Development of a Prototype Spatial Information Processing System for Hydrologic Research

Dr. Jayanta K. Sircar

November, 1991

Prepared for
Goddard Space Flight Center
Greenbelt, MD 20771

Contract NAG5-1466

Development of a Prototype Spatial Information Processing System for Hydrologic Research

Dr. Jayanta K. Sircar

November, 1991

Prepared for
Goddard Space Flight Center
Greenbelt, MD 20771

Contract NAG5-1466

Contents

Introduction

Significant advances have been made in the last decade in the areas of Geographic Information Systems (GIS) and spatial data analysis technology, both in hardware and software. Science user requirements are so problem-specific that currently no single system can satisfy all of the needs. The work presented here forms part of a conceptual framework for an all-encompassing science-user workstation system. While definition and development of the system as a whole will take several years, it is intended that small scale projects such as the current work will address some of the more short term needs. Such projects can provide a quick mechanism to integrate tools into the workstation environment forming a larger, more complete hydrologic analysis platform.

This report describes two components that are very important to the practical use of remote sensing and digital map data in hydrology. Part I of this report describes a graph-theoretic technique to rasterize elevation contour maps, while part II describes a system to manipulate synthetic aperture radar (SAR) data files and extract soil moisture content. The principal investigator for this project was Dr. J. K. Sircar with research assistance provided by Mr. Russell Fink.

Traditionally, the hydrologist uses topographic data, which allows for time-area methods of devising runoff models on watersheds. Such data is rapidly becoming available on digital media; however, at the present time, much of it still exists on paper and there is no foreseeable date at which most of it will be digitized. Consequently, the hydrologic workstation concept suffers because of the unavailability of data in the digital form. Scanning technology has improved over the last decade to the point that converting planar contour data into digital form is now feasible on the microcomputer level. Thus, the workstation analysis of topographic maps requires a tool that is able to convert paper map data into digital form.

Recent advances in the application of Synthetic Aperture Radar (SAR) to the detection of soil moisture content has added a new type of data to the field of hydrology. Currently, NASA's Jet Propulsion Laboratory (JPL) maintains a comprehensive collection of aircraft-collected SAR data, and plans to expand this collection by offering spacecraft-acquired data. SAR data is stored in a format that must be decompressed in order to be analyzed. A tool for decompressing as well as extracting soil moisture data from SAR data would need to be included in the hypothetical hydrologic workstation to make ground moisture data available to the hydrologist.

The first of the two hydrologic tools that will be of interest in the workstation determines the time-area curve and drainage isochrons using digitized topographic

data. A system of desktop microcomputer programs is presented in section one, and accepts raster digitizations of topographic maps and channel delineations and produces the time to drain as well as the S-hydrograph of the watersheds.

The second program provides a mechanism for decompressing SAR scenes into tractable data files containing information on soil saturation, and is presented in section two. This section includes code that can be used for a theoretical workstation system, and thus includes comprehensive operating and installation details.

Both of these projects demonstrate the concept of modular design in the ideal workstation in that the science user has available many different smaller tools that share a larger system. With the continued bottom-up development of such systems, the short term needs of the science user can be met now while progress is made toward the larger, fully integrated platform.

Part I: Computation of Time-Area Curves

Engineers engaged in hydrologic modeling and simulation recognize the considerable effort required for manually determining key terrain related information, e.g., overland flow lengths, slopes, channel velocities. In most cases, the required information is derived from available maps. With recent developments in the use of remotely sensed imagery, electronic data capture technology for map digitization, and the means to efficiently manipulate digital terrain data using a Geographic Information Systems (GIS) framework, the task of hydrologic process simulation has become significantly easier (Sircar, 1986; Van Blargan, 1990).

However, the modest investments required for data acquisition and processing to implement these methods limit their applicability to large, regional scale studies. The "small user", engaged more frequently in projects over relatively small, isolated areas can rarely afford the sophisticated data acquisition and computing systems widely in use by the larger, more "funding rich" organizations. Consequently, a significant number of small to medium sized engineering firms continue to rely on tedious and time consuming manual procedures.

Meanwhile, the use of microcomputers and desktop raster scanning equipment is gradually gaining prominence as a standard environment to support routine office productivity, even within many of these resources for developing tools that will remove some of the inefficiencies in routine hydrologic simulation. A valuable contribution towards such a task is the experience and technical concepts gained form past developments in remote sensing based GIS technology-- concepts that provide both a basis, and a framework, for the search for cartographic data analysis.

The overall objectives of the present research are to:

1) improve the efficiency in cartographic map analysis and spatially distributed hydrologic simulation, in particular, for small watershed studies;
2) economize the use of available hardware and software technology, and
3) extend the use of GIS and remote sensing technology by small to medium sized engineering firms for use in routine small scale hydrology.

The practical utility of any new technique is best understood through a realistic engineering application. Recognizing that a popular approach for hydrologic simulation in many routine engineering applications is the routing of a time-area histogram (Sircar, 1986), the success of the developed strategy is demonstrated in a test application. Key to the demonstration is the integration of a set of GIS based processing techniques to provide the spatial boundaries of the time-area isochrons.

Test Data Set

To demonstrate the capability of the proposed methodology and the associated computer based programs, a small watershed was selected from a list of experimental watersheds published by the Agricultural Research Service (ARS) of the United States Department of Agriculture (USDA) (ARS, USDA, 1964). The watershed is located in Treynor, Iowa and has drainage area of approximately 64 acres. Slopes in the area vary from near zero to approximately 19 percent. The predominant landuse in the watershed is cultivated contoured corn.

A strategy to Generate Time-Area Curves

The overall structure of the proposed strategy is partitioned into a series of tasks. Each of the tasks are implemented through a number of software modules. Fig. 1 illustrates the sequence of the major tasks associated with the development of the semi-automated digital approach. The functional descriptions of each of the principal components are described below.

Watershed Terrain Data Acquisition

Data Structure The more popular methods in GIS based hydrologic modeling use a grid-cell data structure to store and represent terrain information (Sicar, 1988). In most practical applications, the

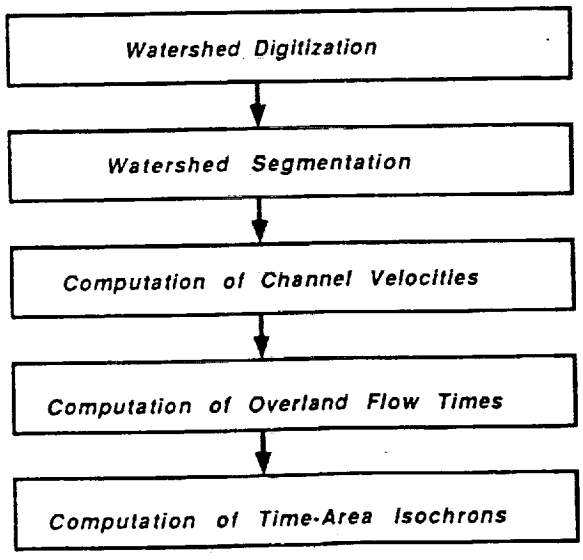

FIG. 1. A structure for Automating the Computation of Time-Area Isochrons in Small Watershed

study area is divided into a grid-matrix, with each grid cell (pixel) having a unique value for each physiographic property. In typical surface flow simulations, as in the application used in this study, the excess runoff for each pixel in the watershed is routed to the appropriate channel over neighboring pixels, and then along the channel to some specified point of interest where the hydrographic is produced.

Hardware/Software The present research is based on a standard off-the-shelf Macintosh coupled to a commercially available raster scanner. While most of the GIS based hydrologic applications were written for this specific study, a public domain Image Display and Processing Package (IMAGE, 1990; available from the National Technical Information Service) provided the principal image handling tool for image display and manipulation on the Macintosh screen.

Data Acquisition The principal data components required for hydrologic simulation of surface flow velocities are land use, elevations, channel locations, and watershed boundaries. Using a grid-cell, or "pixel" data format, Ragan (1991) demonstrated an efficient procedure to convert landuse maps created from digital raster files. The major problem however is posed by the need to quickly create the required elevation information.

A common scenario routinely used by the practicing engineer is a map or a part of a map on "hard copy" paper showing contour lines and channel reached. Using a raster coupled to a microcomputer, the "black and white" hard copy is scanned into a digital raster format file. Using simple image processing and image

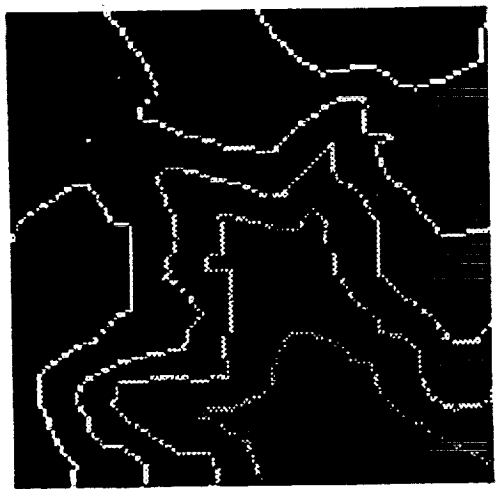

Fig. 2. Raster Scanned Image of a Contour Line Map of the Treynor Watershed

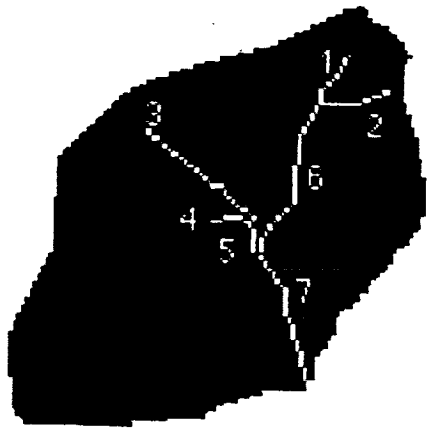

Fig. 3. Scanned Image of watershed boundary

editing tools, (Sircar and Cebrian, 1986), the scan digitized map (fig. 2) is processed to generate digital overlays of the watershed boundary and the channel lines (Figs. 3 and 4).

The output scanned image (fig. 2) is a "dumb" image without any associated attribute of elevation. The conversion from a "dumb" image into an image with contour lines labelled with appropriate contour values is accomplished by using a program developed using a semi-automatic labeling technique outlines in detail by Sircar and Cebrian (1991). The labelled contour image data, the basin boundary image, and the channel location image are input, within a GIS framework, to a program that computes the Digital Elevation Model (DEM) for the scanned watershed. Fig. 5 is an example of the output DEM for the contour image displayed in Fig. 2.

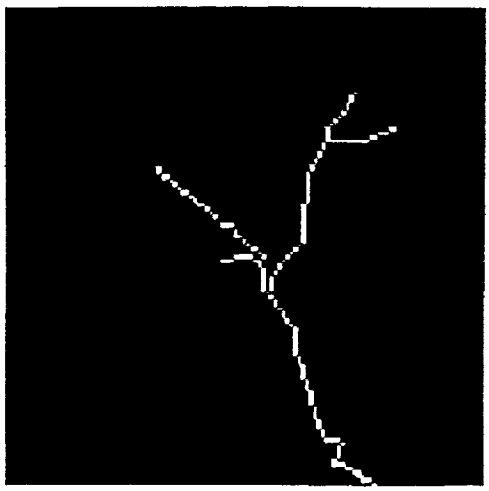

Fig. 4. Image of Channel Lines.

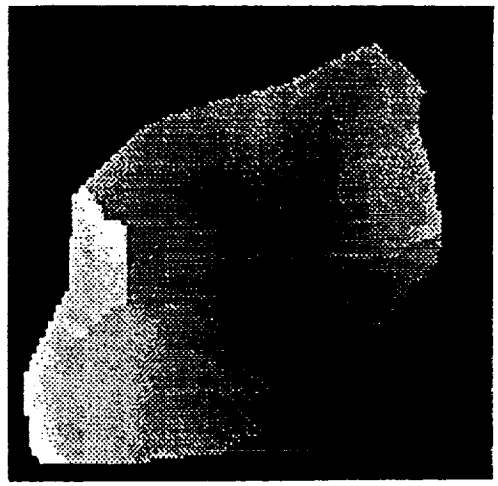

Fig. 5. Gray-Shaded Display of Output DEM for the Treynor Basin

Watershed Segmentation

The channel geometry of fig. 4, and the generated DEM shown in fig. 6, is next input to a program that segments the overall basin into a series of sub-basins, with each lower order sub-basin draining into the next higher order sub-basins at the stream junctions. The program also allows the user to interactively point to a screen display of the channel geometry, point to outfall pixels of interest along the channel system, and generate the corresponding upland sub-basin. Simple slope-aspect relationships between pixels are used to link higher to lower pixels to

generate a flow direction. A set of heuristics (Sircar, 1986) are used to guide flow directions when anomalous elevation peaks, flat pixels or elevation sinks are encountered in the direction of flow. Fig. 6 shows the result of watershed segmentation by the successive accumulation of lower to higher pixels beginning from stream junctions.

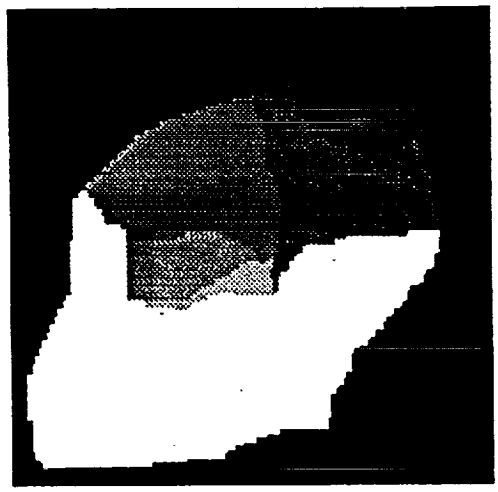

Fig. 6. Segmentation of Treynor Watershed into Subbasins upstream of Stream Junctions

Estimation of Channel-Flow Velocities

The prerequisites for computing the time-area isochrons are first, to determine velocities and times of flow along channel pixels, and second, velocities and times of overland flow. Sircar (1986) demonstrated an efficient technique to correlate channel sizes and discharge characteristics to the drainage area at any location. In the present study, results of watershed segmentation from the previous step are used to compute the drainage area for each of the reach lengths within the basin. The computation of estimates for the flow velocities along the channel reaches were based on a series of geometric relations derived by Dunne and Leopold (1978) and a set of sensitivity analysis performed by Helwa (1983).

The approach by Sicar (1986), incorporated in this research, uses the pixel based 3-dimensional flow conveyance linkages inferred from the DEM to i) estimate upland drainage area, slopes, widths, and lengths of sub-basins. and ii) provide estimates of the channel cross-sections, average channel slopes, and average channel velocities at specified point along a channel. Table I is an example of the type of intermediate output showing several channel/basin characteristics generated by the watershed segmentation module using the channel geometry, basin boundary and DEM shown in figs. 3, 4, and 5. Once the velocities of flow along the channel pixels are determined, the developed program computes for each pixel

along the channel, the time flow. In the case of the present research, the outfall location is indicated by the letter 'O' in fig. 4.

TABLE 1. Channel Geometry Computed using Geomorphic Relationships

*Chan. Seg #	cr-area[1]	depth[2]	radius[3]	dr-area[4]	slope[5]	velocity[6]
1	0.4690	0.2475	0.1095	0.0046	0.0730	3.0482
2	0.2741	0.1950	0.0856	0.0021	0.0464	2.0614
3	1.0078	0.3476	0.1552	0.0138	0.0454	3.0372
4	1.0967	0.3609	0.1613	0.0156	0.0341	2.7011
5	0.4919	0.2528	0.1119	0.0050	0.0126	1.2850
6	0.2384	0.1833	0.0803	0.0018	0.0343	1.6983
7	2.3830	0.5093	0.2297	0.0475	0.0250	2.9306

[1]*cr-area*: Cross-Sectional Area (sq-ft)
[2]*depth*: Bankfull Depth (ft)
[3]*radius*: Manning's Hydraulic Radius (ft)
[4]*dr-area*: Upland Drainage Area for Specified Channel Reach
[5]*slope*: Average Slope of Channel Reach (%)
[6]*velocity*: Bankfull Velocity at Reach Outfall (assumed constant for entire reach) (ft/sec)

**See figure 3 for reference to channel segment numbers*

Computation of Overland Flow Times

The estimation of overland flow times is a three-step process:

1) identify the pixels that constitute a flow line draining into an outfall pixel along the stream, the flow line for every pixel being defined by the gradients of the flow network;
2) determine the time of flow for every pixel based on its corresponding path length along the flow direction and slope, and
3) compute the total flow time across any overland flow line as the sum of individual flow times to the basin outfall for each pixel.

Taking advantage of GIS framework of accessing several data planes concurrently, a program was written to compute overland flow velocities at each pixel. The methodology to estimate overland flow velocities was based on a set of curves developed by SCS and is presented in detail by McCuen (1982). In equation

form, the relationship of velocity (V) to pixel slope (S) as developed in the SCS method is of the form:

$$V = a \cdot S^b$$

in which 'a' and 'b' are coefficients with values taken form McCuen (1983) for varying types of landuse.

In the present case, since the landuse in the test area, as mentioned previously in the section describing the test data set, was uniformly cultivated corn, the values of 'a' and 'b' for Eq. 1 selected from Table 2 were '0.5' and '0.5' for all pixels in the watershed. This alleviated the need, in the given example, for the creation of a separate digital, pixel based, landuse data base. The overland flow model used in the present application may, however, be replaced by any model of user choice.

Computation of the Time-Area Isochrons and the Time-Area Curve

The output from the previous steps are a pixel image containing the times of flow to the outfall for each channel pixel and an image with the times to flow for each channel pixel. The results from these two image planes are combined into a single image plane (Fig. 7) in which the value at each pixel is the flow time in seconds for each pixel in the watershed. To provide the user with a meaningful visualization of the results of flow time computation, the flow times have been grouped in 5 minute intervals. The boundary of each of the time intervals implies the corresponding 5 minute time-isochron. The corresponding time-area curve (Fig. 8) is constructed from the histogram of the number of pixels within each 5-minute time interval.

One expects the time-area curve to be in the form of an 'S'. As seen in Fig. 8. the digitally simulated time-area curve does indeed conform to the classical 'S' curve concept. The time required to implement the digital approach to the example shown in the study was approximately 40 minutes.

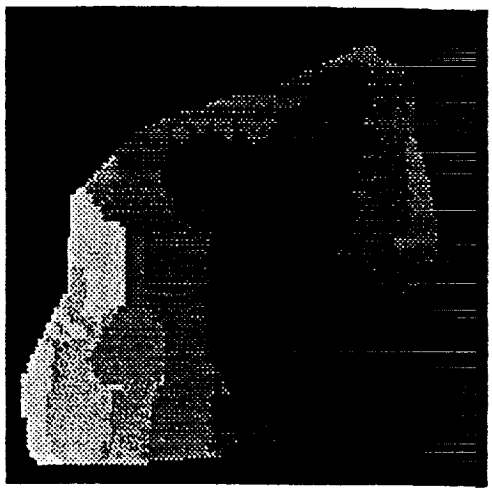

Fig. 7. A Gray Shaded Display of the 5-minute Time-Area Isochrons in the Treynor Watershed.

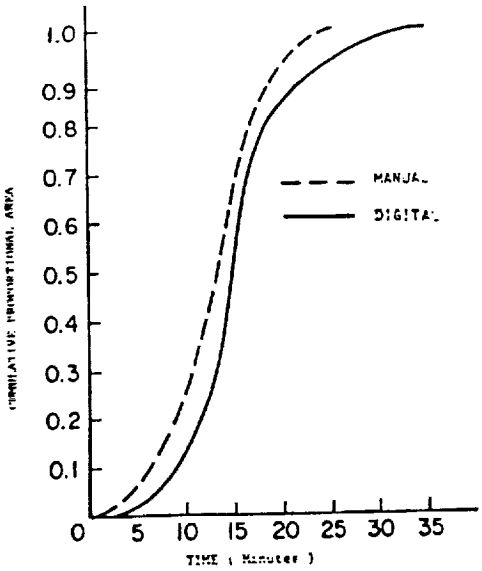

Fig. 8. Time-Area Curve for the Treynor Watershed.

Conclusion

The close agreement between the digitally simulated curve and the corresponding manually determined curve, particularly in shape, confirms the strength of the system for use as a practical engineering tool. Compared to the 5 or 6 hours of time required to compute the time-area curve for a watershed even as small as that selected here, the digital approach is significantly better. An added advantage of the developed approach is the ability to respond to very small and subtle variations in the way in which we per-pixel physiographic and landuse characteristics are disposed. The pixel based system therefore indicates a promising strategy to evaluate the spatial variability in watersheds once appropriate overland flow models are incorporated.

References

Dunne, T., and Leopold, L.B., 1978, Water in Environmental Planning. W.H. Freeman & Co.

Helwa, M.F., 1983, "The Channel Network in Hydrologic Simulation: Improved Modeling and Evaluation of Significance," Ph.D. Dissertation, Dept. of Civil Engrg., University of Maryland, College Park, MD.

McCuen, R.H., 1982, A Guide to Hydrologic Analysis Using SCS Methods, Prentice-Hall, Englewood Cliffs, NJ.

Sircar, J.K., 1986, Computer Aided Watershed Segmentation of Spatially Distributed Hydrologic Modeling. Ph.D. Dissertation, Dept. of Civil Engrg., University of Maryland. College Park, MD.

Sircar, J.K. and Cebrian, J., 1986, "Application of Image Processing Techniques to the Automatic Labeling of Raster Digitized Contour Maps", Proceedings. Second International Symposium on Spatial Data Handling, July 5-10, Seattle, Washington, pp. 171-184.

Sircar, J.K., 1987, "Definition and Testing of the Hydrologic Component of the Pilot Land Data System", Report on NASA-University of Maryland Project, Grant No. NAG5-512.

Sircar, J.K., 1988, "Linking Spatial/Geographic Information Processing Capabilities to NASA's Land Analysis System", Report on NASA-GSFC Contract No. S-79638-D.

Sircar, J.K. and Cebrian, J., 1991, "An Automated Approach for Labeling Raster Digitized Contour Maps", Journal of The Amer..Soc. of .Ph. & Rem. Sensing. In Press.

USDA, Agricultural Research Service, 1964, "Hydrologic Data for Experimental Agricultural watersheds in the United States," Misc. Publ. No. 1194.

VanBlargan, E.J., Ragan, R.M., and Schaake, J.C., 1990, "Hydrologic Geographic Information Systems," Geographic Information Systems 1990: Transportation Research Record, Trans. Res. Board.

Part II: Analysis of Synthetic Aperture Radar Data

Dubois, et.al., developed a method of radar scene compression, which led to the efficient storage of large 3-band scenes. The method described generates Compressed Stokes Matrix (CSM) data from 3-band radar images; using certain assumptions about the data and the Stokes Matrix representation of the reflected radar intensities, the compression method effectively reduces 64 bytes of data to a 10 byte unit, resulting in an efficient compression ratio. Since the development of this compression technique, all 3-band radar data available at JPL are stored in this CSM form.

Included with the Dubois paper is some VAX FORTRAN code designed to implement the reversal of the compression equations; however, there are problems with its efficiency and portability. It is also a VAX dependent version that would require some conversion to run on other platforms. These problems require knowledge about the Stokes Matrix, and thus will be examined in greater detail in section 1 of this report.

A second piece of software to analyze CSM data, developed for the Apple Macintosh, is available from JPL. This package is called MacMultiview [Norikane, 1989]. There are several problems with this system, the primary one being the reliance on the use of the Macintosh windowing environment and thus is not portable to other operating environments. This dependency makes analysis or extension of the code difficult, since the windowing system makes use of an unorthodox flow of program logic. The Macintosh code is, however, easier to implement than the Dubois code in that it provides a pleasant user interface.

The general lack of flexibility and readability of both MacMultiview and the Dubois program created the need for a simple decompression code to be developed for a portable environment. For example, the Hydrologic Sciences Branch recently purchased a color Silicon Graphics workstation that runs the UNIX operating system; because of its use of UNIX, neither of the existing systems are able to operate on this machine. We have chosen to implement C code under the UNIX environment, and have removed specific dependencies on UNIX as much as possible to allow portability to other operating environments.

Objectives

The current task is to create and verify an efficient, portable C program to decompress Stokes Matrix Radar image files for the analysis of polarized radar reflection intensities of hydrologic phenomena. Specifically, the computer program developed here is a simple decompression algorithm to decompress the 10-byte CSM data and generate a 4-byte real per-pixel total power value.

Organization of Report

This report is organized in the following way: section 1 of this report describes the program, its features and use; section 2 presents an automated verification of the CSM program; section 3 contains compilation and execution instructions; and the appendices contain the source code listings for the verification method and the C programs.

Section 1: A Program to Decompress SAR Data

The JPL representation of SAR scenes relies on a matrix representation of received power, known as the Stokes Matrix. The Stokes Matrix (SM) is a 4 by 4 matrix (16 elements) of real data corresponding to a received radar scattering matrix. The C program tp.c presented below implements the Dubois algorithm to compute total power. The SM requires 16 elements of real storage, or 64 bytes; for three bands of radar and a typical sample size of 1024 x 750, the storage requirements are fairly large. Reduction of the scene size is possible, however, because of some inherent mathematical properties of the SM. First, the SM is a symmetric matrix; thus, only the upper (or lower) triangular elements are unique. Further, one of the main diagonal elements is a sum of the other three, yielding only 9 distinct elements in the matrix. Another useful property of the SM is that the first element is the largest element of the matrix, and thus the other elements can be normalized by this first one to reduce the range of data values and thus reduce the size of the storage needed for each element. Finally, the whole radar scene can be normalized by a general scale factor, which could reduce the range of received intensities allowing for a smaller and more precise storage model.

These properties of the SM data form the basis for a data compression scheme. Dubois, et. al. have developed such formulas for storing the SM data in a 10-byte record as distributed in the JPL data sets. Decompressing the 10-byte record and extracting total power proceeds in the following way. First, the element $\hat{F}_{11}$ (the largest element of the SM) is obtained from the first two bytes of the record and then used to rescale the other 8 elements. Next, the SM is vectorized through multiplication by the transmit and receive vectors corresponding to the desired polarization angles. Once the 9-element real number SM is vectorized, the total power is obtained by summing the elements of the SM. A detailed analysis of how the 10-byte record is used to reconstruct the 64-byte SM is now presented.

The exact replication of the SM elements

$$\begin{matrix} F_{11} & F_{12} & F_{13} & F_{14} \\ F_{21} & F_{22} & F_{23} & F_{24} \\ F_{31} & F_{32} & F_{33} & F_{34} \\ F_{41} & F_{42} & F_{43} & F_{44} \end{matrix}$$

from the 10-byte CSM data is performed through the following formulas; the X_{ij} are obtained as an intermediate step:

$$X_{12} = \frac{byte(3)}{127.0}$$ [1]

$$X_{13} = \left[\frac{byte(4)}{127.0}\right]^2 \cdot sign[byte(4)]$$

$$X_{14} = \left[\frac{byte(5)}{127.0}\right]^2 \cdot sign[byte(5)]$$

$$X_{23} = \left[\frac{byte(6)}{127.0}\right]^2 \cdot sign[byte(6)]$$

$$X_{24} = \left[\frac{byte(7)}{127.0}\right]^2 \cdot sign[byte(7)]$$

$$X_{33} = \frac{byte(8)}{127.0}$$

$$X_{34} = \frac{byte(9)}{127.0}$$

$$X_{44} = \frac{byte(10)}{127.0}$$

where $byte(i)$ represents the i^{th} byte of the 10-byte CSM record and $sign(x) = \frac{|x|}{x}$. Next, F_{11} is computed with

$$\hat{F}_{11} = \left[\frac{byte(2)}{254.0} + 1.5\right] * 2^{byte(1)}$$

where $byte(1)$ represents the exponent and $byte(2)$ the mantissa of $\hat{F}_{11}$. Computing the other elements F_{ij} is done by multiplying X_{ij} by $\hat{F}_{11}$, excluding

[1] note that Dubois, et.al. shows this formula to be $byte(3) = INT(127 * X_{13})$ in the text; after reading their FORTRAN code, it is evident that X_{12} was erroneously listed as X_{13}.

element F_{22}:

$$F_{12}=F_{21}=\hat{F}_{11}*X_{12}$$

$$F_{13}=F_{31}=\hat{F}_{11}*X_{13}$$

$$\vdots$$

$$F_{44}=\hat{F}_{11}*X_{44}$$

Finally, F_{22} is computed as

$$F_{22}=F_{11}-F_{33}-F_{44}$$

After the Stokes Matrix is decompressed, its elements are then multiplied by gen fac to create the actual Stokes Matrix. The elements of the actual Stokes Matrix are then transformed by G_r and G_t in the following way to obtain the total power:

$$P=\vec{G}_r^{\,T}\cdot F\cdot\vec{G}_t$$

Implementing a direct, straight-forward translation of the above decompression method yields simple, small, yet inefficient computer code, as is presented in Dubois. The Dubois code provides a way of decompressing a CSM file into total power, but because of various problems, this code is not suited to a general application.

Code Enhancements

The primary problem with the code is its inefficiency. Certain basic properties of the Stokes Matrix (SM) are ignored in order to present a clear example of a decompression technique, resulting in redundant calculations and a slow decompression of data.

A concept is presented here which allows for some speedup of the process. The concept suggested is the use of storing all possible values of $\hat{F}_{11}$ in a two-dimensional array XXX, from which a given combination of values for byte(1) and byte(2) can be used to dereference the corresponding value of $\hat{F}_{11}$. By precomputing predicted values of an equation, certain calculations can be replaced with array accesses, resulting in a run-time speedup.

The inefficiency of the FORTRAN code comes not from the use of such a method, but from the limited scope of the method as the authors apply it. The program presented in this report extends this basic lookup cost-saving measure to include all aspects of the Stokes Matrix decompression process, not restricting its use to the computation of $\hat{F}_{11}$.

By employing extensive use of array lookup, disk buffering techniques, and only decompressing unique elements of the symmetrical Stokes Matrix, tp.c manages to perform the task in slightly less than 1 minute 30 seconds on a Sparcstation II, proving the effectiveness of these speedups in data decompression. This contrasts significantly to the reported times from the VAX and Macintosh versions.

Tp.c has other advantages over the FORTRAN code that make it better for a practical application of decompression. One such advantage is its portability -- the code can be transferred directly to a wide variety of operating environments with simple recompilation. Being separated from the task of image grayscaling, another advantage is that the design is simple, easy to read and modify. It also employs very general procedures that will allow for the extraction of data from changing header types for future applications.

Presented below is a description of the code to compute the per-pixel total power from a CSM data file. The code has been omitted in this section to maintain brevity; however, a complete copy of the code appears in the appendix.

Description of Code

One function of the decompression code is to interpret and extract information from the header. To be fully flexible, the code is able to search through the header looking for specific ASCII strings, and retrieve data values stored nearby. The header is composed of 50-byte records; the leftmost part of a given record contains text describing the data value that exists in the rightmost part. The strategy employed to extract information is to scan the header fields for known strings, and return the rightmost portion of the record matching the strings searched for as numeric data. Extraction of header data takes place in the function **void prepfiles()**. This function is designed to handle both the new and old header formats, and extracts data such as offsets of the headers and CSM data, the general scale factor, and the like.

The second function of tp.c is extract the Stokes Matrix elements from the CSM data, accomplished within the body of function **csm2pwr()**. Presented here is a discussion of the CSM decompression method and how it applies to the operations of the code.

The first task after evaluating the header data is the computation of the transmit

and receive polarization vectors. The vectors are stored as **stvec** and **srvec**, respectively, as is in the Dubois code. Array M_{ij} is then loaded with these vectors according to

$$\vec{M}=\vec{S}_r^{\,T}\times\vec{S}_t$$

The 2-element sum in the line

M[i][j] = srvec[i]*stvec[j] + srvec[j]*stvec[i]

is done since although only the upper triangular elements will be computed in the decompression loop, the vector multiplication must be carried out over the whole matrix.

The next step is to precompute and store all possible results of the decompression formulas in array variables; **EQ1** is an array containing the precomputed results of $\hat{F}_{11}$; **EQ2**, those of the expression $\frac{i}{127.0}$; and **EQ3**, those from $\left[\frac{i}{127.0}\right]^2 * sign(i)$, comprising a basic set of operations of the decompression equations.

The main decompression loop begins by reading in one whole 1024-sample line; then, for every pixel, the following process takes place. Each 10-byte sample is broken into individual bytes by the macro **dbyte(i)**, $\forall i \in [0,9]$ corresponding to the i^{th} byte of the sample. $\hat{F}_{11}$ is obtained by using bytes 1 and 2 (**dbyte(0)** and **dbyte(1)**) to dereference array **EQ1**. The other elements, represented by array **F**, actually represent elements of the X_{ij} array; it is only in the last part of the loop -- the summation of variable **power** -- that these elements take on their corresponding F_{ij} values. Notice that the distributive laws of arithmetic allow for the convenience of saving the rescaling by $\hat{F}_{11}$ for the last computation -- **power** *= $\hat{F}_{11}$.

The final part of the loop writes the line of decompressed real numbers to the output file, and the last part of the code closes the appropriate files.

An important note on the decompression loop is that the use of array **F** represents the computation of the X_{ij}, not the computation of the F_{ij} elements; the true values of the elements F_{ij} are never individually computed. This is a consequence of the design of the loop which is intended to minimize the number of arithmetic operations. True determination of the F_{ij} values would require a per-element multiplication by both the arrays **M** and the transmit and receive vectors **srvec** and **stvec**, which is done in the Dubois code; as is, tp.c has been designed for speed, and thus contains shortcuts to minimize the number of arithmetic operations.

Section 2: Verification of Decompression Program

Presented in this section is an automated method for verifying the total power values generated by tp.c. This method of verification uses a 10-byte sample with a direct application of the decompression formulas. The program makes use of an efficient system for decompressing 10-byte samples based on, but not directly resembling, these formulas, and thus requires some form of verification.

The verification method is coded as a LOTUS 1-2-3® spreadsheet, and offers versatility in that any 10-byte sample may be decompressed; however, there is some complexity in the implementation of the decompression formulas in LOTUS. Presented in the appendices is a detailed listing of the cell formulas; one formula of particular interest is a formula to compute x^y for positive x. A logarithmic equivalent is used in LOTUS since no direct function exists:

$$x^y = e^{y \cdot \ln(x)}$$

for positive x.

The LOTUS spreadsheet was executed on the byte values taken from the Mahantango 080-3 data set, element (1,1):

byte	value
1	128
2	66
3	204
4	210
5	136
6	211
7	194
8	171
9	92
10	163

For a general scale factor of 44.51778, the C decompression system reports a power value of 71.871094, in close agreement with the spreadsheet results.

The spreadsheet output is presented below:

Mahantango 080-3 L-BAND, pixel (1,1)

User Entry Area:

Gen_fac
44.51778

Vectors	Degrees
tr ellip	45
tr orien	45
rc ellip	45
rc orien	45

Byte Values	[0..255]
byte 1	128
byte 2	66
byte 3	204
byte 4	210
byte 5	136
byte 6	211
byte 7	194
byte 8	171
byte 9	92
byte 10	163

Computation Area:

stvec

1	0	0	1

Vectors	Radians
tr ellip	0.785398
tr orien	0.785398
rc ellip	0.785398
rc orien	0.785398

srvec

1
0
0
1

Bytes-128	Xij array	
0	55.91013	F11hat
-62		
76	0.598425	X12
82	0.416889	X13
8	0.003968	X14
83	0.427119	X23
66	0.270073	X24
43	0.338583	X33
-36	-0.28346	X34
35	0.275591	X44

F array

55.91013	33.45803	23.30831	0.221852
33.45803	21.57162	23.88027	15.09979
23.30831	23.88027	18.9302	-15.8485
0.221852	15.09979	-15.8485	15.4083

Multiply cols by stvec below

55.91013	0	0	0.221852
33.45803	0	0	15.09979
23.30831	0	0	-15.8485
0.221852	0	0	15.4083

Multiply those rows by srvec below

```
55.91013          0          0 0.221852
       0          0          0          0
       0          0          0          0
0.221852          0          0  15.4083

Results:
Total Power:       71.76213
```

Section 3: Compilation and Execution

This section presents compilation and usage instructions for the system. Complete copies of the code can be obtained from the authors through electronic mail or other means. The figures accompanying this appendix represent an actual compilation and execution of the system on the Mahantango 080-3 data set (L-band), and are presented for quick reference on constructing the system.

Compilation

The system of programs has been developed for use on UNIX systems, and consists of several C subprograms and header files that comprise a heirarchical menu system of tasks related to CSM data decompression. Figure 1 is a listing of the C subprograms as they appear in the sample directory; each program is listed below and described:

Program Name	Function
Makefile	directs creation of the system under the UNIX "make" program; requires BSD 4.3 or later of UNIX
data_file	listed in figure 2, contains information on where to find the data file
types.h, menu.h	global header files containing basic data types and menu system definitions
main2.c	official definition of function main(); calls function inter_main() in file im.c, for an interactive execution (this has been left open for an extension for a batch routine later)
im.c	definition of function inter_main(), drives the main menu and controls prepfiles(prep.c) and csm2pwr(tp.c), and other general purpose functions
prep.c	contains function prepfiles(), reads the header information as described in section 1

tp.c	definition of function csm2pwr(), performs the data decompression into real data as described in section 1
cls.c	contains a routine to clear the screen; the user can modify the definition of "CLEAR_COMMAND" to implement a clear-screen feature
ga.c,sub.c,util.c	various routines called from other functions
imold.c	not used at present

```
cemeserv[~/nasa2]-16:ls
Makefile    data_file   imold.c     prep.c      types.h
avgreals.c  ga.c        main2.c     sub.c       util.c
cls.c       im.c        menu.h      tp.c
```

Figure 1

The system is compiled using the UNIX "make" utility, and requires ANSI-C ("gcc" or equivalent for UNIX systems). Currently, only ANSI-C is supported, although only minor changes need be made to support non-ANSI C. Finally, the resulting file "main" is executed.

Customization

The system has been designed with a bit of customization possible through defines and the make file. Currently, most of the customizable options are handled through the C preprocessor and the Makefile, although one specific option is available to the user.

The Makefile handles all the compilation; thus, to change any of the compilation options (such as the name of the C-compiler), the user can modify the Makefile directly in accordance with the documentation on the UNIX make utility. Debugging is available to the user for the purposes of extending or modifying the code: execute the command

make "CFLAGS=-DDEBUG"

in order to turn the debugging flags on. This change in definition effectively activates certain sections of code that make debugging easier.

The only customizable user option at the present is the ability to change the definition of clear screen in the file cls.c. To do this, first identify the clear screen command on the machine this is to be run on. Next, change the definition of "CLEAR_COMMAND" in line 3 of the file to contain the operating system command to clear the screen; for UNIX, this definition would be:

#define CLEAR_COMMAND "clear"

In the current setting, "echo ----------" causes UNIX to print a horizontal line segment each time a clear screen request is made of the program; the effect of this is to separate the menus by a line of dashes.

Execution

Program "main" is the compiled result of the system of subprograms. Main operates interactively with a series of heirarchical menus: the user can select from just those tasks that are feasible at the current time. A description of these tasks is now presented in the form of a sample interactive session. Figures 3-7 represent an actual execution of the program on the Mahantango 080-3 data set; each of these menus will be described below.

Figure 3 is the opening menu that the user sees. The user can open a file, or quit the program. In this case, option 1 was selected to open a data file.

```
JPL CSM Decompression Program -- Main Menu

        1.      Open Data File

        0.      Quit

Your choice [0-1]: 1
```

Figure 3

In figure 3a, the user enters the name of the data file. Data file names can either name the actual CSM data file, or can name a file consisting of a '#' as the first character, and a path to an actual CSM file; the file "data_file" is listed in figure 3b as an example to this indirect file referencing.

```
JPL CSM Decompression Program -- Open Data File

Enter the name of the data file (or pointer file): data_file
(opening /homes/jsircar/airsar/machydro_data/tango.lband)
File is in new header format...
```

Figure 3a

```
cemeserv[~/nasa2]-17:cat data_file
#       /homes/jsircar/airsar7machydro_data/tango.lband
#
# Put the name of the file you want to reference indirectly on the top
# line of this file; later, this could be made into a database for
# multiple files, selectable at the user's option.
```

Figure 3b

Once the file is opened, the menu of fig. 4 appears. The user selects option 2 to enter the data set attributes. Figure 4a shows the resulting screen where the user is prompted for the integer degrees of various angles corresponding to the transmit and receive orientation vectors; in this case, a 45-degree angle was chosen for all orientations.

```
JPL CSM Decompression Program -- Main Menu

        1.      Open Data File

        2.      Enter DS Attributes

        0.      Quit

Your choice [0-2]: 2
```

Figure 4

```
JPL CSM Decompression Program -- Get Attributes

Enter the following INTEGER attributes for scene:  Mahantango 080-3


Transmit ellipticity angle:  45
Transmit orientation angle:  45
Receive ellipticity angle:   45
Receive orientation angle:   45
(Press ENTER to continue...)
```

Figure 4a

After the data set attributes are selected, the menu of figure 5 is shown. Here, the user must choose a quantity of CSM data to decompress; either the whole data set can be chosen, or a small rectangular subsection bounded by row, column coordinates can be used. The user chooses option 4, to select a partial area, bringing up the screen in figure 5a. Now, the user enters the "ULHC" and "LRHC" (upper-left-hand-corner and lower-right-hand-corner) coordinates, in

row(y, top-to-bottom), column(x, left-to-right) format. Note that the origin is centered at (0,0), not (1,1).

```
JPL CSM Decompression Program -- Main Menu

      1.    Open Data File
      2.    Enter DS Attributes
      3.    Select Entire DS
      4.    Select Partial Area
      0.    Quit

Your choice [0-4]: 4
```

Figure 5

```
JPL CSM Decompression Program -- Sub Section

Enter ULHC of bounding box (row,col) orig = (0,0): 0,0
Enter LRHC (row,col): 200,200
(Press ENTER to continue...)
```

Figure 5a

Note that whenever a partial area has been selected for decompression, any future row, column references are taken as offsets from within the partial area; whenever the whole data set has been selected, row and column references are treated as references to absolute coordinates. For example, assume a partial area with bounding ULHC,LRHC corners of (1,1) and (10,10) has been selected and decompressed; now, a request for a power value of a pixel located at (5,3) will result in the pixel at (6,4) to be returned, as (6,4) is 5 rows down and 3 columns over from (1,1).

Once an area has been selected, the program displays the menu of figure 6, which allows decompression of the selected area. The user chooses this option, and the screen of fig. 6a pops up as the decompression is performed.

```
JPL CSM Decompression Program -- Main Menu

        1.      Open Data File
        2.      Enter DS Attributes
        3.      Select Entire DS
        4.      Select Partial Area
        5.      Decompress Your Selection
        0.      Quit

Your choice [0-5]: 5
```

Figure 6

```
JPL CSM Decompression Program -- Decompression

Preparing for decompression...
Beginning decompression loop...

(Press ENTER to continue...)
```

Figure 6a

After the decompression, the menu of figure 7 is displayed. From this menu, options 6 through 8 allow pixel- or region-wise analysis of the resulting real data. Figure 7a shows a selection of option 6, which computes an average of all the real values decompressed from the selected area; figure 7b shows a listing of the 10-byte record corresponding to pixel (1,1) in the data set, including values to supply to the Lotus spreadsheet program for verification; and figure 7c shows the decompressed real value corresponding to the 10-byte record listed in figure 7b.

```
JPL CSM Decompression Program -- Main Menu

        1.      Open Data File
        2.      Enter DS Attributes
        3.      Select Entire DS
        4.      Select Partial Area
        5.      Decompress Your Selection
        6.      Find Avg Power for Selection
        7.      List Byte Values for a Pixel
        8.      List Power Values for a Pixel
        0.      Quit

Your choice [0-8]: 6
```

Figure 7

```
JPL CSM Decompression Program -- Average Reals

Average power over data region is 32.840782
(Press ENTER to continue...)
```

Figure 7a

```
JPL CSM Decompression Program -- List Bytes

Enter the row and column (r,c) of the element: 1,1
Byte 0: 0               lotus, use 128
Byte 1: -62             lotus, use 66
Byte 2: 76              lotus, use 204
Byte 3: 82              lotus, use 210
Byte 4: 8               lotus, use 136
Byte 5: 83              lotus, use 211
Byte 6: 66              lotus, use 194
Byte 7: 43              lotus, use 171
Byte 8: -36             lotus, use 92
Byte 9: 35              lotus, use 163
(Press ENTER to continue...)
```

Figure 7b

```
JPL CSM Decompression Program -- List Power

Enter the row and column (r,c) of the element: 1,1
Decompressed value is 71.871094
(Press ENTER to continue...)
```

Figure 7c

At the end of the analysis, option 0 is chosen, and the program terminates and returns the user to UNIX.

Appendix 1: LOTUS Spreadsheet Listing

Mahantango 080-3 L-BAND, pixel (1,1)

```
A:A1: 'User Entry Area:
A:D1: 'Byte Values
[0..255]
A:D2: 'byte 1
A:E2: 128
A:D3: 'byte 2
A:E3: 66
A:A4: 'Gen_fac
A:D4: 'byte 3
A:E4: 204
A:A5: 44.51778
A:D5: 'byte 4
A:E5: 210
A:D6: 'byte 5
A:E6: 136
A:A7: 'Vectors
A:B7: 'Degrees
A:D7: 'byte 6
A:E7: 211
A:A8: 'tr ellip
A:B8: 45
A:D8: 'byte 7
A:E8: 194
A:A9: 'tr orien
A:B9: 45
A:D9: 'byte 8
A:E9: 171
A:A10: 'rc ellip
A:B10: 45
A:D10: 'byte 9
A:E10: 92
A:A11: 'rc orien
A:B11: 45
A:D11: 'byte 10
A:E11: 163
A:A13: 'Computation Area:
A:A14: 'stvec
A:A15: 1
A:B15: @COS(2*B18)*
       @COS(2*B19)
A:C15: @SIN(2*B18)*
       @COS(2*B19)
A:D15: @SIN(2*B19)
A:A17: 'Vectors
A:B17: 'Radians
A:E17: 'srvec
A:A18: 'tr ellip
A:B18: +B8*@PI/180
A:E18: 1
A:A19: 'tr orien
A:B19: +B9*@PI/180
A:E19: @COS(2*B20)*@COS(2*B21)
A:A20: 'rc ellip
A:B20: +B10*@PI/180
A:E20: @SIN(2*B20)*@COS(2*B21)
A:A21: 'rc orien
A:B21: +B11*@PI/180
A:E21: @SIN(2*B21)
A:A24: 'Bytes-128
A:C24: 'Xij array
A:A25: +E2-128
A:C25: @EXP(A25*@LN(2))*
       (A26/254+1.5)*A5
A:D25: 'F11hat
A:A26: +E3-128
A:A27: +E4-128
A:C27: +A27/127
A:D27: 'X12
A:A28: +E5-128
A:C28: (A28/@ABS(A28))*@EXP(2*@LN(
       @ABS(A28/127)))
A:D28: 'X13
A:A29: +E6-128
A:C29: (A29/@ABS(A29))*@EXP(2*@LN(
       @ABS(A29/127)))
A:D29: 'X14
A:A30: +E7-128
A:C30: (A30/@ABS(A30))*@EXP(2*@LN(
       @ABS(A30/127)))
A:D30: 'X23
A:A31: +E8-128
A:C31: (A31/@ABS(A31))*@EXP(2*@LN(
       @ABS(A31/127)))
A:D31: 'X24
A:A32: +E9-128
A:C32: +A32/127
A:D32: 'X33
A:A33: +E10-128
```

```
A:C33: +A33/127
A:D33: 'X34
A:A34: +E11-128
A:C34: +A34/127
A:D34: 'X44
A:A36: 'F array
A:A37: +C25
A:B37: +C27*C$25
A:C37: +C28*C$25
A:D37: +C29*C$25
A:A38: +B37
A:B38: +A37-C39-D40
A:C38: +C30*C$25
A:D38: +C31*C$25
A:A39: +C37
A:B39: +C38
A:C39: +C32*C$25
A:D39: +C33*C$25
A:A40: +D37
A:B40: +D38
A:C40: +D39
A:D40: +C34*C$25
A:A42: 'Multiply cols by
       stvec below
A:A43: +A$15*A37
A:B43: +B$15*B37
A:C43: +C$15*C37
A:D43: +D$15*D37
A:A44: +A$15*A38
A:B44: +B$15*B38
A:C44: +C$15*C38
A:D44: +D$15*D38
A:A45: +A$15*A39
A:B45: +B$15*B39
A:C45: +C$15*C39
A:D45: +D$15*D39
A:A46: +A$15*A40
A:B46: +B$15*B40
A:C46: +C$15*C40
A:D46: +D$15*D40
A:A48: 'Multiply those
       rows by srvec
       below
A:A49: +$E18*A43
A:B49: +$E18*B43
A:C49: +$E18*C43
A:D49: +$E18*D43
A:A50: +$E19*A44
A:B50: +$E19*B44
A:C50: +$E19*C44
A:D50: +$E19*D44
A:A51: +$E20*A45
A:B51: +$E20*B45
A:C51: +$E20*C45
A:D51: +$E20*D45
A:A52: +$E21*A46
A:B52: +$E21*B46
A:C52: +$E21*C46
A:D52: +$E21*D46
A:A54: 'Results:
A:A55: 'Total Power:
A:C55: @SUM(A49..D52)
```

Appendix 2: Source Listing

```
***** FILE: Makefile

CC=gcc
CFLAGS=-g
LFLAGS=-lm
TPFILS=prep.o main2.o cls.o util.o sub.o avgreals.o im.o ga.o tp.o
TPDEPS=types.h menu.h

main: $(TPFILS) $(TPDEPS)
      $(CC) $(CFLAGS) $(TPFILS) $(LFLAGS) -o main

clean:
      rm -f main
      rm -f *.o
      rm -f core
      touch *.c

clear:
      rm -f *.o
      rm -f core
```

```
***** FILE: types.h

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <math.h>

#define flag            printf ("<%s:%d>", __FILE__,__LINE__); printf
#define querys(var)     printf ("String variable var is %s\n", var)
#define queryi(var)     printf ("Integer variable var is %d\n", var)
#define queryf(var)     printf ("Real variable var is %e\n", var)
#define bomb(str)       fprintf(stderr,"Fatal error: %s\n",str);
#define max(x,y)        ((x > y) ? x : y)
#define min(x,y)        ((x > y) ? y : x)
#define pol(x)          x*3.14159265/90.0 /* only valid for stokes computations
*/
#define abs(x)          (x < 0) ? -1 * x : x /* works for reals */
#define MAGIC           0x4a4d4f69          /* All bytes different, used for
write32*/
#define SIGN(X)         (((X)<0)?-1:1)
#define MAX(X,Y)        (((X)>(Y))?(X):(Y))
#define ToRad(X)        ((float)X*acos(-1.)/180.)
#define writereal(R,OUT)      fwrite(&(R),4L,1L,OUT)
#define readreal(PR,IN)       fread(&(PR),4L,1L,IN)
#ifndef MAX_STR_LEN
#define MAX_STR_LEN     255
#endif
#define FIELD_WIDTH     50
#define bool            int
#define true            -1
#define false           0

/**** definitions for generic use ****/
#define WAIT {printf("(Press ENTER to continue...)"); getchar(); getchar();}
#define TOPLINE(str) { clear_screen(); \
      printf("JPL CSM Decompression Program -- %s\n\n",str);}

/**** structure definitions, global variables *************************/
typedef struct headertypestruct {
      FILE *fp;               /* pointer to disk file */
      char fname[80];               /* name of disk file */
      char scene[80];               /* scene title */
      int window;             /* 0=whole ds; 1-9=subwindows */
      int rlib;                     /* record length in bytes */
      int nohr;                     /* number of header records */
      int boooh;              /* byte offset of old header */
      float gen_fac;                /* general scale factor */
      int ichit;
      int ipsit;
      int ichir;
      int ipsir;                    /* polarization angles */
      int nline;                    /* number of lines(records) */
      int nsamp;              /* number of pixels(samples) per line */
      int boofdr[10];               /* byte-offset of first data record */
      int w[10];              /* subsection width */
      int h[10];                    /* subsection height */
      } *headertype;

#define make(TYP) (TYP)malloc(sizeof(struct TYP##struct))

/**** global functions ********************************************/
```

```
extern char *matchafter();
extern void prepfiles();
extern void clear_screen();
extern void sub_sect();
extern void csm2pwr();
extern float avg_reals();
extern void inter_main();
extern void get_loc();
extern void dump_header();

/**** global variables ********************************************/
extern char *_cls_string;
```

```
***** FILE: menu.h

#define MAXMENUS  20

#define MENUSTART       { int MENU_num, MENU_ans=-1, MENU_level=0, \
                  MENU_ct=0, MENU_goto=0; \
                  MENU_type MENU_ents[10];\
                        while(MENU_ans) { \
                        MENU_num=1; MENU_ct=0;
#define MENUEND    } /* end menu while */ }
#define MENU(lvl,str)    MENU_ct++;  \
                  if (MENU_level>=lvl){ \
                        MENU_ents[MENU_num].level=lvl; \
                        MENU_ents[MENU_num].casetag=MENU_ct; \
                        printf(MENUFMT,MENU_num++,str); }
#define MENUIN            MENU_level=MENU_ents[MENU_ans].level+1
#define MENUOUT    MENU_level=MENU_ents[MENU_ans].level-1
#define MENUSELBEGIN     printf(MENUFMT,0,"Quit"); \
                     MENU_ans=getchoice(MENU_num-1); \
                     if(MENU_ans!=0) \
                       MENU_goto=MENU_ents[MENU_ans].casetag; \
                     else MENU_goto=0; \
                       switch(MENU_goto){
#define MENUSELEND       default: /* quit code here */; break; }
#define MENUFMT          "\t%d.\t%s\n\n"

typedef struct menustruct {
      int level, casetag;
      } MENU_type;

int getchoice(int n)
      /* returns an integer in the range [0..n] corresponding to user
         input; routine will lock until proper input is obtained */
      {
            char choice[10];
            int ich = -1;

            while( ich<0 ) {
                  printf("Your choice [0-%1d]: ",n);
                  scanf(" %s",choice);
                  if ((!isdigit(choice[0])) ||
                     ((ich=atoi(choice))>n) || (ich < 0)) {
                        printf("Invalid input \"%d\"; try again. ",ich);
                        ich = -1;
                        }
                  else
                        return ich;
                  } /* while */
      } /* getchoice */
/*****************/
```

```
***** FILE: main2.c

/* main2.c
 * intended as a replacement for main.c
 *
 * integrate checks on argc, argv to be able to run batch mode, later.
 */

#include "types.h"

main(int argc, char **argv)
{
      inter_main(argc, argv);
      printf("Operation complete.\n");
}
```

```
***** FILE: im.c

/*** im.c, inter_main interactive function */

#include "types.h"
#include "menu.h"       /* new menu system */

void inter_main(int argc, char **argv) { /* variable declarations */
      headertype head, realwin;
      int c;
      long int fpos;

      head=make(headertype);
      realwin=make(headertype); /* contains tmpfile of real #'s */
      fclose(realwin->fp);
      fclose(head->fp);
      /* loop until done */
      MENUSTART
            TOPLINE("Main Menu");
            MENU(0,"Open Data File");                          /* 1 */
            MENU(1,"Enter DS Attributes");                      /* 2 */
            MENU(2,"Select Entire DS");                        /* 3 */
            MENU(2,"Select Partial Area");                      /* 4 */
            MENU(3,"Decompress Your Selection");                /* 5 */
            MENU(4,"Find Avg Power for Selection");            /* 6 */
            MENU(4,"List Byte Values for a Pixel");             /* 7 */
            MENU(4,"List Power Values for a Pixel");           /* 8 */
#ifdef DEBUG
            MENU(2,"List DS header values -- DEBUGGING");       /* 9 */
            MENU(3,"List realwin header values");               /* 10 */
#endif
            MENUSELBEGIN
                  case 1: /* open file menu */
                        open_file(argc, argv, head);
                        prepfiles(head);
                        MENUIN;
                        break;
                  case 2: /* get attrs -> tr, rc, angles... */
                        get_DS_attrs(head);
                        MENUIN;
                        break;
                  case 3: /* whole data set */
                        TOPLINE("Whole DS");
                        head->window=0; /* use whole set */
                        printf("Entire data set selected.\n");
                        WAIT;
                        MENUIN;
                        break;
                  case 4: /* partial data set */
                        TOPLINE("Sub Section");
                        head->window=1;
                        sub_sect(head);
                        WAIT;
                        MENUIN;
                        break;
                  case 5: /* perform decompression from above selections */
                        TOPLINE("Decompression");
                        csm2pwr(head,realwin);
                        WAIT;
                        MENUIN;
                        break;
                  case 6: /* average power */
```

```
                    TOPLINE("Average Reals");
                    avg_reals(realwin);
                    WAIT;
                    break;
                case 7: /* look at one pixel's values */
                    TOPLINE("List Bytes");
                    {
                    char ch;
                    get_loc(head,10);
                    for(c=0;c<10;c++){
                      ch=(char)getc(head->fp);
                      printf("Byte %d:\t%d\t\tlotus, use %d\n",
                          c,(int)ch,(int)ch+128);
                      }
                    }
                    WAIT;
                    break;
                case 8: /* look at point power, if decompressed */
                    {
                    float f;
                    TOPLINE("List Power");
                    get_loc(realwin,4);
                    readreal(f,realwin->fp);
                    printf("Decompressed value is %f\n",f);
                    WAIT;
                    }
                    break;
                case 9: /* dump out header data type */
                    TOPLINE("Dumping Header");
                    dump_header(head);
                    WAIT;
                    break;
                case 10: /* dump realwin header info */
                    TOPLINE("Dumping realwin header");
                    dump_header(realwin);
                    WAIT;
                    break;
        MENUSELEND
    MENUEND
} /* inter_main */
```

```
***** FILE: prep.c

/*** prep.c, function prepfiles (how much is global???) */
#include "types.h"

char *matchafter(char *string, char *buffer, int buflen)
/* looks for string in buffer, returns 50 character field with the string minus
the string
 * itself. */
      {
        char *fs;
        int i,found;

        fs=(char *)malloc((long)51);
        fs[50]='\0';

        /* preload the comparison buffer */
        for(i=1;i<50;i++){
            fs[i]=*(buffer++);
            buflen--;
        }

        found=0;
        while ((buflen) && (!found)){
            /* slide the window */
            for(i=0;i<49;i++) fs[i]=fs[i+1];
            fs[i]=*(buffer++);
            buflen--;

            /* do the comparison */
            for(found=1,i=0; (i<strlen(string)) && (found); i++)
                  found &= (string[i]==fs[i]);
      }

      if (found) {
            /* clean up string, i.e., remove control characters */
            for (i=0; i<50; i++)
                  if ((fs[i]<(int)32)||(fs[i]>(int)126))
                        fs[i]=(int)32;

            /* now, remove the original part of the search path */
            for (i=0; i<50-strlen(string); i++)
                  fs[i]=fs[i+strlen(string)];
            fs[i]='\0';

            return fs;
      }
      else
        return (char *)NULL;
} /* matchafter */

void prepfiles (headertype header)
      {

      char *tmp, buf[20480];
      int ch;

      int newheader, i, j, flen;

      /* additional change: check if file is a pointer to another file;
       * Such files will meet the following format specification:
       * <FILE> ::=     #<path> | <data>
```

```
        * <path> ::=      standard UNIX path without wildcards
        * <data> ::=      standard JPL CSM data
        *
        *  Originally developed to eliminate name-hunting on networked file
systems.
        */

       /* determine new or old header */
       fread(buf,(long)1,(long)10,header->fp);
       buf[9] = '\0';
       newheader = (strcmp (buf, "RECORD LE") == 0) ? -1 : 0;
       fseek(header->fp, 0L, 0);

       /* process new header file */
       if (newheader){
            printf ("File is in new header format...\n");

            /* determine record length in bytes */
            flen=fread(buf,(long)1,(long)50,header->fp);
            tmp=matchafter("RECORD LENGTH IN BYTES = ",buf,flen);
            sscanf(tmp," %d",&(header->rlib));

            /* determine number of header records */
            flen=fread(buf,(long)1,(long)50,header->fp);
            tmp=matchafter("NUMBER OF HEADER RECORDS = ",buf,flen);
            sscanf(tmp," %d",&(header->nohr));

            /* rewind, and get rest of header */
            fseek(header->fp,(long)0, 0);
            flen=fread(buf,(long)1,(long)header->rlib*header->nohr,header->fp);

/*** ready to process header file info */

            /* get number of samples per record */
            tmp=matchafter("NUMBER OF SAMPLES PER RECORD = ",buf,flen);
            sscanf(tmp," %d",&(header->nsamp));

            /* get byte offset of old header */
            tmp=matchafter("BYTE OFFSET OF OLD HEADER = ",buf,flen);
            sscanf(tmp," %d",&(header->boooh)); /* use this to skip to old header
*/

            /* get number of lines in image */
            tmp=matchafter("NUMBER OF LINES IN IMAGE = ",buf,flen);
            sscanf(tmp," %d",&(header->nline));

            header->w[0]=header->nsamp;
            header->h[0]=header->nline;

            /* get byte offet of first data record */
            tmp=matchafter("BYTE OFFSET OF FIRST DATA RECORD = ",buf,flen);
            sscanf(tmp," %d",&(header->boofdr[0]));

            /* skip to old header, get scale factor; reload buffer */
            fseek(header->fp,(long)header->boooh,0);
            /* read in the old header record */
            flen=fread(buf,1L,(long)header->rlib,header->fp);

            /* get gen_fac */
            tmp=matchafter("COMP SCALE FACTOR:",buf,flen);
            sscanf(tmp," %e",&(header->gen_fac));
            /* old comp scale factor --
```

```
                fseek (header->fp, 16872, 0); */

            /* get scene title */
            tmp=matchafter("SCENE TITLE:",buf,flen);
            sscanf(tmp,"%35c",&(header->scene));

        } /* endif */
        else { /* old header */
            printf ("File is in old header format,");
            printf (" assuming 1024x750 pixels...\n");
            header->nsamp = 1024;
            header->nline = 750;
            header->boofdr[0] = 10240;
            header->w[0]=header->nsamp;
            header->h[0]=header->nline;
            fseek (header->fp, (long) 10492392, 0);
            fread (buf, 1L, 13L, header->fp);
            sscanf (buf, "%e", &(header->gen_fac));
            } /* endelse */
        fseek(header->fp,header->boofdr[0],0L);
} /* prepfiles */
```

```
***** FILE: sub.c

/*** sub.c */
#include "types.h"

void sub_sect(headertype h)
      /* prompts user for row, col bounding coords, returns subsectioned CSM
       * data from "in" into "sw" */
      {
            int r1,c1,r2,c2;
            int w;

            w=h->window;

            printf("Enter ULHC of bounding box (row,col) orig = (0,0): ");
            scanf("%d,%d",&r1,&c1);
            printf("Enter LRHC (row,col): ");
            scanf("%d,%d",&r2,&c2);

            /* determine width,height in terms of 10-byte pixels */
            h->w[w] = c2-c1+1;
            h->h[w] = r2-r1+1;

            /* compute boofdr */
            h->boofdr[w] = h->boofdr[0]+10*(r1*h->nsamp + c1);
            fseek(h->fp, h->boofdr[w], 0);

            /* later, modify to allow for multiple subsection selections */
      } /* sub_sect */
```

```
***** FILE: tp.c

/* TotalPower.c, by Russell Fink (rfink@eng.umd.edu)*/
#include "types.h"

/* global array definitions -- used so they don't have to always be recomputed
*/
float EQ1[256][256],EQ2[256],EQ3[256];
int computed_yet=0;     /* not yet computed */

void csm2pwr(headertype h, headertype rw)
{
/* variable declarations */
      float stvec[4],srvec[4];
      int i,j;
      float M[4][4];
      extern float EQ1[256][256],EQ2[256],EQ3[256];
      float F[4][4],F11hat,power;
      int line;
      int pixel;
      char dataline[10][1024];
      float line_of_reals[1024];
      long int skip;

      printf("Preparing for decompression...\n");

/* initialize real window (rw) and copy some attributes from the main window (h)
*/
      fclose(rw->fp);   /* if already open; */
      strcpy(rw->fname,"<tempfile>");     /* be able to rename this later */
      rw->fp=tmpfile(); /* for some reason, opening a normal file "wb+" won't
work */
      fseek(rw->fp,0L,0L);
      rw->rlib=4*1024;  /* sizeof real numbers * records -- not important*/
      rw->nohr=0;
      rw->nline=h->h[h->window];
      rw->nsamp=h->w[h->window];
      rw->boofdr[0]=0;
      rw->h[0]=rw->nline;
      rw->w[0]=rw->nsamp;
      rw->window=0;

/* convert ellipticity and orientation angles from degrees to radians, and
 * calculate transmit and receive stokes vectors for chosen polarization
 * combination */

      stvec [0] = 1.0;
      stvec [1] = cos (2.0*ToRad(h->ipsit)) * cos (2.0*ToRad(h->ichit));
      stvec [2] = sin (2.0*ToRad(h->ipsit)) * cos (2.0*ToRad(h->ichit));
      stvec [3] = sin (2.0*ToRad(h->ichit));

      srvec [0] = 1.0;
      srvec [1] = cos (2.0*ToRad(h->ipsir)) * cos (2.0*ToRad(h->ichir));
      srvec [2] = sin (2.0*ToRad(h->ipsir)) * cos (2.0*ToRad(h->ichir));
      srvec [3] = sin (2.0*ToRad(h->ichir));

/* Create transmit, receive polarization multiplicity matrix (scale array) */
      for(i=0;i<4;i++)
          for(j=i;j<4;j++){
            if(i==j)
                  M[i][j]=srvec[i]*stvec[j];
            else
```

```
                M[i][j]=srvec[i]*stvec[j] + srvec[j]*stvec[i];
        }

/* Construct decompression 'equation' arrays --
 * EQ1[byte 1][byte 2]: returns F11hat given values of bytes 1 & 2
 * EQ2[byte n]:         returns byte n/127; used for X12,X33,X34,X44
 * EQ3[byte n]:         returns square(n/127)*sign for X13,X14,X23,X24
 *
 * These equations remove a bulk of the arithmetic operations from the
 * main decompression loop by replacing the costly arithmetic with simple
 * look-up operations.
 *
 * If these have been computed already (from another pass), do NOT recompute.
 */
     if(!computed_yet) {
           for(i=-128;i<128;i++){
               for(j=-128;j<128;j++)
                 EQ1[i+128][j+128]=((float)j/256.0+1.5)*pow(2.0,(float)i)
                                   * (float)h->gen_fac;
               EQ2[i+128]=(float)i/127.0;
               EQ3[i+128]=(float)SIGN(i)*pow((float)i/127.0,2.0);
               }
           computed_yet=-1;
           } /* if */

/* decompress and scale the power for each pixel in the data array; to simplify
 * expressions, use the c preprocessor macro utility to quickly dereference
 * data in the array */

/* Skip to byte offset of first data record */
     fseek(h->fp,h->boofdr[h->window],0);

/* begin decompression loop */
#define dbyte(x) ((int)dataline[x][pixel])+128

printf ("Beginning decompression loop...\n\n");
     for (line=0; line < h->h[h->window]; line++) {

#ifdef DEBUG
          printf ("^[[A"); /* fill in the ansi move-up code */
          printf ("Decompressing line %d... \n", line);
#endif

/* read the next line of data */
         for (pixel=0; pixel<h->w[h->window]; pixel++)
           for(i=0; i<10; i++)
                dataline[i][pixel]=fgetc(h->fp);
         for (pixel=0; pixel<h->w[h->window]; pixel++) {

/* fill in code for power loop here; F11hat = F[0][0] */
          F11hat=EQ1[dbyte(0)][dbyte(1)];

/* multiply each remaining Xij by F11hat to get Fij */
          F[0][0]=1.0;      /* will be factored in later */
          F[0][1]=EQ2[dbyte(2)];
          F[0][2]=EQ3[dbyte(3)];
          F[0][3]=EQ3[dbyte(4)];
          F[1][2]=EQ3[dbyte(5)];
          F[1][3]=EQ3[dbyte(6)];
          F[2][2]=EQ2[dbyte(7)];
          F[2][3]=EQ2[dbyte(8)];
          F[3][3]=EQ2[dbyte(9)];
```

```
            F[1][1]=F[0][0]-F[2][2]-F[3][3];

/* add up elements of F * rotation matrix M to get total power */
            power=0.0;
            for(i=0;i<4;i++)
                for(j=i;j<4;j++)
                  power += F[i][j]*M[i][j];

            power *= F11hat;

/* above code requires 11 mults and 12 adds */

/* handle problems where power is slightly less than zero */
            power = MAX(0.0000,power);

/* dump raw power */
            line_of_reals[pixel] = power;

/* skip to next record in sample window (from current position */
            skip=10*(h->nsamp - h->w[h->window]);
            fseek(h->fp,skip,1L);

            } /* endfor (pixel) */

/* write a line of data */
      for(pixel=0;pixel<rw->w[0];pixel++){
            writereal(line_of_reals[pixel],rw->fp);
            }
      } /* endfor (line) */

} /* endmain */
```

```
***** FILE: avgreals.c

/*** avgreals.c */

#include "types.h"

float avg_reals(headertype rw)
      {
            int i,j,win;
            float avg, t;

            if(ftell(rw->fp)<0){
                  printf("Must decompress data, first!\n");
                  WAIT;
                  return 0.0;
                  }
            win=rw->window;
            fseek(rw->fp,0L,0L);
            avg = 0.0;
            for(i=0; i<rw->h[win]; i++)
                  for(j=0; j<rw->w[win]; j++) {
                        readreal(t, rw->fp);
#ifdef DEBUG
      flag("Read %f\n",t);
#endif
                        avg += t;
                        }
            avg = avg / (float)(i*j);
            printf("Average power over data region is %f\n",avg);

            return avg;
      } /* avg_reals */
```

```
***** FILE: ga.c

/*** ga.c */
#include "types.h"

void get_DS_attrs(headertype h)
/* loads data set attributes; separate procedure to allow expansion/savable
attributes */
{
      int f;

      TOPLINE("Get Attributes");
      printf("Enter the following INTEGER attributes for scene:
%s\n\n",h->scene);
      printf("Transmit ellipticity angle:  ");
      scanf(" %d",&f);
      h->ichit=f;
      printf("Transmit orientation angle:  ");
      scanf(" %d",&f);
      h->ipsit=f;
      printf("Receive ellipticity angle:   ");
      scanf(" %d",&f);
      h->ichir=f;
      printf("Receive orientation angle:   ");
      scanf(" %d",&f);
      h->ipsir=f;

      WAIT;
} /* get_DS_attrs */
```

```
***** FILE: util.c

/*** util.c */

#include "types.h"

void open_file(int argc, char **argv, headertype h)
      {
            char ch, buf[80];

            TOPLINE("Open Data File");
            if(argc<2) {
                  printf("Enter the name of the data file (or pointer file): ");
                  scanf(" %s",h->fname);
                  }
            else strcpy(h->fname,argv[1]);
            h->fp=fopen(h->fname,"r");

            if('#'==(ch=fgetc(h->fp))) {
                  /* look up other file info, open other file */
                  fscanf(h->fp," %s",buf);
                  fclose(h->fp);
                  printf("(opening %s)\n",buf);
                  strcpy(h->fname,buf);
                  if(!(h->fp=fopen(buf,"r"))) {
                        bomb("pointer to illegal file name");
                        exit(-1);
                        }
                  }
            fseek(h->fp,0L,0L);
#ifdef DEBUG
      flag("File position is %d\n",ftell(h->fp));
#endif
      } /* open_file */

void get_loc(headertype h, int dsiz)
/* prompts user for row,col of a pixel of size dsiz and repositions h->fp
accordingly */
{
      int r, c;

      printf("Enter the row and column (r,c) of the element: ");
      scanf(" %d, %d",&r, &c);

      /* seek the file there -- later, check for valid coords */
      if(ftell(h->fp)<0) {
            bomb("File not open");
            return;
            }
      fseek(h->fp,(h->nsamp*r+c)*dsiz+h->boofdr[h->window],0L);

#ifdef DEBUG
      flag("File position is %ld\n",ftell(h->fp));
#endif

} /* get_loc */

void dump_header(headertype h)
/* prints the contents of the header h */
{
      printf("name of disk file -- %s\n",h->fname);
      printf("scene title -- %s\n",h->scene);
```

```
        printf("selected window -- %d\n",h->window);
        printf("record length in bytes -- %d\n",h->rlib);
        printf("number of header records -- %d\n",h->nohr);
        printf("byte offset of old header -- %d\n",h->boooh);
        printf("general scale factor -- %f\n",h->gen_fac);
        printf("polarization angles:\n\tichit -- %d\n",h->ichit);
              printf("\tipsit -- %d\n",h->ipsit);
              printf("\tichir -- %d\n",h->ichir);
              printf("\tipsir -- %d\n",h->ipsir);
        printf("number of lines(records) -- %d\n",h->nline);
        printf("number of pixels(samples) per line -- %d\n",h->nsamp);
        printf("byte-offset of first data record (current window) -- %d\n",
              h->boofdr[h->window]);
        printf("subsection width (current window) -- %d\n",
              h->w[h->window]);
        printf("subsection height (current window) -- %d\n",
              h->h[h->window]);
} /* dump_header */
```

```
***** FILE: cls.c

/*** cls.c ***/
#include "types.h"
#define CLEAR_COMMAND   "echo ------------------"
char *_cls_string;

void clear_screen()
      {
#ifdef CLEAR_COMMAND
      if(_cls_string==(char *)NULL){
            /* create it */
            FILE *tmp;
            char name1[80],com[80], *junk;

            _cls_string=(char *)malloc(80);
            tmpnam(name1);          /* create name of dummy file */
            strcpy(com,CLEAR_COMMAND);
            strcat(com,"> ");
            strcat(com,name1);
            system(com);            /* clear the screen into a file */

            tmp=fopen(name1,"r");
            junk=_cls_string;
            while(EOF!=(*junk++=(unsigned char)fgetc(tmp)));
            *--junk='\0';           /* string terminator */
            fclose(tmp);

            strcpy(com,"rm -f ");
            strcat(com,name1);
            system(com);            /* delete the temp file */
            }

      printf("%s",_cls_string);
#else
      {     /* use for dumb terminals, or non-unix machines */
            int i=0;
            for(;i<40;i++)
                  printf("\n");
      }
#endif
      } /* clear_screen */
```