 'i)
; Allocate a flag code, e.g. (assoc_dpylock 64)

(getaddress 'set_efn 'set_efn 'i); Define set_efn
    ; (set_efn 64) sets the event flag to indicate
    ; the resource is free, returns the value = 9.

(getaddress 'clear_efn 'clear_efn 'i); Define clear_efn
    ; (clear_efn 64) returns a value = 1 if flag is clear.

(getaddress 'wait_for_efn 'wait_for_efn 'i); Define wait_for_efn
    ; (wait_for_efn 64) will wait on the event flag.

27
The Testbed FRANZ LISP Environment
Section 5

The Testbed EMACS Editor System

EMACS is an extensible editor that provides its own powerful local programming environment; several major Testbed utility systems are implemented within the EMACS environment itself. For editing text and programs, the user is free to use either EMACS or any other available text editor such as vi. However, since no other editor can support the utility systems available from EMACS, users are advised to become familiar with some of the basic EMACS features.

The EMACS program can be suspended and restarted without losing the files being edited; one simply invokes the pause-emacs command instead of the exit-emacs command when one wants to return temporarily to the system command level. Users typically bind pause-emacs to ctrl-X/ctrl-Z or ctrl-C; in order to avoid losing files in an EMACS which one did not intend to kill, many users do not bind exit-emacs to anything (one can always invoke it as an extended command).

The SRI Testbed EMACS system is tuned to work with customized Datamedia terminals with a special character keypad and a special meta key, which acts to toggle the eighth bit of the ASCII character set. The function of the meta key is implemented on other terminals by two-key sequences beginning with the <ESC> key.

5.1. Where to find EMACS files

The EMACS system readable copy of the EMACS manual is in
/usr/newemacs/man/emacs.doc. SCRIBE sources for the manual are in

The Testbed library of customized EMACS macros and utility systems is found in
/emacslib, which is currently an alias of /usr/lib/emacslib. The UNIX
environment variable EPATH is set to this same directory and is used by EMACS
to search for macros to be loaded if a full path is not given. If this path is
changed, the hard-wired path name in the macros
describeX.ml, desword.ml, makealiases.ml
must also be changed for the EMACS internal help database to function

correctly.

The INFO on-line documentation system described below uses the directory
/usr/info
as its default source of information files.
The Testbed EMACS Editor System

5.2. The EMACS Editor

Full descriptions of the features of the EMACS editor itself are available in a separate document, "UNIX EMACS," by James Gosling at Carnegie Mellon University. The Lisp-like macro language "MLISP" that EMACS supports allows the development of complex user systems within the EMACS environment. There are number of advantages to writing user systems within EMACS:

* Reprogrammability of the keyboard
* Multiple window support
* Scrolling and pageable window access to long text files
* Ability to extract parts of files for display
* Ability to restrict user access to a subwindow of a file
* Ease of providing HELP information in pop-up windows
* Ability to execute other program systems in an EMACS window context.

Providing any subset of these functions in separate user programs would be exceedingly complex; EMACS supports a uniform context in which sophisticated text-oriented procedures may be implemented with minimal effort.

5.3. EMACS Personalization

If the file $emacsinit$ (under VMS) or $emacs.pro$ (under UNIX) exists in the user's home directory, it is loaded into EMACS at program startup time. The initialization file customarily contains MLISP expressions that customize the editor environment by binding keys, defining functions, and loading library systems. A typical file will contain initialization commands written in the EMACS MLISP command language, such as:
The Testbed EMACS Editor System

;;; Emacs initialization file.
;; FILE = ~/emacsinit or ~/emacs.pro

(load "autoloads.ml"); basic required functions
(load "modeline.ml"
  ; sophisticated modeline with message notices

(autoload "visit-tag-table" "ltags.ml"
  ; Load the LTAGS functions when
  ; "visit-tag-table" is invoked.

(autoload "visit-tag-table" "ctags.ml"

(auto-execute "lisp-mode" ".l"
  ; Invoke "lisp-mode" whenever
  ; a file ending in ".l" is read into a buffer.

(auto-execute "c-mode" "*.c"
(auto-execute "mlisp-mode" "*.ml"
(auto-execute "text-mode" "*.txt"

: SET PERSONAL VARIABLE VALUES
(set "checkpoint-frequency" 0)
  ; Disable autosave
(set "default-case-fold-search" 1)
  ; so x = X on search
(set "case-fold-search" 1)
  ; necessary for buffer "main"
(set "ctlchar-with-" 1)
  ; show -A rather than 01
(set "track-col-on-" "N-"
  ; don't move to end of line
(set "help-on-command-completion-error" 0)
  ; when I need your help...

; THESE VARIABLES SHOULD BE SET AS FOLLOWS:
(set "wrap-long-lines" 0)
  ; truncate long lines
(set "quick-redisplay" 1)
  ; disable quick-redisplay.

The association between physical keys and functions is user controllable. Here we list the conventional key assignments for the most fundamental operations. The operations can be split into several categories:

Buffers
The principal EMACS text object is the buffer. A buffer contains text that has been loaded from a file or typed in by the user. EMACS may have many buffers active at one time. One or more of these buffers may be displayed in windows on the terminal screen.

- ctrl-X/ctrl-V prompts for a file name to be read into a new buffer.
- ctrl-X/ctrl-W prompts for a file name and writes a buffer to that file.
- ctrl-X/ctrl-S saves the current buffer in the file it was read from.
- ctrl-X/ctrl-R reads a file into an already-existing buffer.

Screen Windows
A screen window displays a portion of a text buffer.

- ctrl-V displays the next page of a buffer in the current screen window.
- meta-V displays the previous page of a buffer in the current window.

The Cursor, the Dot, and the Mark
The current position of attention in the displayed buffer window is
The Testbed EMACS Editor System

indicated by a blinking cursor. The LEFT edge of the cursor is the "dot." Text is inserted at the dot, and cursor-motion commands move the dot around the buffer. Text regions are delineated by marking the beginning of the region and moving the dot to the end of the region. To set the current mark to the current dot position, type

ctrl-@ or <null> or ctrl-space.

Text Insertion

EMACS, unlike many text editors, is usually in "self-insert" mode: typing any ordinary printable character causes that character to be inserted in the buffer at the dot, that is, IN FRONT OF the blinking cursor.

Moving the Cursor

The cursor can be moved in several modes:

In Character Mode,

ctrl-f = RIGHT (Forward character)
ctrl-b = LEFT (Backward character)
ctrl-d = Delete Following Character
<DEL> = Delete Previous Character.

In Word Mode,

meta-f = Forward Word
meta-b = Backward Word
meta-d = Delete Following Word
meta-DEL = Delete Previous Word.

In Line Mode,

ctrl-p = UP (Previous line)
ctrl-n = DOWN (Next line)
ctrl-a = Beginning of line
ctrl-e = End of line
ctrl-k = Delete rest of line.

In Region Mode,

ctrl-@ = Set Mark
ctrl-w = Delete region between mark and cursor
ctrl-y = Insert previously deleted region at cursor.

There are many more capabilities, some of which have standard key bindings and some of which do not. Named functions without key bindings are invoked by the Extended Command sequence,

meta-x command-name [optional arguments].

For additional details, see the UNIX EMACS manual, or go through the EMACS tutorial in the INFO system, described below.

5.4. C programming in EMACS

C programmers will find a number of utilities in EMACS that support C programming tasks and file editing. The most useful standard utilities include the following:

-c-mode
C-mode includes such features as meta-" ("back-quote") and meta-"
The Testbed EMACS Editor System

("quote") to bracket comments with /* and */, flashing of matching open parentheses when a closed parenthesis is typed, remembered indentation for rough pretty-printing, and meta-<linefeed> for pretty-printing a whole procedure. Typically, one inserts the line

\begin{verbatim}
(auto-execute "c-mode" "*-c")
\end{verbatim}

into one's "emacsinit" file in order to have c-mode in effect for any C program.

cTags: visit-ctag-table, goto-ctag, next-ctag
The UNIX ctags program assembles a table of subroutine entries together with their positions in their source files; this information is then used to locate individual routines within a file without involving the expense of global searches through a potentially large number of long files. If the set of routines of interest has had ctags run on it, these EMACS functions will place a named routine into the current EMACS buffer automatically. The user may bind keys to ctags functions by defining the function &ctags-load-hook to do the binding in emacsinit.

compil-it
This function, usually bound to ctrl-x/ctrl-e, runs the makefile in the current working EMACS directory. If there are any errors, they are listed in the Error buffer; invoking next-error (ctrl-x/ctrl-n) pops the next offending routine into a buffer with the cursor on the erroneous line. This enables one to edit, check, and compile C programs very effectively from within EMACS.

compil-it with a prefix
If the ctrl-x/ctrl-e invocation of "compile-it" is prefixed with ctrl-u, the user is prompted for a UNIX command line to execute. The output of the command is piped into the Error buffer and displayed. A particularly useful application is the invocation of grep -n, which places each line containing the specified text string into the Error buffer; next-error then pops the file containing the next found line into a buffer for editing.

execute-monitor-command
This executes a single C-shell command and puts the output into the "Command Execution" buffer.

describe-word-in-buffer
This command gives the syntax of the C system subroutine name pointed to by the cursor. It makes use of the EMACS database facility, and so the appropriate database files must exist for proper function.

5.5. INFO Utility

The INFO system is a self-contained MLISP subsystem of EMACS that permits interactive perusal of any on-line documentation system written in the INFO format.

The most complete documentation of INFO is actually contained in the INFO tree itself. Here we will give a rough outline of the main features.

INFO root directory
The INFO root directory, e.g. "/iu/info", contains a number of files of INFO nodes, including the "dir" file, which specifies the top level structure of the tree.

Tree structure
The root directory contains a menu of INFO nodes that may be selected.
The Testbed EMACS Editor System

Each subordinate node may contain pointers to the parent ("up") node, a logically preceding ("previous") node, and a logically following ("next") node, in addition to a menu of subordinate nodes. A node may also consist of a raw text file from some other directory. The basic syntax for entering a node is

```
-<L
File: foo, Node: Wish List, Up: Top, Next: Future Wishes, Previous: Old Wishes
< .... text .....>
```

All fields except "Node" are optional. Either commas or tabs may be used as field separators.

The menu format is as follows:

- menu:
  - Phones: (/usr/baz/phones)Top
    - An INFO node in a separate file.
  - Log: (/sysmgr/progress.doc)*
    - A text file with no INFO structure.
  - Wish List::
    - An INFO node whose text is in the current file.

Menus are invoked with the "m" command: any unambiguous string followed by a space or carriage return will cause the node named in the menu to be selected and displayed. To get back to the parent node after invoking a pure text file, use the "l" command.

Executable preambles

Any node in standard INFO format may be preceded by a user-invisible, executable MLISP s-expression. This allows the insertion of text into the following node that depends on the user, that reflects a current transient condition of the system, or that executes some user program to illustrate the information in the node.

Key bindings

INFO is a completely independent subsystem of EMACS that uses printing keys to execute distinct INFO commands instead of causing the insertion of text into a buffer. `ctrl-` the 'a' key on SRI-customized Datamedia terminals, is bound to the "help" function and causes a list of key bindings to be displayed. Typical commands are:

- `<space>` to see the next page of a multi-page node
- `<backspace>` to see the previous page
- "n" to see the next node
- "p" to see the previous node
- "u" to see the parent ("up") node
- "b" to go to the beginning of the current node
- "e" to go to the end of the current node
- "m" followed by an unambiguous substring to select a node from a menu
- "l" to return to the "last" node invoked prior to the current node
- "d" to go to the "dir" or root node of the INFO tree.

Editing INFO nodes

INFO nodes may be edited by users who have sufficient privilege to write in the INFO root directory. `meta-ctrl-E` is bound to the `info-edit` function and allows editing of the current node. `ctrl-` exits from the editing mode.
The Testbed EMACS Editor System

and causes the edited node to be saved if desired.

Personal INFO nodes

Users may construct their own INFO nodes or trees and switch among
different trees by setting global variables within a running EMACS. Typi-
cally, one would put a function in one's .emacsinit file containing:

```
(if (not (info-is-initialized))
   (load "info.ml")
   (setq info-default-directory "/usr/baz/info")
   (setq info-root-file "dir")
```

5.6. RMAIL Utility

RMAIL is a self-contained subsystem of EMACS that is written in MLISP and supports
local and ARPANET mail processing. meta-r is typically bound to the "rmail" com-
mand and invokes the mailer. Like INFO, RMAIL binds ordinary printing keys to vari-
ous program functions. ctrl-m the hold key on SRI-customized Datameida terminals,
is bound to the "help" function and gives a full list of key functions. Typical functions are

- <space> to see the next page of a message
- <backspace> to see the previous page
- "n" to see the next message
- "p" to see the previous message
- "r" to reply to a message
- "m" to compose and mail a new message
- "d" to delete a message
- "q" to quit and update the file.

NOTE: A new VAX version of the TOPS-20 "MM" mail utility is expected to replace
RMAIL in the near future.

5.7. FRANZ LISP Programming in EMACS

FRANZ LISP programming is supported in EMACS by a number of helpful utilities. The
LEDIT system, which runs only on VMS/UNICE systems, allows one to switch back
and forth between EMACS and a running FRANZ LISP; newly edited functions may be
loaded from EMACS directly into FRANZ LISP, while interpreted functions defined in
the running FRANZ LISP can be loaded into the running EMACS for easy editing.

The functions and utilities designed for use with LISP include the following:

- **lisp-mode**

  If the function "&atomic-lisp-hook" is defined by the user, it will be run to
  initialize "lisp-mode."

  In LISP mode, matching open parentheses are pointed to by the cursor as
each closed parenthesis is typed; matches falling outside the current
screen are displayed in the EMACS prompt buffer, and the user is notified
of mismatches. The variable "paren-flash-wait" determines how many
tenths of a second the parenthesis-matching process pauses.

- <linefeed> and <tab> are used to invoke automatic indentation to pretty-
  print s-expressions. Among the other useful functions are:
meta-<up> = Forward s-expression
meta-<down> = Backward s-expression
meta-<left> = Delete following s-expression to Kill Buffer
ctrl-<left> = Delete indentation (undoes effect of <linefeed>.

ltags: make-tag-table, visit-tag-table, goto-tag, next-tag
The LTAGS facility provides the LISP analog of the CTAGS facility. To use
LTAGS, the user must load or autoload "ltags.m1", and invoke "visit-tag-
table". The tag table is a file that may have any name the user wishes. To
make a LISP tag table, one first writes a file each line of which contains
"ctl-\" followed by a file name (usually a full path):
~/usr/baz/lisp/utilities.l
~/usr/baz/src/toplevel.l.
Then one invokes "make-tag-table" with the name of the empty tag table to
generate a modified file that has the names of all defuns and defmacros
inserted after the source file name. "goto-tag" and "next-tag" are then
used to pop the desired function into the editing window. The function
"&ltags-load-hook" may be defined in one's "emacsm1" to bind function
keys whenever LTAGS is in use.

ledit
The LEDIT system supports context-switching between a running FRANZ
LISP and a running EMACS. Any desired initialization, such as loading an
LTAGS file into EMACS, may be carried out by defining the function "ledit-
load-hook."

(LEDT presently is implementable only under VMS/EUNICE due to the
absence of appropriate interprocess mechanisms in UNIX 4.1BSD; compar-
able utilities run under 4.1BSD in EMACS process windows; under UNIX
4.2BSD, it is expected that more flexible LEDIT-like systems will be sup-
portable.)

To use LEDIT, one must have a suspended EMACS that was invoked from the
VMS command level (not from a C-shell). Then one invokes a FRANZ LISP
executable image, also from VMS. The typical operations performable from
LISP are:

(ledit) or "control-E" suspends LISP and starts up the suspended EMACS
(ledit function-name pp) loads the function "function-name" into EMACS.

Note that when EMACS is resumed, it will look for temporary files of the
form leditz.tmp, which it uses for communication with LISP. If an error
causes either process to terminate abnormally, the user may find some of
these files in his directory; they should be deleted by hand.

From EMACS, the user may either mark a current defun for later loading,
or load functions into LISP and resume LISP execution:

ctrl-c = save current file and restart LISP without loading anything
meta-ctrl-s = mark current function to be saved and loaded later
meta-ctrl-l = load all marked functions into LISP and resume LISP.

The EMACS MLISP file /emacst1ib/ledit.m1 loads the EMACS side of the LEDIT
system and defines the EMACS parameter ledit-save-files to allow user con-
trol of which files are saved when EMACS is suspended and control is
returned to LISP. The meanings of the values of this flag are as follows:
The Testbed EMACS Editor System

*1 = save current file [DEFAULT]
0 = don't save anything
+1 = save all modified files.
The Testbed EMACS Editor System
Section 6

Testbed Applications Programs

6.1. Overview of Applications Software Contributed to Testbed

In addition to SRI International, the institutions contributing software systems to the DARPA/DMA Image Understanding Testbed are Carnegie-Mellon University (CMU), the Massachusetts Institute of Technology (MIT), Stanford University, the University of Maryland, the University of Rochester, and the University of Southern California (USC). Modified or re-implemented versions of some routines have also been provided by a DARPA project at Hughes Aircraft.

Software modules integrated into the Testbed system include main programs, program systems, libraries of user utilities, graphics routines, and image access routines. Each of the designated Testbed contributor sites has defined and delivered contributions to the Testbed system. Among the research contributions are four modules from SRI and two from CMU; also running on the Testbed are one contribution each from Rochester, Maryland, and USC, as well as a major system in FRANZ LISP from Stanford. MIT has contributed a Stereo matching system in Lisp Machine LISP which has been integrated into the Testbed environment. CMU has also furnished utilities, graphics, and picture access packages, while SRI has implemented an extended picture format and many user utilities.

A summary of the currently operational research software contributions is given in Table I.

The following subsections summarize the status of each of the currently integrated contributions.

6.2. Carnegie-Mellon University Contributions

CMU Grinnell Graphics and Image Packages.

Date received: August 1981.
Responsible party: David McKeown.
Language: C (Berkeley UNIX) running on the VAX.

Documentation:
For complete documentation, see CMU002, "Grinnell Display Software Support," CMU003, "CMU Image Format and Paging System," and CMU004, "Image File Naming Conventions." man entries providing high level descriptions are available under the /iu/tb/man/man3 directory in the files cmuimglib.3, gmfrmlib.3g, and gmrlib.3g. Testbed-based extensions to the CMU capabilities are described in the files dspilib.3g, frmilib.3g, imgfrmlib.3v, imglib.3, imgnamelib.3, namelib.3, picolib.3, and piclib.3.
Testbed Applications Programs

<table>
<thead>
<tr>
<th>INSTITUTION</th>
<th>CONTRIBUTION</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRI</td>
<td>Road expert</td>
<td>MAINSAIL</td>
</tr>
<tr>
<td></td>
<td>RANSAC</td>
<td>MAINSAIL</td>
</tr>
<tr>
<td></td>
<td>CAMDIST camera modeler</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>SHOWDTM terrain map utility</td>
<td>C</td>
</tr>
<tr>
<td>CMU</td>
<td>Picture access and display packages</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>PHOENIX segmentation system</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>Stereo/correlation system</td>
<td>C</td>
</tr>
<tr>
<td>STANFORD</td>
<td>ACRONYM 3-D model-based vision system</td>
<td>FRANZ LISP</td>
</tr>
<tr>
<td>MARYLAND</td>
<td>Relaxation package</td>
<td>C</td>
</tr>
<tr>
<td>ROCHESTER</td>
<td>Generalized Hough transform system</td>
<td>C</td>
</tr>
<tr>
<td>MIT</td>
<td>Stereo reconstruction system</td>
<td>LISP MACHINE LISP</td>
</tr>
<tr>
<td>USC</td>
<td>Linear feature analysis</td>
<td>C</td>
</tr>
</tbody>
</table>

Description:
These packages provide basic access to the functions of the Grinnell display system, as well as the capability of accessing image data files independently of the display system.

Remarks:
A number of minor modifications were needed to make the CMU package work with the SRI Grinnell configuration. The present code will support any CMU configuration or the SRI Testbed configuration. The CMU image access package has also been integrated into the testbed environment; a new, extended Testbed picture format has been implemented.

PHOENIX Segmentation Package
Date received: December 1981.
Responsible party: Steve Shafer.
Language: C (Berkeley UNIX) running on the VAX.

Usage:
Invoke the command

```
phoenix inimage -o cutimage -f feat1 [feat2 ...] [-i file] [-l file]
[-e] [-s] [-O file] -r reg# -R reg#
```
Testbed Applications Programs

Documentation:

Description:
PHOENIX performs image segmentation by recursive region splitting. This segmentation package uses the Ohlander histogram-partitioning method to segment color imagery. Each pixel in the input image is assigned a segment identification label according to the image characteristics and the parameters selected. Segmentation is carried out hierarchically, with higher level regions segmented into subregions. Segmentation ceases in a given region when the program criteria for significance of the next level of segmentation have not been met.

Remarks:
This system has a sophisticated user interface and a very useful checkpoint system.

Stereo Reconstruction and Correlation Package

Date received: September 1981.
Responsible party: Charles Thorpe.
Language: C (Berkeley UNIX) running on the VAX.

Usage:
Invoke either of the two commands

```
correlate [-nqros#v#m#t#j#filename -filename]
```

```
stereo
```

and answer the queries for additional program input parameters.

Documentation:

Description:
This is a C version of the Moravec correlation and stereo reconstruction package written originally in SAIL at Stanford. The package consists of two portions: CORRELATE selects a set of "interesting" points in one image, using the Moravec interest operator, and attempts to locate the corresponding points in a second image using an efficient hierarchical correlation matcher; STEREO uses the same method as CORRELATE to find corresponding points in a series of up to 9 images, then employs the Moravec method to assign a stereo depth value and confidence level to each match point.

Remarks:
This package implements all the basic capabilities of the original Moravec SAIL system, plus a number of enhancements introduced by Charles Thorpe.
Testbed Applications Programs

6.3. University of Maryland Contributions

Relaxation Package

Date received: Final version received 9 July 1981.
Responsible party: Bob Kirby (author: Russell Smith, revised by Joe Pallas).
Language: C (Berkeley UNIX) running on the VAX.

Usage:
Invoke the command

```
relax
```

or invoke various elements of the package individually. The individual programs making up the system include:

```
defcom defnbr imgprb prbimg relax relaxpar setup.csh
```

Documentation:
A man entry for RELAX is available in /u/tb/man/man1/relax.1. Testbed documentation is provided in "The RELAX Image Relaxation System: Description and Evaluation."

Description:
This relaxation package takes an initial set of probabilities that a pixel belongs to each of a set of classes and iteratively adjusts them according to the class probabilities of neighboring pixels. Two options are provided: an additive Hummel-Zucker-Rosenfeld relaxation algorithm and a multiplicative Polig relaxation algorithm.

A simple two-class algorithm is used to generate a luminance representation corresponding to the reassigned pixel probabilities; the resultant grey scale image can be displayed for the user to monitor the progress of the relaxation process.

Remarks:
A multiclass method of generating probability assignments corresponding to luminance values has been added for test and demonstration purposes.

6.4. MIT Contributions

Marr-Poggio-Grimson Stereo System

Received: February 1983
Responsible parties: Mike Brady, Eric Grimson, and Keith Nishihara.
Language: Lisp Machine LISP.

Documentation:
Current documentation consists of comments in the programs themselves. Additional documentation is planned.

Description:
This system uses zero-crossing matches at several scales to compute disparity values between stereo pairs. Additional consistency checking is available as an option.

Remarks:
This system makes use of an extensive package of Lisp Machine vision utilities, some generated at MIT and some revised or newly developed at SRI. In particular, support for reading and writing 8-bit images in Testbed format has been provided; images may be read and written on the local Lisp
Testbed Applications Programs

Machine file systems, or may be read and written across the local network to the Testbed VAX. The MIT Lisp Machine utility systems have been enhanced for use with the Testbed and now run in the Symbolics 3600 Lisp Machine environment. Convolutions are currently done in software. To enhance performance, it would be desirable to have convolution hardware on the Lisp Machines.

6.5. University of Rochester Contributions

Hough Transform Package

Date received: May 1981.
Responsible parties: Dana Ballard and Bill Lampeter.
Language: C (Berkeley UNIX) running on the VAX.

Usage:
Invoke the command

    ghough

and answer the queries for program input parameters.

Documentation:
A man entry for GHOUGH is available in /u/tb/man/man1/ghough.1.
Testbed documentation is available in "The GHOUGH Generalized Hough
Transform Package: Description And Evaluation."

Description:
This program takes a geometric shape template and attempts to find
matching shapes in the image using the generalized Hough transform tech-
nique. The matched shapes may differ in displacement, rotation, and
scale from the supplied shape template. The most likely values of location,
rotation angle, and scale are output by the program and the reoriented
templates are displayed over the image.

Remarks:
The CMU graphics package has been used as a basis for incorporating full
interactive graphics into this system for both template generation and picture
processing. Several improvements have been made in the user interface
and in the efficiency of the code, and the package was extended to
handle multiple instances of an object.

6.6. SRI Contributions

Road Expert

Date received: January 1981.
Responsible party: Lynn Quarn and Helen Wolf
Language: MAINSAIL running under EUNICE on the VAX.

Usage:
While connected to the /u/sri/road/cmd directory, start up the MAINSAIL
system and invoke the trkacq module.

Documentation:
A man page is available in /u/tb/man/man1/road.1. Demonstration
instructions are provided in Appendix A of this document.
Testbed Applications Programs

Description:
This package acquires and tracks linear features such as roads in aerial imagery. Tracking is done automatically in imagery with a known ground truth data base. Once a road is identified and tracked, a separate subsystem is available to analyze road surface anomalies and to place them into categories such as vehicles, road surface markings, and shadows.

RANSAC Image-to-Data-Base Correspondence Package
Date received: January 1981.
Responsible parties: Martin Fischer and Bob Bolles.
Language: MAINSAIL running under EUNICE on the VAX.

Usage:
While connected to the /iu/vision, start up the MAINSAIL system and invoke the intmod module.

Documentation:
A man page is available in /iu/tb/man/man1/ransac.1. Demonstration instructions are provided in Appendix A of this document.

Description:
This package selects a best fit to an array of control points possibly containing gross errors. RANSAC offers significant improvements over least-squares fitting techniques if gross errors are present. A typical application is to compute the camera model from a given set of landmarks in aerial imagery.

camdist
Date received: March 1983
Responsible party: Marsha Jo Hannah
Language: C(Berkeley UNIX) running on the VAX

Usage:
Invoke the command
  camdist [options]
with desired options.

Documentation:
A man entry is available in /iu/tb/man/man1/camdist.1.

Description:
Camdist provides a facility for performing a generalized least-squares solution for the relative position and orientation angles between two cameras, given a series of points in the two camera views, and/or for calculating the distances to the points from such information. Wild points are automatically edited out, and the associated error propagation is done from the image plane points, through the camera model, to the distances.

showdir
Date received: February 1983
Responsible party: Marsha Jo Hannah
Language: C(Berkeley UNIX) running on the VAX
Testbed Applications Programs

Usage:
invoke the command
    showdtm [options]
with desired options.

Documentation:
A man entry is available in /iu/tb/man/man1/showdtm.1.

Description:
This is an interactive program for displaying a digital terrain model and producing either a perspective grid plot or a perspective range image of a portion of a model. When invoked with no arguments, showdtm will prompt for the name of a terrain model (an image in testbed format), then wait for commands. If the name of an image file is specified, the program will open that, then wait for commands. If an initial command string is specified, the program will execute each of those commands, then wait for more.

6.7. Stanford University Contributions

ACRONYM System

Date received: March 1982.
Responsible parties: Tom Binford and Rod Brooks.
Language: FRANZ LISP running on the VAX. An extensive macro package is used to preserve most of the original MACLISP code.

Usage:
While connected to the directory /iu/acronym/sys, invoke acronym. Connect to the models directory using (chdir '../models'), invoke (PARSE model/file-name), and proceed with the desired ACRONYM process.

Documentation:
Some basic instructions are contained in /iu/acronym/info and are accessible by invoking info.com from VMS; this command starts up an EMACS INFO system with a special ACRONYM node. Other information is available in /iu/acronym/doc. A man entry for ACRONYM is available in /iu/tb/man/man1/acronym.1. See also "ACRONYM: The Facts," a partially completed Stanford University document by Rodney Brooks. A more complete documentation set will eventually be supplied by the Hughes Aircraft ACRONYM-based vision project.

Description:
ACRONYM takes a scene that has been reduced to a set of two-dimensional ribbons and searches for instances of three-dimensional models that have been supplied to the system as data. This is a rule based system that allows great flexibility in the interpretation and scene-prediction process. Models can also be defined in a very general manner by using generalized cones, constraints, and subclass definitions.

Remarks:
Reduction of an image to a list of ribbons must now be done by hand, starting with a corresponding file of line segments generated by a program such as the Nevatia-Babu line finder. While some test imagery is available with the ribbon reduction already carried out, the Testbed ACRONYM system
Testbed Applications Programs

would profit from the addition of an automated ribbon reduction module. Such a module has been promised by the Hughes Aircraft project.

6.8. University of Southern California Contributions

Nevatia-Babu Line Finder

Date received: June 1981 (SAIL version);
June 1982 (C version from Hughes Aircraft).
Responsible party: Ram Nevatia at USC;
Julius Bogdanovich at Hughes Aircraft.
Language: C (Berkeley UNIX) running on the VAX.

Usage:
Connect to the /usr/usc/tst directory and run the following programs in order:

  ../bin/convolve
  ../bin/thrin
  ../bin/psmaker
  ../bin/linkseg

The output on seg.dat may be put in a device-independent display format by invoking ../bin/segdisp; the Testbed graphics utility can be used to show the resulting display file.

Documentation:
For documentation, see the Hughes Aircraft document """"C VERSION OF THE NEVATIA-BABU LINEFINDER."""" A brief man entry is available in

  /bu/tb/man/man1/line.1.

Description:
This package extracts linear features from an image and produces a data base of line segments. The Testbed C version supports 5x5 convolution masks configured to identify edges oriented at 30-degree intervals. The edges are then linked together into chains and broken into straight line segments.

Remarks:
The C version of this package lacks some of the parallel-line and supersegment extraction features of the SAIL version. Although it would be useful for comparison purposes to have these capabilities available, it is not known just how effective the algorithms are on general imagery. Support for different convolution masks and parallel-line extraction may be added at a later time.
Section 7

Testbed Utility Programs

A variety of utilities, research tools, and other programming aids have been assembled in the context of the Testbed. Some have been contributed by other Testbed participants, some are modified versions of utilities originating at other sites, and some were developed specifically for the Testbed at SRI.

7.1. Image Manipulation Utilities

Image files may be processed in a variety of ways by the following routines:

normalize
This CMU program normalizes a grey-scale image to produce a new output image with desired compression or clipping. SRI modifications allow grey-scale stretching as well.

reduce
This CMU program extracts a subwindow of an image or rescales an image by an integer sampling factor.

invert
This program inverts the matrix of data making up a picture file by putting the top row at the bottom, and so on.

shapeup
The original CMU program of this name has been entirely rewritten to support conversions among many image formats, including the new Testbed format.

convert
This routine supports color transformations of various types, such as the conversion of red-green-blue imagery to YIQ or Hue-Intensity-Saturation features.

7.2. Image Display Utilities

An entire subsystem of routines is configured for quick and easy display of images on a default display device. For complete details, see the appropriate Testbed man pages.

show
Displays any rectangular portion of a black and white, anaglyphic stereo, or color image anywhere on the display screen.

erase
Erasers a window on the default display device.

overlay
Places a one-bit image in an overlay window on the default display device.
Testbed Utility Programs

describe
Types out a brief description of a picture file.

graphics
A simple line-drawing display system that accepts commands and file input
in the form:

    screen x1 y1 x2 y2
    line x1 y1 x2 y2

defines a floating point screen coordinate system;
draws lines relative to the declared coordinates.

imgsys
Allows picture files to be accessed, described, and displayed in windows on
the display system; it is based on the ICP command interpreter.

dpy
Uses the ICP interpreter to implement an alternative interactive display
and windowing system.

gmrays
Performs low-level operations to facilitate installation and debugging of the
Testbed Grinnell display system.

7.3. User Libraries

The following libraries of utilities permit useful information or features to be made
available to users of the Testbed:

sublib
Lowest level Testbed utility routines, including the following sublibraries:

    arglib asdlib blklib clib doclib errlib filelib icplib
    listlib listlib mathlib matrixlib parselib pntlib
    seglib stringlib striplib syslib timelib vectorlib wdlib

imagelib
Testbed picture data access and graphics utilities, including the following
sublibraries:

    dsplib frmlib gmrilb hdrlib imgfrmlib imglib
    imgnamelib picolib piclib

visionlib
High-level routines that interface between image file access and display
systems, that refer to high-level blk data structures, and that deal with
"image understanding" operations. The following sublibraries are included:

    blklib convlib histlib intervlib matchlib
    patchlib polygonlib relaxlib windowlib xformlib

cmuimglib, cmuvrlib
CMU analogs of imagelib and visionlib for use in running native CMU pro-
gram systems.

The following library-related files and functions may be of special interest:

testbed.h
In the standard Testbed include directory, ~/test/include, there are a
number of files that are called for as include files in standard Testbed
Testbed Utility Programs

programs. The file testbed.h contains standard global variables, constants, and macros that are found in Testbed code.

arglib
A set of utility routines available for requesting, parsing, and checking program parameters. To use arglib for command-line parsing, one begins by invoking

```c
openarg(argc, argv, <# of required arguments>,
        "Usage string to print if given fewer args");
```

Then one is free to use the remaining routines to parse out the nth "argv" argument or to ask the user for values. For example, to read the third argument or enter the default if the user types a carriage return, use

```c
argval = intarg(3, minvalue, maxvalue, defaultvalue,
                "Optional prompt string in case I have to ask the user");
```

You may substitute NOTINT for 3 if you just want a value from the user without regard to the command line. (NOTINT is defined in testbed.h.) For further examples of usage, see, for example, /u/tb/src/shapeup/shapeup.c and the Testbed man page.

printerr
A Testbed package that supports flexible and user-friendly reporting and handling of error conditions; see the man entry for details.

7.4. Miscellaneous Packages

Among the generally useful utility systems available on the Testbed are the following:

ccr
Cross-reference program for the C language.

ci
A command interpreter contributed by CMU. It allows one to link a variety of subroutines into a top-level command processor and to invoke the subroutines with arguments provided interactively by the user. Extensive help and utility facilities are supported.

icp
A command interpreter for the C language contributed by SRI. It is very similar to ci, except that its treatment of arguments and local variables is more general. icp, for example, is able to invoke system cr user subroutines directly, while ci must have an argument-parsing interface written for each routine. icp also has more powerful facilities for context scoping.

doc
A CMU utility for generating UNIX man entries to document programs with little need to know details of the troff phototypesetting system. All information that the program needs to generate a syntactically correct man entry is obtained by interrogating the user.

indent
Routine for carrying out pretty-print indentation of C language files.

sc
Simple but serviceable public-domain spread-sheet program.

view
Utility for looking at segments files containing one or two-dimensional
Testbed Utility Programs

arrays. This routine basically extends the capabilities of `od` to include two-dimensional array indexing and file headers.

whist
A program for adding history entries to `man` pages or to source code files.
Section 8

Guidelines for Programming in the Testbed Environment

The Image Understanding Testbed will be of most benefit to the research community if it evolves into a uniform environment that can be utilized by many different sites. In the course of assembling the system of programs described here, we have noted a number of programming practices that would have made the task easier. The following paragraphs summarize some of the things individual programmers should do to make their programs easier to integrate into the Testbed environment and of more general use to the community. We present first some general suggestions on programming style, and then a list of specific procedures for integrating a new program into the Testbed system.

8.1. Programming Style Guidelines

**COMMENT YOUR CODE**
Each file of source code should begin with a series of comments including:
- Name of the file itself;
- General description of the functionality of the routines in this file;
- One-line synopsis of each subroutine contained in the file;
- Author and date of creation;
- Author and date of significant modifications;
- Warnings about assumed pathnames and other site-dependent aspects.

General comments, especially remarks about special techniques and assumptions, should be dispersed where appropriate throughout the code.

**MODULARIZE YOUR CODE**
Only closely related subroutines should be placed in the same file. Creating long files with large numbers of subroutines is poor practice and makes location and maintenance of the code very difficult. Maintain a clear hierarchy in your systems; low-level routines should be grouped together, and the higher levels of code should utilize the lower level calls while avoiding direct use of any low-level structures. The dubious practice of mixing low-level, site-dependent structures into high-level code calls has been one of the chief sources of difficulty in integrating Testbed systems together. Equally troublesome is the practice of passing high-level structures down to low-level routines; this prevents other independent high-level code from interfacing cleanly to the low levels.

**KEEP I/O FLEXIBLE AND REDIRECTABLE**
Make sure your program will run normally under script and other systems that replace normal teletype I/O by pipe I/O. This allows your program to be run as a subprocess from, e.g., a LISP expert system that puts another level of intelligence between your program and the user. Also attempt to keep the terminal data input and output streams clean enough so that
Guidelines for Programming in the Testbed Environment

they may be reassigned to files that are generated or used by other pro-
grams in complex chains of processes.

ISOLATE INTERFACES TO SITE-DEPENDENT MODULES

If you must use routines that are site-dependent in some essential way, group all such references together in a clearly identified set of modules. Think about how you would transport the functionality to another site and try to make that task as transparent as possible.

USE RELATIVE PATH NAMES

Each site will have unique directory systems and paths. Use of absolute paths in compiled code and in makefiles causes severe difficulties in transporting the code to other sites. Use relative file path names wherever possible, especially in compiled code. In makefiles, the best technique is to define all needed absolute paths symbolically at the beginning of each makefile, and to force all subsequent path references to be made relative to the symbolic definitions.

COMPILE FROM SCRATCH BEFORE DISTRIBUTING

If you are planning on sending a system out to another site for testing or evaluation, gather all of your sources and libraries together and attempt to compile everything from scratch in a different directory from your usual one. This is one way to check that your system is really self-contained and to find potential problems such as incomplete directory paths.

PROVIDE SAMPLE INPUT, OUTPUT, AND DEMONSTRATIONS

If you are sending code to other sites or having it evaluated by other users at your site, provide sample input and output so that it is easy for others to verify that your code is working as you expected it to. Another very useful practice is to have a working demonstration more or less built into the program as a default series of parameters. Some mechanism should be in place that allows a quick and foolproof demonstration of the sorts of things the program does.

WRITE A MAN PAGE

Using the doc system or using an existing man entry as a template, one can easily create a UNIX man entry that describes the basic characteristics of your software. You should plan it so that it is possible to provide a detailed written description of your software; a man page is the minimum acceptable documentation that must be provided to give other users a rough idea of what you have done.

8.2. Program Installation Guidelines

Installing a program system named PROGRAM in the Testbed directory hierarchy involves several simple steps:

* Install the source code in the directory /tu/tb/src/PROGRAM/
* Install a makefile that generates the program and moves it to the
Guidelines for Programming in the Testbed Environment

Executable file /u/tb/bin/PROGRAM.

* Add the line cd PROGRAM; make to the file /u/tb/src/makefile.

* Place a man page describing the function of PROGRAM in the file /u/tb/man/man1/PROGRAM.1.

* Add a synopsis line for PROGRAM to the file /u/tb/lib/whatis, or run sh mkmwhatis.sh.

Programs with complex input data are often provided with subdirectories such as /u/tb/src/PROGRAM/data or /u/tb/src/PROGRAM/demo to give examples of data and program usage. Complete sample demonstrations of Testbed application programs are usually installed as subdirectories of the /u/testbed/demo/ directory; in such a subdirectory, the demo file should be a shell file invoking a standard demonstration sequence using data files contained locally within the subdirectory.

Major new programs should have brief descriptions added to Section 6 of this document, along with demonstration instructions in Appendix A.
Appendix A

A Sampler of Testbed Demonstrations

We present here basic instructions for running some characteristic demonstrations of each of the Image Understanding Testbed contributions. We remind the reader that these are examples of transfer of established IU technology and do not necessarily represent current research efforts. The individual systems that the user may choose to demonstrate include the following:

* SRI - Road Expert
* SRI - RANSAC
* SRI - SHOWDTM
* SRI - CAMDIST

* CMU - CORRELATE
* CMU - STEREO
* CMU - PHOENIX

* Rochester - GHOUGH

* Maryland - RELAX

* USC/HUGHES - LINE

* Stanford - ACRONYM
SRI - Road Expert

This demonstration tracks a roughly located road in an aerial image and locates road anomalies such as vehicles.

`set def init[sri.road.cmd]`
`set ns`
`*trkacq`
`trkacq: image pm280a`
`picfn: get vision/1pm280a4s.pic`
`picfn: sho`
`picfn: get vision/1pm280as.pic`
`picfn: quit`
`trkacq: err`
`errmod: abs 10`
`errmod: quit`
`trkacq: sho`  ! Show the known roads
`trkacq: track bita`  ! Track one road; sr2b, pmwa are others
`monitor: <cr>`
`trkacq: analyze`  ! Look at the anomalies
`trkacq: quit`  ! End demo.
`<cr>`
SRI - RANSAC

This is a model matcher, applied to camera modeling in this demonstration.

$ set def iu: [vision] 
$ ms 
* intmod 
[open display] Y > switch ("rdemo.cmd"): ! starts RANSAC with synthetic data 
   ! 1. Show ground control points 
   ! 2. Predict ground search areas 
   ! 3. Pick 3 points and predict the rest; 
<cr> in response to prompts. ! known pts in red, candidate matches in green, 
   ! true matches in yellow. 
   ! 4. Try camera models, X out inconsistencies. 
   ! Circle good matches when good model found. 
> ^C ! Kill process.

The following steps are executed:

1. Show ground control points. 
2. Predict ground search areas. 
3. Pick 3 points and predict the rest; known pts in red, 
   candidate matches in green, true matches in yellow. 
4. Try camera models, X out inconsistencies. 
5. Circle good matches when good model found.
SRI - SHOWDTM

This is a utility to plot contour views of digital terrain map data; the demonstration uses the Phoenix South Mountain Park data set.

% cd /iu/testbed/demo/mtpark
% demo
  <Wait for contour plot to finish,
   then enter single-character commands>
  *1 <Switch to image>
  *2 <Switch back to contour plot>
  <To select a different view>
  *p
  *c
  *p <Plot the newly specified view>
  *1
  *2
  *q <Quit program>
SRI - CAMDIST

This is a utility computes an iterative fit to a camera model.

```
% cd /iu/testbed/demo/camdist
% demo
```

The program results for a sample data set contained in the `/iu/testbed/demo/camdist` directory are displayed on the terminal as they are computed. No graphic displays are generated.
CMU - CORRELATE

This demonstration correlates Moravec interest points on a stereo pair of images.
  % correlate
  # Repeatedly typing <cr> will start up standard set of demo parameters.

CMU - STEREO

This demonstrates slider stereo depth reconstruction on a triple of images. Slider
stereo gives an effectively broader baseline than stereo systems working on a single
pair, and improves the statistical reliability of image matches and depth values.
  % stereo
  # Repeatedly typing <cr> will start up standard set of demo parameters.

CMU - PHOENIX

This demonstrates some capabilities of the Ohlander method for segmenting color
imagery.

PHOENIX demo number 1.

  % cd /iu/testbed/demo/chair   # -* -cmds, *x* - vars
  % demo
  > seg                        # hsmooth 5 ; flags = +A
  > stop
  > exit

  % cd /iu/testbed/demo/portland
  % show9
  % demo
  > describe region 22
  > history 22
  > describe region 7
  > exit

PHOENIX demo number 2.

  % cd /iu/testbed/demo/chair   # -* -cmds, *x* - vars
  % demo
  > restore demo.okp
  > dis seg
  > dis reg 12
  > describe region
  > history

PHOENIX demo number 3.

  % cd /iu/testbed/demo/skyline
  % demo
  Z
Rochester - GHOUGH

This demonstrates the ability of the Hough transform technique to identify partially obscured, uniquely shaped objects in aerial imagery and to determine the location, angular position, and scale of the object.

GHOUGH demo number 1.

```
% cd /iu/testbed/demo/lake
% demo
```

# quick, no-pause run-through

GHOUGH demo number 2.

```
% cd /iu/testbed/demo/lake
% gough
> <lake.cmd
> [position cursor]
> first 5
> top 1
> disp 1
> q
```

GHOUGH demo number 3.

```
% cd /iu/testbed/demo/circle
% demo
```

OR

```
% dspsys
> < dsp.cmd
```

GHOUGH demo number 4.

```
% cd /iu/testbed/demo/ellipse
% dspsys
> <dsp.cmd
```

# reconstruction of very long analysis
Maryland - RELAX

This demonstrates the elementary functions of the relaxation procedure for evolving noisy two-component data into its most probable identification.

RELAX demo number 1.

```
% cd /iu/testbed/demo/tank
% show9                    # displays result of HZR iteration
```

RELAX demo number 2.

```
% cd /iu/testbed/demo/relax
% demo
> tank.cmd                 # IMGSYS commands
% cleanup
```

RELAX demo number 3.

```
% cd /iu/testbed/demo/tank
% hdemo                    # full demo of HZR
% pdemo                    # full demo of Peleg
```
This demonstration shows the elementary functions of the convolution, thresholding, predecessor-successor linking, and segment compilation functions of the Nevatia-Babu line finder.

% cd /lu/usc/tst
% ../bin/convolve
    <user responses>
% ../bin/thrin
    <user responses>
% ../bin/psmaker
    <user responses>
% ../bin/linkseg
    <user responses>
% ../bin/segdisp
    <user responses>
        output-file = disp.txt
% graphics
    att gmr
    open
    < disp.txt
    exit
This demonstration takes an image of the ACRONYM program that has a prestored analysis of a San Francisco airport scene and displays the L-1011 aircraft parts that have been identified.

```
% cd /iu/acronym/sys
% airport
ACRONYM> (open-acronym-graphics)
ACRONYM> (DISPLAY-LIST LIN SPGLS) ; show original ribbon scene of SFO airport
ACRONYM> (DISPLAY-LIST LIN SIGLS) ; show found L1011's
ACRONYM> $SCENE-PREDICTION ; shows the constraint graph
ACRONYM> (DISPLAY-PREDICTION-GRAPH $SCENE-PREDICTION)
(Z0032)
ACRONYM> (DISPLAY-PREDICTION-GRAPH 'Znnnn) :Z0055, Z0037,63,96
ACRONYM> 'Z0031, 34, 36, 64, 92
ACRONYM> (IDB)
ACRONYM> (FLUSH-ATM)
ACRONYM> (PARSE ...)
...
ACRONYM> (KWIT)
```

This is a simple demonstration of the loading and examination of a graphic display of an ACRONYM model file.

```
% cd /iu/acronym/sys
% acronym
ACRONYM> (chdir "./.models)
ACRONYM> (PARSE tbcube) ; or "tbtank", "motors"
ACRONYM> (G) ; to invoke graphing utility
 ; Note: type "?" for graphics command summary.
 ; User is advised to begin with "PP" to keep image on screen.
 ; "Z" exits the graphics mode.
ACRONYM> (KWIT)
```
Appendix B

The Testbed Picture File Access and Format

The Image Understanding Testbed utilities support a general picture file format which accesses pictures in blocked format to optimize use of the VAX-11/780 paging hardware. The picture format consists of a 1024-byte ASCII header followed by the blocked picture data, and a trailer that can be used for supplementary data.

B.1. Picture Data Access

Manipulation of picture data files is accomplished by the picture utilities such as

- `PIC openpic(filename, mode);`
- `PIC newpic(filename, protection);`
- `PIC tmppic();`
- `PIC closepic();`

When the picture utilities are used to open a picture, the ASCII header is translated into a simple structure, called a "PIC," which contains the corresponding binary data that user programs need to use. "Opening" a picture means to retrieve the image header and data from a named disk file and to place that data where it can be used by application programs. "Closing" a picture means to remove the data from the application program data space.

A library of utilities, `/usr/lib/imagelib/pictlib`, is used to to retrieve picture data and pass them to application programs for computation. Retrieval may be accomplished on a pixel basis, or by image rows, columns, or blocks.

The picture file format itself is described briefly below; for further details, see the `hdrformat`, `docformat`, and `imgformat` manual pages. Descriptions of the picture file manipulation utilities and picture data retrieval utilities are available in the `pictlib` and `pictlib` manual pages.

B.2. Picture File Format

Image data is stored in a block raster format consisting of rectangular data blocks that cover the image in a raster scan pattern. (Scan direction is recorded in the header scan type field.) Pixels within a data block are ordered in the same raster pattern. All data blocks in a picture file are the same size, and each must contain an integral number of pixels on each row. Any additional bits in the block row will be padded with zeros following the valid image data.

The array of valid data pixels is an image. The data elements are addressed by pixel within scan line and by scan line within the image. These may be thought of as rows and columns, but the relation to any physical scene depends on the scanning direction.

Data blocks are groupings superimposed on the linear stream of pixels and padding
that form the data file. The basic unit is the block row, which contains an integral number of pixels and possibly some padding bits. Block rows are stacked into blocks, which contain rectangular portions of the image. Blocks are grouped into block strips, which contain an integral number of scan lines. These block strips are stacked to form the data file.

B.2.1. Picture header data from /iu/tb/include/tbhdr.h

```c
#define IMGHDRLEN 1024 /* Standard IMGHDR length. */

typedef struct IMGHDR {
    char formattype[8];      /* Format, version: "SRI TB01" */
    char recordtype[8];      /* Header/trailer type: 'IMGHDR' */
    char recordbytes[8];     /* Length of this record: ASCII value of IMGHDRLEN. */
    char scanotype[8];      /* Scan direction: pixel-to-pixel, then scan-to-scan. LRTSCAN is normal for testbed image data. LRTSCAN is the normal television raster scan order. */
    char bandtype[8];       /* Pixel band format code: UNTYPED: arbitrary byte pattern; UNSIGNED: unsigned binary integer. TWOSCOMP: two's complement signed integer. VAXREAL: VAX floating-point format. */
    char imagebands[8];     /* Bands per pixel. <1> means the string "1" is a typical field value */
    char bandbits[8];       /* Bits per band. <8> */
    char pixelbits[8];      /* Bits per pixel. <8> */
    char linepixels[8];     /* Pixels per scan line. <512> */
    char imagemlines[8];    /* Number of image scan lines. <512> */
    char blockcols[8];      /* Bytes per block row. <32> */
    char blockrcw[8];       /* Rows (scan lines) per block. <32> */
    char stripblocks[8];    /* Blocks per scan line strip. <32> */
    char imagestrips[8];    /* Strips per image. <32> */
    char unused[IMGHDRLEN-112]; /* Pad to recordbytes bytes. */
} IMGHDR;
```

B.2.2. Octal Dump of Typical Picture Header

Below is an octal dump, `od -v`, of a typical Testbed picture header (blank spaces denoting the ASCII "space" character fill out the 16-column rows):
<table>
<thead>
<tr>
<th>Address</th>
<th>Printhead 1</th>
<th>Printhead 2</th>
<th>Printhead 3</th>
<th>Printhead 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0002000</td>
<td>512</td>
<td>512</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>000140</td>
<td>16</td>
<td></td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

*Image data follows at this point*
Appendix C

Graphics System Summary

The current Testbed graphics system is based on the CMU Grinnell graphics package. This package is documented separately in the Carnegie-Mellon University document CMU-802 "GRINNELL DISPLAY SOFTWARE SUPPORT". Here we attach a brief summary of the available facilities.

**Allocation of Resources**

```c
  g_init(0)
    Initialize Grinnell.
  frm = g_allocfrm(mode, pixelsize)
    Allocate a frame descriptor.
  g_chgdisp(frm)
    Display the specified frame.
  g_freefrm(frm)
    Free specified frame allocation.
  g_ersfrm(frm)
    Erase frame.
  g_grmon(), g_grmoff()
    Turn Grinnell visibility on or off - nondestructive.
  g_exit(0)
    Deallocate all Grinnell resources and prepare to exit.
```

**Pixel Access**

```c
  g_setpixel(frm, row, col, pval), g_setc pixel(frm, row, col, rval, gval, bval)
    Set value of specified pixel in the frame.
  g_getpixel(frm, row, col, pval), g_getpixel(frm, row, col, rval, gval, bval)
    Return pixel value into int pval, etc.
  g_setblkpix(frm, srow, erow, scol, ecol, pval),
    g_setcblkpix(frm, srow, erow, scol, ecol, rval, gval, bval)
    Set rectangular area to specified values.
  g_blkmov(sourcefrm, srow, erow, scol, ecol, destfrm, drow, dcol)
    Copy data from rectangle in sourcefrm to destfrm at (drow, dcol).
  g_blkset(frm, srow, erow, scol, ecol, buf),
    g_blkset(frm, srow, erow, scol, ecol, rbuf, gbuf, bbuf)
    Copy contents of buffers into a frame.
  g_blkget(frm, srow, erow, scol, ecol, buf),
    g_blkget(frm, srow, erow, scol, rbuf, gbuf, bbuf)
    Copy contents of frame into buffers.
```

**Picture Display**

```c
  g_rdimg(img, frm, img_srow, img_erow, img_scol, img_ecol, frm_srow, frm_scol)
  g_rdcimg(rng, bimg, cimg, frm, img_srow, img_erow, img_scol, img_ecol, frm_srow, frm_scol)
    Transfer data from IMAGE structure to frm and display.
  g_wrtimg(...), g_wrtcimg(...)
    Transfer data from frm in the display to the IMAGE structure.
```
Graphics Access

\texttt{g_str} (frm,string,row,col, height, width, direction, add_flag, 
\texttt{tg\_bg\_flag})
\texttt{g_strdraw} (frm,string,row,col, height, width, direction)
\texttt{g_strerase} (frm,string,row,col, height, width, direction)
\hspace*{1cm} Draw or erase strings on screen using various modes.
\texttt{g_linedraw} (frm,srow,scol,erow,ecol,0,0)
\texttt{g_linerase} (frm,srow,scol,erow,ecol,0,0)
\texttt{g_listdraw} (frm,list\_of\_lines,0,0)
\texttt{g_listerase} (frm,list\_of\_lines,0,0)
\hspace*{1cm} Draw or erase lines and lists of lines; last two args are nonfunctional.
\texttt{g_vstrdrow} (frm,string,row,col, height, width, direction, \texttt{value})
\texttt{g_vstrdraw} (frm,string,row,col, height, width, direction, rvalue, gvalue, bvalue)
\texttt{g_vlinedraw} (frm,srow,scol,erow,ecol,0,0,\texttt{value})
\texttt{g_vclinedraw} (frm,srow,scol,erow,ecol,0,0, rvalue, gvalue, bvalue)
\hspace*{1cm} Same as above except that valued lines are drawn.
\texttt{g_rectdraw} (frm,srow,erow,scol,ecol)
\texttt{g_recterase} (frm,srow,erow,scol,ecol)
\hspace*{1cm} Draw or erase a rectangle with all bits set in \texttt{frm}.
\texttt{g_lineclip} (srow,scol,erow,ecol,clip\_srow,clip\_scol,clip\_erow,clip\_ecol)
\hspace*{1cm} Returns clipped values of int\texttt{srow,scol,erow,ecol;}
\hspace*{1cm} value of \texttt{g_lineclip} is nonzero only if clipped line is visible.
\texttt{g_listclip} (frm,list\_of\_lines,clip\_srow,clip\_scol,clip\_erow,clip\_ecol,0,0)
\hspace*{1cm} Same as \texttt{g_listdraw()} except does clipping in specified window.

Overlays

\texttt{g_assocfrm} (image\_frm,overlay\_frm)
\texttt{g_dassocfrm} (image\_frm,overlay\_frm)
\hspace*{1cm} Display overlay with image when \texttt{g\_chgdisp} is invoked.
\texttt{g_ovlcon} (image\_frm,overlay\_frm)
\texttt{g_ovloff} (image\_frm,overlay\_frm)
\hspace*{1cm} Turn overlays on or off.

Mapping Hardware

\texttt{g_mapfile} (frm,mapfile)
\texttt{g_mapwrite} (frm,mode)
\hspace*{1cm} Associate char*mapfile with frm, write it into hardware.
\texttt{g_mapaddr} (frm,mapaddr)
\texttt{g_mapcore} (mapfile,mapaddr)
\hspace*{1cm} Associate short unsigned*mapaddr with frm, load mapfile.

Cursors

\texttt{g_setcur} (frm,curid,row,col,visible,blinking)
\hspace*{1cm} Write cursor to frame.
\texttt{g_getcur} (frm,curid,row,col,cursor\_status)
\hspace*{1cm} Read cursor status into int\texttt{row,col,cursor\_status}.
\texttt{g_curon} (frm,curid,mode), \texttt{g_curoff} (frm,curid,mode)
\hspace*{1cm} Turn cursor on or off.
\texttt{g_trackcur} (frm,curid,row,col,device,switch\_value)
\hspace*{1cm} Track cursor into \texttt{int row, col, switch\_value} using one
\hspace*{1cm} of: device = 1 -> keyboard, device = 3 -> tablet.

Graphics Tablet Support

\texttt{g_suminit()}, \texttt{g_sumclose()}
\hspace*{1cm} Open and close the graphics tablet (Summographics) for use.
\texttt{g_rdsumma} (xpos,ypos,bit0,bit1,bit2,bit4)
Read a point and button value (bits correspond to 4 buttons) from

Zoom and Pan

* g_setzoom(frm_row, col, zoom_factor, crosshair, blink, zoomed)
  * g_getzoom(frm_row, col, zoom_factor, crosshair, blink, zoomed)
    * Initialize or read back values of zoom cursor.
  * g_pan(frm_row, scol_row, scol_column, zoom_factor, zoom_cursor)
    * Track zoomed image by means of g_trackcur device.
Appendix D

A Tour of the Testbed File system
NAME
tbrier – IU Testbed file system hierarchy

DESCRIPTION
The following gives a quick tour through the IU Testbed directory hierarchy.

/u/tb/

    This is the root of the current Testbed file system at SRI. The following
    are the major directories of programs and data which support the
    Testbed system:
    bin  bugs  doc  docsrc  graphics  include  lib  lisp  man  ms  pic  src
    bin/  Testbed-specific executable programs, utilities, and shell scripts
    The source code for all main programs in bin/ is found in the src/
    directory. The documentation for each program or package is
    found in the man/man1/ directory or in the doc/ directory.
    Major Testbed application programs in bin/ are:
    correlate  camdist  ghough  line  phoenix  stereo  showdtn
    RELAX relaxation package programs include:
    relax  defcom  defnhr  imgprg  prbing  relaxpar
    Image processing utility programs are:
    clip  convert  describe  dpy  erase  gmrsys  graphics  imgsys
    invert  normalize  optronics  overlay  reduce  shapeup  show
    The Testbed customized FRANZ LISP environment is tblisp.
    General system utility programs and shell files are:
    apropos  catman  ccr  cpdir  doc  indent  man  mkwhatis.sh
    sc  view  whatis  whereas  whist

    bugs/  text files describing known bugs and suggested system
    improvements.

    doc/  major Testbed documentation text files.

    docsrc/  photo-typesetter source files for Testbed documentation.

    graphics/  source files for device-independent graphics test programs

    include/  declarations and macro definitions used in Testbed C-language
    programs.
lib/  Testbed subroutine libraries and object module archives. These systems of subroutines are documented in the man/man3/ directory.

cmuimglib/
    CMU-style image access utilities for reading Testbed image formats.
cmupiclib/
    CMU routines to open and close picture files.
cmunmelib/
    CMU image name parsing functions.
demo/
    programs to exercise the cmuimglib routines.
picolib /
    CMU picture access routines.
tbpiclib/  Testbed picture access routines.
cmuvsnlib/  CMU-style utilities needed to run CMU code "native."
convlib/
    CMU image smoothing and convolution operators.
devlib/  UNIX and VMS device driver sources.
emacslib/  EMACS editor subroutine libraries.
imagelib/
    Testbed image and display software.
demo/  programs to exercise the imagelib routines.
dsplib/  Testbed display allocation.
frmlib/  Testbed picture frame system.
gmrlib/  Grinnell display device access code.
crslib/  cursor control functions
ctlslib/  display control functions.
frmlib/  CMU frame system.
gmrlib/  miscellaneous Grinnell functions.
include/  declarations and definitions.
hdrlib/  picture header manipulation.
imgfrmlib/
    utilities to display image files.
imglib/  IMG image structure manipulation.
imgnmelib/
    image name parsing functions.
picolib/  Testbed picture access functions.
piclib/  Testbed picture file manipulation.
sulib/  local system utility subroutines.
arglib/ interactive Testbed argument parsing routines.
asklib/ CMU-style query routines used by CI driver.
blklib/ low-level blklib structure manipulation.
ciiib/ CI command interpreter utilities.
doclib/ documentation record manipulation routines.
errlib/ Testbed error-handling code.
filelib/ file manipulation functions.
icplib/ utilities supporting the ICP command driver.
listlib/ list searching and manipulation functions.
lstlib/ LST list datatype routines.
mathlib/ mathematical functions.
matrixlib/ dynamic matrix manipulation routines
parselib/ argument parsing routines.
pntlib/ PNT point datatype routines.
seglbl/ SEG line segment routines.
stringlib/ string manipulation functions.
syslib/ local operating system extensions.
timelib/ time and date utilities.
vecortlb/ vector manipulation functions.
wdwlib/ WDW window structure utilities.
visionlib/ high level machine vision subroutines.
blklib/ high-level blklib structure manipulation.
convlib/ PIC-based convolution routines.
histlib/ histogram manipulation routines.
tierlib/ histogram interval utilities (PHOENIX).
matchlib/ correlation and image matching routines.
patchlib/ patch and region utilities (PHOENIX).
polygnlib/ polygon manipulation utilities (PHOENIX).
reladlib/ relaxation package support routines.
windowlib/ display window (DPY) manipulation subroutines
xformlib/ color coordinate conversion routines.
whatis database for the whatis command

lisp/ root directory for the Testbed tblisp FRANZ LISP system.
src/ sources for the Testbed FRANZ LISP routines in tblisp.
cs / src/ sources for C-language programs loaded into t lisp.
lib/ standard LiSP utility sources; this directory is aliased to /lisp /lib/.
graphics/grinnell/ Grinnell graphics access from Lisp.
help/ help files needed by the t lisp help utilities.

man/ UNIX Testbed manual page files.

man1/ main program documentation. Program sources are
generally in src/ and executable images in bin/.
man3/ subroutine documentation. Subroutine sources are in
lib/.
man5/ data structure documentation.
man7/ miscellaneous documentation.

ms/ MAINSAIL program root directory. The root also contains the
major executable image main.sain.e, the system library system.lib,
and the miscellaneous files needed to reconstruct them.
csrc/ C-language sources needed supporting the MAINSAIL system.
lib/ VMS-style Grinnell and picture libraries to link into
MAINSAIL systems.
src/ MAINSAIL sources for the system.

pic/ the Testbed demonstration picture library. Each picture is
assigned a unique subdirectory that contains image data and
descriptive files. (e.g., red. img, blue. img, green. img, pic. dat). See
also /aux/tbpic/.

src/ root directory for the sources of the Testbed main programs and
utilities. Executable images for these programs are in the bin/
directory, while documentation is in man/man1/.

Major program directories include:
camdist correlate gough line phoenix relax showditm
stereo

Image processing utility program directories are:
clip convert describe dpy erase gmrsys graphics imgsys
invert normalize optronics overlay reduce shapeup show

General system utility program directories are:
catman ccr doc indent man sc whist view

Development in progress:
draw vshow

/iu/acronym/ root directory of the ACRONYM 3D object identification system.

compat/ compatibility packages for converting ACRONYM MACLISP
source code to run under FRANZ LiSP.

doc/ miscellaneous documentation for installers and implementers.
graphics/ graphics packages for ACRONYM displays.
info/ EMACS INFO nodes for interactive documentation.
models/ geometric model files.
sys/ executable image of acronym and all the source files required
to build it.

/iu/usr/
preliminary version of Nevatia-Babu line finder, containing the following
source and executable directories:
apar bin convolve distribution doc linkseg psmaker showdat thin tst

/iu/info
Source of text information for the EMACS online information system.

/iu/testbed
Standard demonstration directory, also a generic user whose files can be
customized for each new user.
demo/
standard demonstration directory. Names of directories often
correspond to the standard demo image data. The file demo in
each subdirectory is a shell file that runs the standard
demonstration.

/aux/tbpic
auxiliary directory of image data. See the /iu/tb/pic directory for
frequently used demonstration images.

/aux/backup
this directory is used to back up programs in certain makefiles. If
something goes wrong in the make procedure, the files can be recovered
from this directory.

SEE ALSO
apropos(1), whatis(1), whereis(1), which (1), ncheck(8), find(1).

BUGS
The location of directories and files is subject to change without notice.
Appendix E

Programming Examples Using the Testbed Picture and Graphics Systems
/*
 * copypic input pic
 * Copy a picture file.
 * COMMENTS
 * This is intended as a template for writing image
 * transformation programs.  For another method of
 * copying picture files, see the copypic() subroutine
 * in the piclib library.  The UNIX cp command may
 * also be used, but only with full path names.
 * The entire picture file is copied with the original
 * header type and blocking factors.  The access
 * permissions of the original file (available via the
 * picmode() routine) are not preserved.
 * The input file name is sought along the path
 * specified by your PICPATH environment variable.
 * If you do not provide the input and output
 * arguments, you will be asked for them.  The input
 * name is used as the default output name, which
 * will simply copy the picture to itself.
 * Type copypic "??" to get a usage message.
 * The quotes are necessary.
 *
 * HISTORY
 * 27-Sep-83  Dave at SRI-IU
 * Added mode preservation and row-by-row copying
 * to a previous demo program.
 */

#include "testbed.h"
#include "imglib.h"
#include "arglib.h"
#include "err.h"

/* External routines */
char *malloc();

main(argc,argv)

    int argc;
    list argv;
{

    string infile,outfile;
    PIC inpic,outpic;

    /* Initialize the error package */
    errname("copypic");
/* Allow arguments to be omitted. */
openarg(argc,argv,0,"copypic inpic outpic");

/* Open the input picture file. */
infile = stringarg(-1,"bw.img","Input picture:");
inpic = openpic(infile);
if (not inpic) then {
  printf("Unable to open file %s",infile);
  exit(ERROR);
}

/* Open the output picture file. */
outfile = stringarg(-2,infile,"Output picture:");
outpic = newpic(inpic->hdr,outfile);
if (not outpic) then {
  printf("Unable to open file %s",outfile);
  exit(ERROR);
}

/* Copy the picture data. */
begin(Copy Data)

    register int nowrow;
    register int nncols, maxcol;
    int nncols;
    int *buf;

    /* Copy out the data dimensions. */
    nncols = inpic->nncols;
    nncols = inpic->nncols;
    maxcol = nncols-1;

    /* Allocate a temporary buffer. */
    buf = (int *)malloc(nncols*sizeof(int));
    if (not buf) then {
        printf("Unable to allocate a buffer.");
        return(FALSE);
    }

    /* Make sure the images are the same size. */
    if (nncols != outpic->nncols or nncols != outpic->nncols) then {
        printf("Incommensurable images.");
        free(buf);
        return(FALSE);
    }

    /* Copy the data row by row. */
    for (nowrow = 0; nowrow < nncols; nowrow++) {
        getrow(inpic,nowrow,0,maxcol,buf);
        putrow(outpic,nowrow,0,maxcol,buf);
    }

    /* Free working storage. */
    free(buf);
end(Copy Data)
/* Copy the documentation, if any. */
outpic->doc = copydoc(inpic->doc);
outpic->writedoc = TRUE;

/* Close the picture files. */
closepic(inpic);
closepic(outpic);
/*
 ** document picname textstring
 ** Add documentation to a picture file.
 **
 ** TO MAKE THE PROGRAM:
 **
 ** cc -o document -O document.c -I/u/tb/include \ 
 ** /u/tb/lib/imglib.h /u/tb/lib/sublib.a -im
 **
 ** COMMENTS
 **
 ** This is a stripped-down program intended
 ** to be used as a template.
 **
 ** If you do not provide the picture name or the
 ** test string you will be asked for them.
 **
 ** A flag argument is needed so that the user can
 ** specify replacement of the existing text rather
 ** than appending.
 **
 ** Type document "??" to get a usage message.
 ** The quotes are necessary.
 **
 ** HISTORY
 **
 ** 06-Sep-83 Laws at SRI-IU
 ** Wrote initial version.
 */

#include "testbed.h"
#include "imglib.h"
#include "doclib.h"
#include "arglib.h"

/* External routines. */
char *malloc();
char *append();

main(argc,argv)
{

int argc;
list argv;

string filename;
PIC pic;
DOC indoc;
string addtext;
int addlen;
char *newtext;

/* Allow arguments to be omitted. */
openarg(argc,argv,0,"document pic");

Sep 12 10:35 1983
/* Open the picture file for read/write. */
filename = stringarg(-1, "by.img", "Input picture:");
pic = openpic(filename, 2);
if (not pic) then exit(ERROR);

/* Copy a pointer to the documentation structure. */
indoc = pic->doc;

/* Check for non-text documentation types. */
if ((indoc->doctype == NODOC) and (indoc->doctype != TEXTDOC)) then {
    printf("Cannot append to the current documentation type.\n");
    exit(ERROR);
}

/* Ask for text to be appended. */
addtext = stringarg(-2, NULL, "Additional text:");
addlen = strlen(addtext);

/* Create the new documentation record. */
newtext = append(indoc->text, indoc->textlen, addtext, addlen);
pic->doc = newdoc("TEXTDOC", newtext, indoc->textlen+addlen+1);
pic->writedoc = TRUE;

/* Close the file and write the documentation. */
closepic(pic);

/**
* static char *append(oldtext, oldlen, addtext, addlen)
* **
* ** Concatenate a text buffer, a linefeed, another text buffer, and a null.
* */

static char *append(oldtext, oldlen, addtext, addlen)
{
    char *oldtext;
    int oldlen;
    char *addtext;
    int addlen;

    char *newtext;

    /* Relocate the text string. */
    newtext = malloc(oldlen+addlen+2);
    strncpy(newtext, oldtext, oldlen);

    /* Concatenate a linefeed and the new text. */
    strncpy(newtext+oldlen, ",1);
    strncpy(newtext+oldlen+1, addtext, addlen);

    /* Add a terminal NULL, just to be friendly. */
    newtext[oldlen+addlen+1] = NULL;
document.c

/** Return the new text buffer. */
return(newtext);
}

...static

Dec 30 17:24 1983
/* Basic Test Program for the standalone Grinnell GMR275 routines. */

/* Andy Hanson Thu May 26 12:42:59 1983 */

#include <stdio.h>
#include "grinnell.h"

int ifrm; /* Pointer passed to all Grinnell frame routines */

open_bw()
{
    printf(" Allocating and displaying a frame.\n");
    /* Open the Grinnell and allocate a frame. */
    if (g_init(0) == G_GMRERROR) {printf("Unable to initialize\n"); exit(1);}
    if ((ifrm = g_allocfrm(G_HW,8)) == G_GMRERROR) exit(1);
    if (g_chgdisp(ifrm) == G_GMRERROR) exit(1);
    if (g_erfrm(ifrm) == G_GMRERROR) exit(1);
}

/* Allow color frame for color testing purposes */

open_color()
{
    printf(" Allocating and displaying a color frame.\n");
    if (ifrm >= 0)
        g_freefrm(ifrm); /* Close BW frame if existing */
    if ((ifrm = g_allocfrm(G_RGB,8)) == G_GMRERROR) exit(1);
    if (g_chgdisp(ifrm) == G_GMRERROR) exit(1);
    if (g_erfrm(ifrm) == G_GMRERROR) exit(1);
}

/* Routine for drawing simple lines on the Grinnell screen */

line(x1,y1,x2,y2)
    int x1,y1,x2,y2;
{
    g_linedraw(ifrm,y1,x1,y2,x2,G_SHEIGHT,G_SWIDTH);
}

/* Routine for tracking the cursor */

track_cursor()
{
    int flag;
    int returnval;
    x,y,old_x,old_y;
    printf("Entering KEYCUR, type \"q\" to exit.\n");
ttyreset();
    returnval = 0;
    flag = OK;
    old_x = 0;
    old_y = 0;
    while (flag == OK)
    {
        flag = g_keycur(0.0x0001,old_y,old_x,x,y,&returnval,0);  
        printf("Keycur returned %i, x = %i, y = %i, flag value = %i\n", 
            flag,x,y,returnval);
        g_linedraw(ifrm,old_y,old_x,x,y,G_SHEIGHT,G_SWIDTH);
        old_x = x;
        old_y = y;
    }
}
main()
{
    printf(" Grinnell GMR275 test system\n\n");

    /* Open Grinnell */
    printf("Performing basic self-tests; check overlay subchannels.\n");
    g_init(0);
    g_intest(1);
    g_intest(2);
    g_intest(3);
    g_intest(4);
    askbool("Ready to proceed to BW frame test?",TRUE);
    open_bw();
    line(0,0,511,511);
    line(0,0,0,511);
    line(0,0,511,0);
    line(0,511,511,0);
    line(0,511,511,511);
    line(511,511,511,0);
    track_cursor();
    askbool("Ready for color frame?",TRUE);
    open_color();
    g_setclbpix(ifrm,100,200,100,200,255,0,0);
    g_setclbpix(ifrm,200,300,200,300,0,255,0);
    g_setclbpix(ifrm,300,400,300,400,0,0,255);
    g_exit();
}
/**
 * showpic picname
 * Display a BW picture on the Grinnell.
 * TO MAKE THE PROGRAM:
 cc -o showpic -O showpic.c -I/tb/include \
 /tb/lib/imageLib.a /tb/lib/sublib.a -lm
 * COMMENTS
 * The screen is cleared and the picture is displayed
 * in the lower-left corner.
 * This is a simplified version of "show" which handles
 * only monochrome images, does no windowing, does not center
 * the display, and does very little error checking. For
 * a more elaborate example see show.c in /tb/src/show.
 * If you omit the picname argument, you will be
 * reminded with a usage message. If you specify an
 * invalid name, you will be asked for a replacement.
 * 
 * HISTORY
 * 
 * 17-Jan-83 Laws at SRI-IU
 * Changed bit.h to piclib.h, frmlib.h.
 * Moved external declarations to the include files.
 * 
 * 03-Nov-82 Laws at SRI-IU
 * Wrote initial version.
 */

#include "testbed.h"
#include "piclib.h"
#include "frmlib.h"

main(argc,argv)

int argc;
list argv;

Pic pic;
FRM frm;

/* Make sure an argument was supplied. */
openarg(argc,argv,1,"showpic picname");

/* Open the picture file. */
pic = openpic(argv[1]);

/* Open the default display. */
frm = openfrm(NULL,BWFRM);
showpic.c

/* Erase the screen. */
erasefrm(frm);
showfrm(frm);

/* Display the picture. */
showpic(pic,frm);

/* Close the picture (optional unless modified) */
closepic(pic);
}
transpose.c

/*
** transpose input output
**
** Transpose a picture file.
**
** TO MAKE THE PROGRAM:
** cc -o transpose transpose.c -I/u/tb/include \ 
**    /u/tb/lib/imagelib.a /u/tb/lib/sublib.a -lm
**
** COMMENTS
**
** This is a stripped-down program intended
** to be used as a template for writing picture
** transformations.
**
** The entire picture file is copied with the
** original header info and blocking factors.
** The access permissions of the original file
** are not preserved by this program.
**
** The copying is done pixel by pixel, which
** is rather slow. Getrow() and putcol() could
** be used instead.
**
** If you do not provide the input and output
** arguments, you will be asked for them. The
** input name is used as the default output
** name, which will simply copy the picture
** to itself.
**
** Type transpose "??" to get a usage message.
** The quotes are necessary.
**
** HISTORY
**
** 28-Sep-83 Laws at SRI-IU
**  Adapted from a previous copypic version.
*/

#include "testbed.h"
#include "imglib.h"
#include "arglib.h"

/* Transposition macros. */
#define outcol irow
#define outrow icol

main(argc,argv)
    int argc;
    list argv;
{
    string infile,outfile;
    PIC *input,*output;
    HDR output;
/* Allow arguments to be omitted. */
openarg(argc, argv, 0, "transpose inpic outpic");

/* Open the input picture file. */
infile = stringarg(-1, "bw.img", "Input picture");
inpic = openpic(infile);
if (not inpic) then exit(ERROR);

/* Make a transposed header with default blocking. */
outhdr = newhdr(inpic->nrows, inpic->ncols, inpic->pixelbits);

/* Open the transposed output picture file. */
outfile = stringarg(-2, infile, "Output picture");
outpic = newpic(outhdr, outfile);
if (not outpic) then exit(ERROR);

/* Copy the picture data */
begin(Copy Data)

  register int inrow, incol;
  register int ncols, nrows;

  /* Copy out the data dimensions. */
  ncols = inpic->ncols;
  nrows = inpic->nrows;

  /* Copy and transpose the data pixel by pixel. */
  for (inrow = 0; inrow < nrows; inrow++)
    for (incol = 0; incol < ncols; incol++)
      putpixel(outpic, outcol, outrow,
               getpixel(inpic, incol, inrow));

end(Copy Data)

/* Copy the documentation, if any. */
outpic->doc = copydoc(inpic->doc);
outpic->writedoc = TRUE;

/* Close the picture files. */
closepic(inpic);
closepic(outpic);