Report Number: WUCS-87-29

1987-01-01

The WUDMA Image Processing System

Authors: Andrew Laine, Steve Reichenbach, and Seymour Pollack

The WUDMA Image Processing System provides a framework that allows many image processing packages to function as the single system. It currently contains several packages that provide a powerful range of image processing tools for use in teaching and research. The WUDMA System overcomes the lack of standardization in image processing by providing bridges between diverse software packages and shielding the user from incompatibilities inherent in the software. As such, it may be considered as a paradigm from integrating packages in other application areas.

THE WUDMA IMAGE PROCESSING SYSTEM

Andrew Laine, Steve Reichenbach
and Seymour Pollack

WUCS-87-29

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Supported by the Defense Mapping Aeronautical Center Contract DMA 700-84-C0037.

Abstract

The WUDMA Image Processing System provides a framework that allows many image processing packages to function as a single system. It currently contains several packages that provide a powerful range of image processing tools for use in teaching and research. The WUDMA System overcomes the lack of standardization in image processing by providing bridges between diverse software packages and shielding the user from incompatabilities inherent in the software. As such, it may be considered as a paradigm for integrating packages in other application areas.

INTRODUCTION

The WUDMA Image Processing System is a large package of image processing commands [1]. It is comprehensive and flexible, containing tools that apply established image processing techniques as well as those of a more experimental nature. The package is worth examining not only because it is a useful image processing resource, but also because it integrates distinct software packages. Combining complementary packages is cheaper than creating new code to fill the gaps in a single package. The WUDMA System provides a uniquely complete resource for image processing and can serve as a model for integrating separate software packages.

The WUDMA Image Processing System was developed to meet the specialized needs of an intensive course of instruction in image processing offered by Washington University (WU) for selected technical employees of the Defense Mapping Agency (DMA). The fifteen-week Professional Development Program [2] extends the education of DMA technical personnel in line with the agency's increasing use of digital techniques for image processing and cartography [3]. Toward this end, it includes many topics found in the WU undergraduate and graduate computer science curricula, with special emphasis on image processing and cartographic databases.

The backgrounds of the students, the goals of the course, and the emphasis on laboratory experience and student projects place special demands on the image processing software system. To meet the curricular needs, the system has to be broad in scope; sophisticated and comprehensive, yet easily learned; and flexible enough to support student project research, development, and implementation. To accomplish this, the WUDMA System incorporates several software packages as subsystems and provides a uniform user-interface and an environment in which programs from distinct packages can operate as one system.

SOFTWARE ACQUISITION

With only three months until the first offering of the course, design and implementation of a new image processing package was not feasible. No commercial image processing packages could be found that met all the needs of the course. A survey of universities turned up many useful systems, but once again none met all of the needs of the course. Although each of these software packages offered useful algorithms that in many cases complemented those provided by other packages, they could not be merged because each was based on its own image formats, command language, and data types. Therefore, the initial offering of the course employed several distinct image processing systems. This was a valuable experience for those involved with the course in that it helped underscore what was and was not desirable in image processing systems.

These packages were selected in tandem with the hardware acquisition. The WUDMA Image Processing Laboratory is built around a VAX 750™ processor equipped with a floating-point accelerator and a second UNIBUS™ to accommodate direct memory access by a DeAnza IP8500™ image processor. The VAX now has eight megabytes of main memory and over 900 megabytes of high-speed disk storage. A tape drive provides long-term storage. The DeAnza IP8500 is a pipeline, matrix processor designed for high-speed image processing. The WUDMA IP8500 is equipped with six 512x512x8-bit refresh memory boards, a digital video processor, one video output controller, an alphanumeric generator, dual cursors, a trackball, and a nineteen-inch RGB monitor. A MicroVAX II™ satellite processor networked via Ethernet™ provides additional computing power. Image processing programs can be executed on the MicroVAX if the load on the VAX 750 justifies it. The lab contains four graphics workstations, each consisting of an alphanumeric terminal, a Vectrix 384™ high-speed graphics device, and a thirteen-inch RGB monitor. A self-contained, table-top camera provides picture-taking capability. The system is accessible via the LA-100™ console, six terminals, four lines on the local network, and four dial-in lines. Both DMA and WU preferred the UNIX™ operating system and the University of California, Berkeley, 4.2 UNIX has proven to be a flexible and productive implementation of this system.

Several image processing packages were acquired from outside sources for the first session of the course. The Picture Display System (PDS) developed by Anthony Reeves [4] and the Visual Shell (VSH) from the University of North Carolina [5] provided most of the fundamental image processing programs. The DeAnza Driver and Display package from PAR Corporation [6] provided the interface to the IP8500. Portions of other packages were used and additional programs were written to expand processing capabilities and to make the packages easier to use.

The VSH and PDS packages are similar in many ways. Both consist of UNIX command level programs written primarily in C and both store an image as a single file containing header information and pixel intensities. The VSH package has two attractive features. First, the intensity histogram is stored with the image, speeding many algorithms such as histogram equalization and intensity stretching. Second, the header and image are accessed via subroutine calls. This allows changing the header and file structure without altering the image processing programs. The image is accessed via calls that specify the limits of the desired sub-image. For example, one can access a two-dimensional 512x512 image by reading or writing a single row (e.g., first row = 0, last row = 0, first column = 0, last column = 511), a subimage (first row = 64, last row = 191, first column = 0, last column = 127), or the entire image (first row = 0, last row = 511, first column = 0, last column = 511). The input or output is handled by the subroutine.

Despite these features, those who have worked with both systems prefer the software model used by the PDS package. Most commands written at WU use the PDS subroutines, image formats, and data structures. The basic PDS pixel datatype is one byte and can therefore be displayed directly on the DeAnza and Vectrix devices. (The VSH package uses a two-byte short integer that has to be mapped to one byte for display.) The PDS package also allows UNIX pipes between programs.

The UNIX pipe enables output from one program to be used as input to another program as soon as it is generated [7]. Thus, PDS allows the sequence

```
trans -v if=inputimage | mean | threshold -t 128 of=outputimage
```

to transpose the image contained in the file *inputimage* about the vertical center-line, then perform a mean filter, and finally threshold at 128 and store the result in the file *outputimage*. The use of pipes can speed processing by running all three programs at once with each program reading input as it is needed and available.

The PDS programs access the header in the image file directly, reading it as a single data structure. Most of the PDS programs access the image directly, using C system calls rather than special subroutines. Changing the header or file structure could mean changing all of the programs, but direct access to the header and image has the advantage of simplicity.

SYSTEM INTEGRATION

Each of the individual packages provided algorithms lacking in the others, but having several software packages for image processing was inconvenient and inefficient. Figure 1 depicts the software environment prior to development of the WUDMA System. Users had to deal directly with several systems, each having different command-line conventions and image-file formats. Users had to be familiar with several processing packages and know the format of all of the image files used. Although format conversion programs were written, a processing sequence using several algorithms still might involve several explicitly invoked format conversions.

In response to the clear need for a unified software system, the Defense Mapping Agency Aeronautical Center authorized development of the WUDMA System [8]. Several objectives were defined:

- Image file formats should be invisible to the user. The user should not be responsible for any file format, header format, or image format specification or conversion.
- Command conventions and syntax should be uniform. Users should not have to learn different command languages for different commands.

- The system should be easily maintained. Most of the original software lacked *makefiles* (files specifying compiling, linking, and installation) for system regeneration [9] and none was under any version control system (such as the Revision Control System developed by Walter Tichy at Purdue University [10]).
- The system should provide complete on-line documentation. Most of the programs had manual pages, but some of the programs were poorly documented and more system documentation was needed. A single facility for all of the image processing documentation was needed.
- It should be easy to incorporate new programs into the system. User contributions are strongly encouraged throughout the course. Guidelines and tools for writing and integrating programs into the system were needed.

The project for development of a single image processing system was constrained by familiar factors: time, money, and personnel. The system had to be finished and working in three months with minimal cost and staffing. Three options were considered:

1. Design a new framework specifying command conventions, image file formats, header structures, data structures, and utilities. This new system could incorporate attractive features from the many individual packages that had been used or examined. This option would allow specification of a superior framework for future development, but the programs already in use would have to be rewritten or substantially revised.
2. Choose one of the frameworks already in use and convert programs from the other packages into that framework. By committing to one of the packages already in use, a great deal of recoding could be avoided. However, each of the packages has its own strengths and choosing one would mean incorporating only those strengths into the unified system.
3. Design a system to overlay all of the packages. This system would maintain the integrity of the individual packages and accomplish all necessary command interpretation and format conversion. The existing programs would not need revision and bug fixes and enhancements could be exchanged with the source institutions. However, this scheme would entail computational overhead. Execution of the underlying programs would be unchanged, but any command interpretation or format conversion would require additional computation.

The first option involved unnecessary work. Although any of the frameworks might be improved, it was not necessary to "reinvent the wheel." The cost of such an effort would not be worth the improvement it would yield. The second option was also rejected. Because the VSH programs did not access the image files directly, we explored the possibility of rewriting the access subroutines to use PDS files. However, format conversions were still necessary. For example, histograms are not contained in PDS files and would have to be generated. A more serious problem was that many of the VSH programs implicitly assumed sixteen-bit pixels. If the data type was changed to one byte, the computations would cause data overflow.

The third option met all of the design objectives. The system, not the user, would handle image formats. A command interpreter would hide command conventions of the different packages. System maintenance would be easier because versions would be compatible with the originating organizations. Existing documentation could be revised to reflect the unified system and combined into a single, on-line source. Contributors could select a subsystem with which they were comfortable and use it as a framework for new programs.

One of the primary considerations was interactive use, so the computational overhead of the integrating system was an important factor. The processing commands required from a few seconds to several minutes, depending on the size of the image and the complexity of the algorithm. Command interpretation and format conversion added from a few seconds, if little conversion was required, up to about twenty seconds if a great deal of conversion was needed (e.g. format change for a large image). We decided that this overhead was tolerable.

Figure 2 illustrates how the WUDMA System integrates many image processing packages. Though the underlying subsystems use different command conventions and file formats, the user sees a single system. A unified manual, available in print or online, documents the whole system. The organization of

the system provides a coherent software environment that is easily maintained and expanded.

The integrated software system uses C Shell programs or scripts. The C Shell command interpreter provides for variables, string and arithmetic operations, conditional execution, and loops [7]. The WUDMA C Shell programs execute in three steps. First, they translate the command-line specification of parameters and options from the global, system conventions to the specific, subsystem conventions. Second, they insure that input images are of the proper format, performing any necessary conversions. Third, they call the appropriate subsystem program, passing it the translated command line and the properly formatted files. Figure 3 traces this process. In this manner, the system uses image processing programs from several packages, but presents a uniform package to the user.

All documentation has been coalesced into a single document [1]. The WUDMA *Image Processing Software* manual is a valuable reference for new users, old hands, and system hackers alike. This manual is available both in print and online. Modeled on the UNIX manual, it contains pages describing the commands, system and library subroutines, special files, file formats and conventions, and system maintenance. Supplementary documents describe parts of the system in detail. The manual also documents the organization of the WUDMA System and provides instruction in its use. Online documentation is available via a command modeled on the UNIX *man* command. The documentation accompanying the original packages has been revised to reflect the integrated WUDMA System and to address users' concerns. Users send comments, criticisms, and bug discoveries via the UNIX *mail* facility to *imanual*, a pseudo-user account established to serve as the WUDMA System manager.

The UNIX *make* command, a tool for maintaining computer programs, has been useful in constructing a system that is easy to maintain and expand. This command uses instructions provided by *makefiles* to generate and install libraries and executable code. *Makefiles* specify such things as where source code, included files, and subroutines are located; how they are compiled and linked; and where the result is installed. To illustrate the utility of the *make* command, consider the situation where a subroutine is found to contain a bug. After the subroutine's source code is fixed, the system manager runs the *make* command. Using the *makefile*, the *make* command determines that the subroutine's source code has been revised and then recompiles and installs the subroutine in its library and all of the programs using it. The WUDMA System contains a network of *makefiles* so that all or parts can easily be regenerated.

The WUDMA System is easily managed because it is well organized. A single directory tree holds the entire system. One subdirectory contains all of the executable code, another all of the C Shell programs, another all of the subroutine libraries, and another all of the included files. A subtree, modeled on the UNIX */usr/man* tree, holds all of the documentation. Another subtree contains all of the source code.

New image processing packages can easily be added to the WUDMA System. Each subsystem maintains its identity and compatibility with the source institution. In fact, it is a simple matter to use a subsystem directly as if it were not part of the larger system. Because the respective frameworks of the subsystems are unchanged, it is easy to exchange code with any institution using any of the subsystems. New programs can be added to the WUDMA System via any of the subsystem frameworks. Prototype programs for each of the subsystems facilitate coding of new image processing programs. These prototype programs already contain the code that most image processing programs will need, such as calls necessary to parse arguments and open files for reading and writing; the user need only specify options and add the code for the particular application. A prototype manual page makes documentation of new programs easier.

Many new programs have been written at WU and incorporated into the system. These include general image processing routines executing on the VAX and programs using the processing capabilities of the DeAnza digital video processor. Commands have been written to interface with the display units for such tasks as reading and writing images, accessing intensity transformation and color-lookup tables, graphic display of histograms, and locator and trackball programs. The format conversion routines can also be accessed directly within the system.

CONCLUSIONS AND DIRECTIONS

The WUDMA System now contains more than 150 image processing commands. It has arithmetic, logical, bitwise, and neighborhood operators; edge detectors and region growers; commands to read and write image buffers, color-lookup tables, and intensity-transform tables; programs that translate, scale, rotate, zoom, roam, and locate; color transforms and histogram and intensity operators; Fourier transform programs; commands for file manipulation; graphics programs; and system help and documentation commands. Appendix 1 lists a few of the commands. The system has proven to be comprehensive and flexible. It includes basic algorithms and sophisticated tools for interactive image-processing research. The system is simple to learn, use, and expand.

There are few standards in the image processing community. There is no standard file format, set of subroutines, or image data structure. No package of commands or operations is complete. The WUDMA System solves this problem by providing a framework that integrates many packages into a single system. Until industry standards for image processing are established, the WUDMA System will provide a dynamic image processing environment.

Besides contributing substantially to the success of the Professional Development Program, the WUDMA Image Processing System is also being utilized by ongoing research projects at WU. Currently, we are expanding the WUDMA System by implementing additional algorithms on the high-speed DeAnza processor. By taking advantage of the special DeAnza architecture, algorithms can run in one tenth the time or less than on the VAX. Some commands run several minutes on the VAX, too long for interactive work. Frequently, work of this type is submitted to the UNIX system in background execution, which is effectively batch processing. The same algorithms on the DeAnza may require only a few seconds. As more algorithms are implemented on the DeAnza, this type of batch job can be avoided and a more interactive work environment achieved.

A new project at WU to develop a semi-automatic photo interpreter has suggested new directions for the WUDMA System. Plans are being made for utilization of the system by expert systems for feature extraction and feature verification. Proposals for a language for image processing are being studied. Such a language would provide a higher-level user interface allowing more efficient use of the system. Work implementing an image database system has also begun.

☐ The WUDMA System is available to non-profit institutions. A license agreement must be executed and there is a distribution fee. For more information write:

The WUDMA Image Processing System
Box 1045
Department of Computer Science
Washington University
St. Louis, MO 63130

☐ We would like to thank the many people associated with the WUDMA project. We would especially like to acknowledge the help of Barry Kalman, manager of the Professional Development Project. Our thanks also to the WUDMA teaching staff and students, DMA administrators working with the program, and Art Toga of the WU Dept. of Neurology.

REFERENCES

1. A. F. Laine and S. E. Reichenbach,*WUDMA Image Processing Software - User's Manual*, Technical Report WUCS-87-30, Dept. of Computer Science, Washington University, St. Louis, MO.

2. B. A. Kalman et al., "Defense Mapping Goes to School: An Educational Model for Automation" *Computer Graphics World*, Vol. 8, No. 12, December 1985, pp. 27-30.

3. M. B. Faintich, "Defense Mapping Undergoes a Digital Revolution" *Computer Graphics World*, Vol. 8, No. 6, June 1985, pp. 10-28.

4. A. P. Reeves, *The PPS-PDS Users' Manual*, Version 2.2, School of Electrical Engineering, Cornell University, Ithaca, NY.

5. J. Zimmerman et al., *V Shell Reference Manual*, Version 2, Dept. of Computer Science, University of North Carolina, Chapel Hill, NC.

6. R. Gray and P. Lentz, *A Package for Image Displays*, Release 1b, Par Technology Corp., New Hartford, NY.

7. W. Joy, "An Introduction to the C Shell" *UNIX User's Manual, Supplementary Documents*, 4.2 BSD, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley.

8. A. F. Laine and S. V. Pollack, *The Enhanced WUDMA Image Processing System*, Technical Report WUCS-85-01, Dept. of Computer Science, Washington University, St. Louis.

9. S. I. Feldman, "Make—A Program for Maintaining Computer Programs" *UNIX Programmer's Manual*, 7th ed., Vol. 2, Holt, Rinehart and Winston, New York, 1983.

10. W. F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System" *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Sept. 1982.

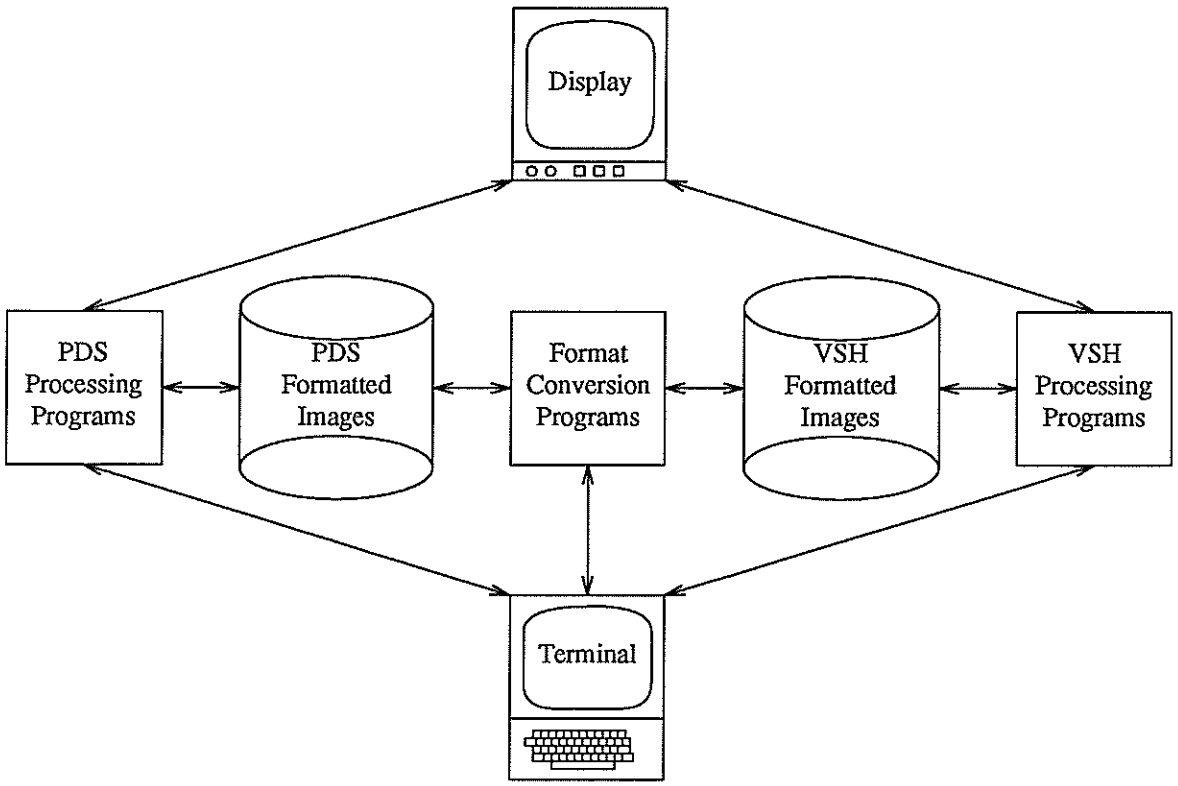

Figure 1. The image processing environment before the WUDMA System.

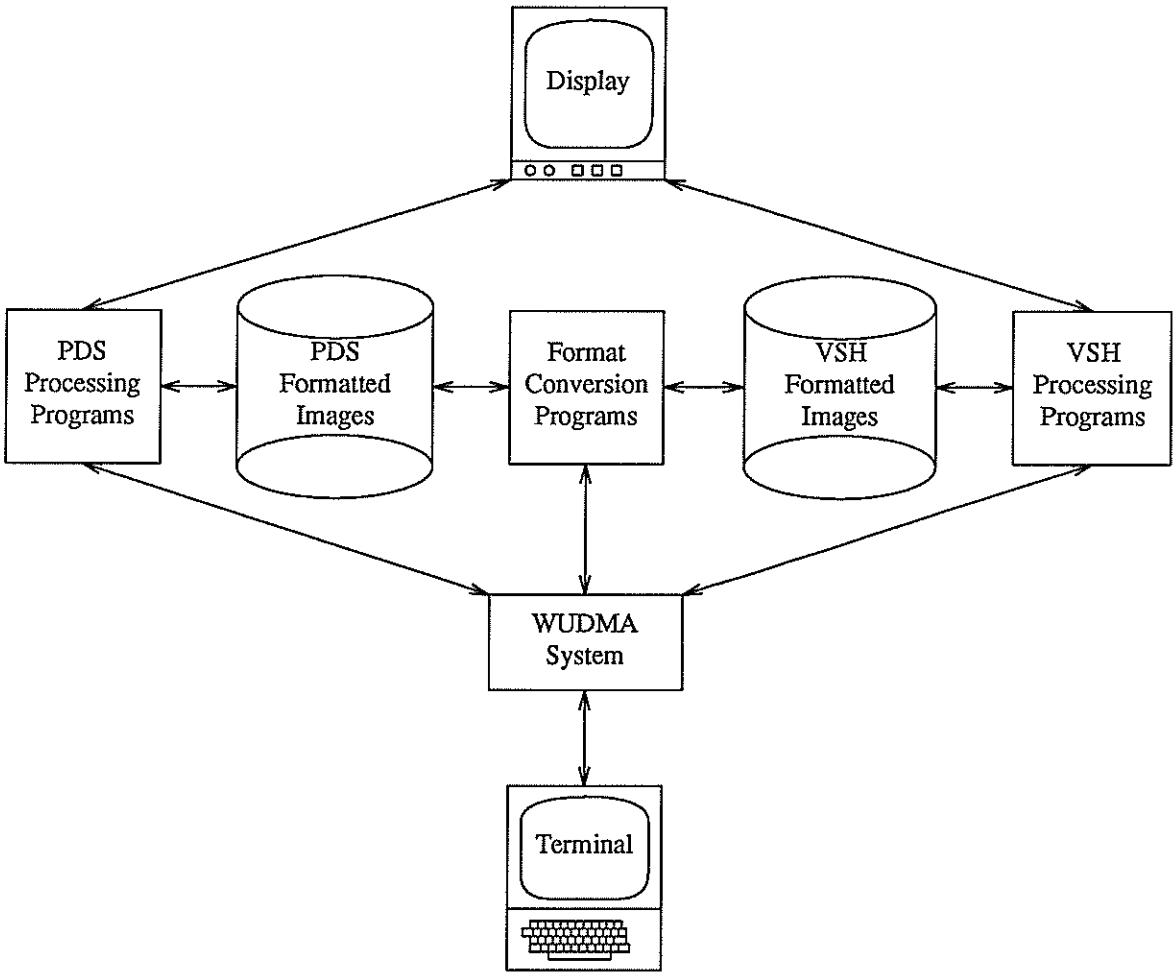

Figure 2. The WUDMA System image processing environment.

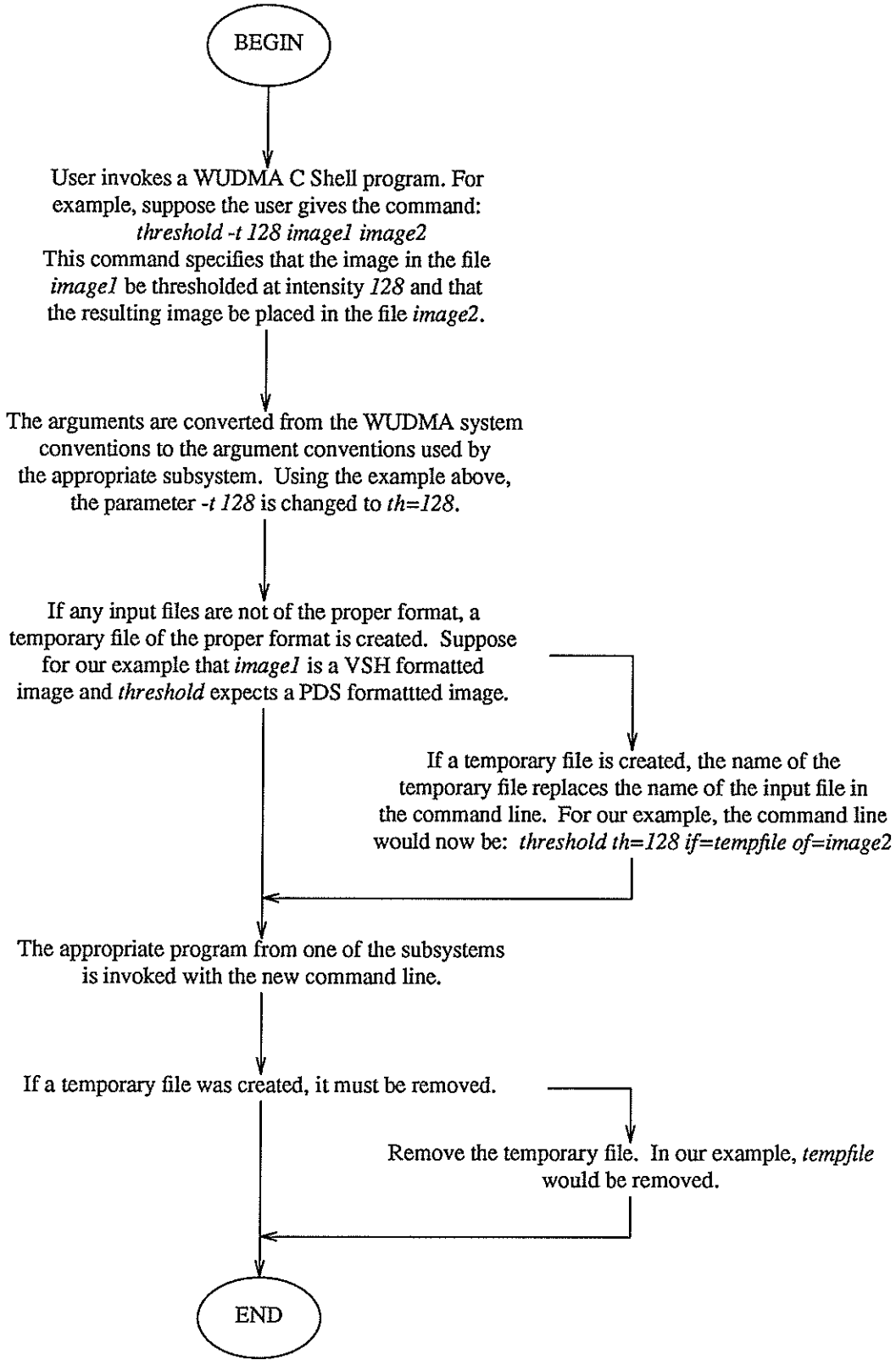

Figure 3. WUDMA C Shell execution.

Appendix 1: Some WUDMA Image Processing Commands

add . . . . . addition operator
ahe . . . . . adaptive histogram equalization
and . . . . . bytewise logical AND operator
bgrtohue . . . . . RGB and IHS space transformations
bnds . . . . . draw boundaries on DeAnza
butter . . . . . Butterworth low-pass filter
ccs . . . . . Robert's cross gradient edge-detector
clut . . . . . set Vectrix or DeAnza color lookup table
cmag . . . . . magnitude of two-channel, complex image
contour . . . . . plot contours
ctobw . . . . . color to black and white
dvpconv . . . . . DeAnza dvp spatial convolution with user-specified mask
dvpmax . . . . . local maximum
dvpmed . . . . . local median
dvpstdv . . . . . local standard deviation
embed . . . . . embed one image within another
extract . . . . . extract a subimage
fft . . . . . fast Fourier transform
gauss . . . . . add Gaussian noise
get_cur . . . . . DeAnza trackball locator
greymap . . . . . grey-scale intensity mapping
header . . . . . write description of an image to stdout
hgram . . . . . graphic intensity histogram
histeq . . . . . intensity histogram equalization
hough_circle . . . . . Hough curve-detector
iman . . . . . find image manual information by keywords
invert . . . . . ones complement
itt . . . . . read/write itt tables from DeAnza
kirsch . . . . . Kirsch edge-detector
loc . . . . . Vectrix locator
meddev . . . . . median deviation from the local median
motion . . . . . optical flow calculator
mse . . . . . mean squared error
mul . . . . . multiplication operator
pscale . . . . . transform and scale intensities
relax . . . . . Rosenfeld relaxation
reshape . . . . . increase or decrease size by bilinear interpolation
retina . . . . . Marr-Hildreth edge-detector
rotate . . . . . rotate an image
rsmooth . . . . . smooths regions while maintaining edges
sobelp . . . . . Sobel edge-detector (PDS)
threshold . . . . . intensity threshold
trans . . . . . diagonal, horizontal, or vertical transpose
unsharp . . . . . unsharp masking filter
vsfmean . . . . . variable shape mean filter
vxlut . . . . . load Vectrix color look-up-table
wfilter . . . . . Wiener filter variant
zcat . . . . . concatenate images into multichannel image
zextract . . . . . extract channel(s) from a multi-channel file
zr . . . . . zoom and roam on DeAnza

Appendix 2: A Sample C Shell Program

```
#
#       csh program to threshold an image producing a binary image
#
onintr cleanup
set path = ( /usr/image/bin /bin /usr/bin )
#       calls PDS threshold program
set cmd=(threshold)
#
#       options
#
while ($#argv > 2)
        if ("$argv[1]" =~ "-t") then
                if ($#argv < 2) goto usage
                set cmd=($cmd th=$argv[2])
                shift argv
                shift argv
        else
                echo 'unrecognized option' $argv[1]
                goto usage
        endif
end
#
#       input image, output image
#
if ($#argv != 2) goto usage
if { isapds $argv[1] } then
        set cmd=($cmd if=$argv[1])
else
        if (! { vshpds $argv[1] /tmp/$user.pds.$$ } ) exit(1)
        set cmd=($cmd if=/tmp/$user.pds.$$)
endif
set cmd=($cmd of=$argv[2])
#
#       execute
#
$cmd
#
#       cleanup and terminate
#
cleanup:
        if ( -e /tmp/$user.pds.$$) rm /tmp/$user.pds.$$
        exit(0)
usage:
        echo 'usage:  threshold [-t N] if of'
        exit(1)
```