Visualization Tools: University of New Mexico's Khoros

by Philip J. Mercurio

Pixel, March-April, 1992

At first glance, Khoros appears to be yet another dataflow-oriented visual-programming environment, similar to apE (*Pixel*, November/December 1990, pp. 30-35) and AVS (*Pixel*, July/August 1990, pp. 30-33). Like apE and AVS, Khoros uses icons on a two-dimensional grid to represent data-visualization processes and lines connecting the icons to represent data flow. Two features, however, distinguish Khoros from apE and AVS. First, each Khoros program is designed to be run from the command line, as a stand-alone program with its own graphical user interface, or as a module inside the visual-programming environment. Second, Khoros is an extensive software development environment facilitating software reuse and straightforward mechanisms for incorporating user-contributed modules.

Currently, you can acquire Khoros through the University of New Mexico, which is distributing it free of charge, via anonymous ftp. While it is not as polished or bulletproof as a seasoned commercial product, Khoros's current release, 1.0, seems solid, consistent, learnable, and broad in scope. Billed by UNM as "an open environment for information processing, data visualization, and software development," Khoros runs on a number of Unix platforms, and its graphical-user-interface programs are available under X Windows Release 4.

In addition to Khoros source for general Unix platforms, you can acquire pre-compiled binaries for Sun SPARCstations, Sun 3/260, Silicon Graphics 4D series, and DEC Vax and MIPS architectures (see sidebar). The entire distribution, including source, documentation in both on-line and Postscript versions, and pre-compiled Sun SPARC and SGI binaries, occupies about 230 megabytes on disk, and took me a couple of hours to ftp from UNM.

Khoros consists of five major subsystems: 1) a user interface and software development system, 2) a visual-programming language, 3) a data exchange format, 4) application-specific libraries for data processing, and 5) a visualization toolkit consisting of programs for interactive data display and manipulation.

User Interface Features & Toolkits

Much of Khoros's power and interoperability stems from a design specifying that all applications be built atop libraries of routines for handling a program's interface. The interface specification consists of the application's parameters, usually input and output file names and numerical values, along with a description of the widgets in the graphical user interface: their type, position, labels, color, etc. The interface description resides in a file read at run-time, so you can modify applications without modifying source code.

In all Khoros programs, command-line arguments follow consistent conventions. For example, you can specify an input file for any program with an "-i" flag and an output file with an "-o" flag. A "-V" flag displays an application's version number, while a "-U" flag summarizes its usage. Usage information fills no more than a single screen of text and highlights only the required and important arguments to the program. For more information, you can issue the "vman" command, which works like the Unix "man" command. Vman even supports keyword searches of the manual, making it feasible for you to browse a software package of Khoros's magnitude.

Each Khoros program, if invoked with the "-P" flag, prompts you to type all required parameters and most optional ones. If you also specify the "-A" flag, the program stores its parameters in an answer file, which you can subsequently invoke by specifying the "-a" flag. This strikes me as an ingenious mechanism for handling programs having complicated calling sequences.

All Khoros programs with graphical user interfaces are built atop a library called "forms," which employs the MIT Athena widget set. Although the Athena widgets' appearance and functionality may seem austere next to the now-common Open Look and Motif interface libraries, you should have no problem learning and navigating the interfaces.

Khoros libraries provide a number of features common to all the applications. Program operation is consistent, and portions of the user interface are reused throughout. For example, both the 2- and 3-D plotting programs ("xprism2" and "xprism3") and the image-editing program ("editimage") support overlays of text and figures. Although the programs themselves differ greatly, you can employ a single control panel for interacting with the overlays. And because the user-interface description is stored in a file read at run-time, you can change the appearance of the panel everywhere by modifying a single file.

In addition, the Khoros user-interface library supports the activation of widgets under program control. This feature provides two important benefits. First, it lets you record interactions with an application in a journal file, which you later can play back. (The Khoros team, for example, uses this feature to great advantage by including, in the release tutorial, journal files for each interactive program.) Second, and more important, it lets you distribute the user interface over multiple machines, enabling a kind of groupware capability, wherein users on separate workstations can interact simultaneously with a single application. One application included in Khoros is a program called "concert," which runs any of the other programs as a groupware application. Concert operates in either (1) master-slave mode, where only one user can provide input to an application, while other users watch; or (2) peer-to-peer mode, where any user can provide input to the application at any time. In either mode, the manual strongly recommends that the group coordinate control by maintaining voice communication throughout the session.

Khoros's groupware features can act as a potent aid to both teaching and collaborative research, while its common user-interface libraries allow its applications to be integrated into the visual-programming environment.

Khoros's "forms" library provides the basis of the environment's visualization toolkit. Khoros also provides a "utils" library, containing higher-level user-interface constructs such as file browsers, help screens, error reporting and pop-up menus. On any Khoros panel in which you can type a file name, selecting the label for "widget" brings up a standard panel for browsing through the system to select the desired file.

Khoros also provides a unique keyword system for creating aliases to commonly used files. You can define keywords in a user-specified file via lines such as the following:

```
:ball                KHOROS_HOME/data/images/ball.xv
plot2D:example2.0    KHOROS_HOME/data/plot_functions/example2.0
```

The first line is a simple alias for the full path name of the ball image file. Anywhere Khoros expects a file name, you can give ":ball" as a substitute for the full path name; it works in both the graphical user interface and the command line interface. The second line demonstrates the hierarchical nature of Khoros keywords. For example, a line defining "project2:example2.0," differs from the example2.0 in the plot2d hierarchy. You also can use the standard Khoros file browser to search the keyword hierarchy, a comfortable way to shift focus from the entire file system to the subset of important files defined in your keywords file.

The Khoros help mechanism is also well thought-out. Each form or subform of a Khoros interface contains a help button. Rather than bringing up a copy of the entire manual, as some systems do, Khoros displays help specific to the current context. I've browsed through the help available in a number of the programs, and rarely have I seen more than one or two screenfuls of text presented at a time. When a lot of information is involved, as in the help attached to the main help button for a complex application, Khoros presents a concise overview of the application and a number of buttons. You can use these buttons, for example, to select among the main topics of the application's manual. By letting you focus on topics of interest at a particular moment, this feature helps smooth the learning curve, especially when you are new to the Khoros environment.

Support for Data Exchange

Khoros programs communicate using the "VIFF" file format, which can represent both multidimensional data and general geometric objects. VIFF is supported by Khoros library routines; read/write routines automatically convert byte order and floating-point formats between different architectures. In addition, Khoros provides file-format converters for the following external formats: TIFF, pbm, BIG, DEM, DLG, ELAS, FITS, Matlab, Sun rasterfile, TGA, and xbm.

Data Processing and Visualization Libraries

To help you build a complete application, Khoros provides a number of other libraries. The "graphics" library supports drawing in two and three dimensions, with output to an X Window or Postscript or HPGL device. The "display" library supports the raster-image display, manages color allocation, and provides tools for editing images and their color maps.

Khoros processes data through a set of library routines. Because these routines exist also as stand-alone programs, you can run them from either the command line or the visual-programming environment. Most Khoros programs, called "vroutines," fall into this category and feature a graphical interface only when you use them inside the visual-programming environment. Vroutines operate on point, one-dimensional, two-dimensional, multiband, or N-

dimensional vector data, and are designed to be polymorphic in that they function on bit, byte, short, integer, float and complex data types. Vroutines operate differently depending on the dimensionality of the data.

At program level, you can define a vroutine interface using the specification mechanism described earlier. The Khoros function interface, as admitted in the documentation, is currently not as well defined; however, it does let you easily combine functions from the vroutine library into stand-alone programs.

The vroutine library consists of more than 260 programs, in the following categories: arithmetic, classification, color conversion, data conversion, file-format conversion, feature extraction, frequency filtering, spatial filtering, morphology filtering, geometric manipulation, histogram manipulation, statistics, signal generation, linear operations, segmentation, spectral estimation, subregion, and transforms.

X Windows Applications

The remaining Khoros programs fall into a category called “xvroutines.” When you run them stand-alone, these programs have both command-line and graphical interfaces; you also can employ them in the visual-programming environment. Xprism2 and xprism3 are 2- and 3-D plotting packages. Each features a wide range of plot types: 2-D, discrete, bar graph, polymarker and linemarker for xprism2-and 3-D, scatter, impulse, mesh, horizon, surface, 2- and 3-D contours and color mesh for xprism3. Each supports all Khoros-supported data types and lets you easily rotate, display scale, and translate plots. You also can control all the usual plot features, fonts, colors, axes, etc., along with annotations.

The annotation feature provides object-oriented drawing tools for lines, circles, rectangles, and polygons, as well as text in any X11 font, and is used in other Khoros tools as well. The Khoros plotting tools, which are fairly complete, produce good output on Postscript, Imagen, and LN03 laser printers and HPGL-based plotters, as well as on screen.

For manipulating raster images Khoros provides a tool called “editimage.” You probably will use editimage primarily for modifying an image’s colormap by scrolling around it or selecting a region of interest to manipulate or extract. Editimage’s color-palette tools operate in eight different color spaces (RGB, CMY, HLS, etc.) and display a histogram of each component and a graph, which you can edit, mapping pixel values to component values.

Unlike most Khoros program interfaces, editimage seems difficult to use, the colormap-graph editor requiring a baroque and hard-to-master combination of single and double mouse clicks. On the plus side, the annotation tools and printing options available in the plotting tools are also available in editimage.

You may often use editimage at the output end of a visual program, for exploring the resulting image; you may also seek an easier way to do this. Fortunately, Khoros provides one in its non-interactive display program called “putimage.” Putimage can display changes to an image as the updates are output by programs earlier in the pipeline. Another program, “animate,” accepts a sequence of images and plays them back as animation, with single-step and continuous playback controls.

Khoros also provides a program called “warpimage,” which warps an image, based on user-specified tie points, to a second image. Another image-manipulation tool is “viewimage,” which combines the capabilities of the xprism3 and editimage programs. Viewimage takes a Khoros image file and maps it onto elevation data obtained from another file. It supports all plot types from xprism3, as well as most image-manipulation tools available in editimage.

Visual Programming with Cantata

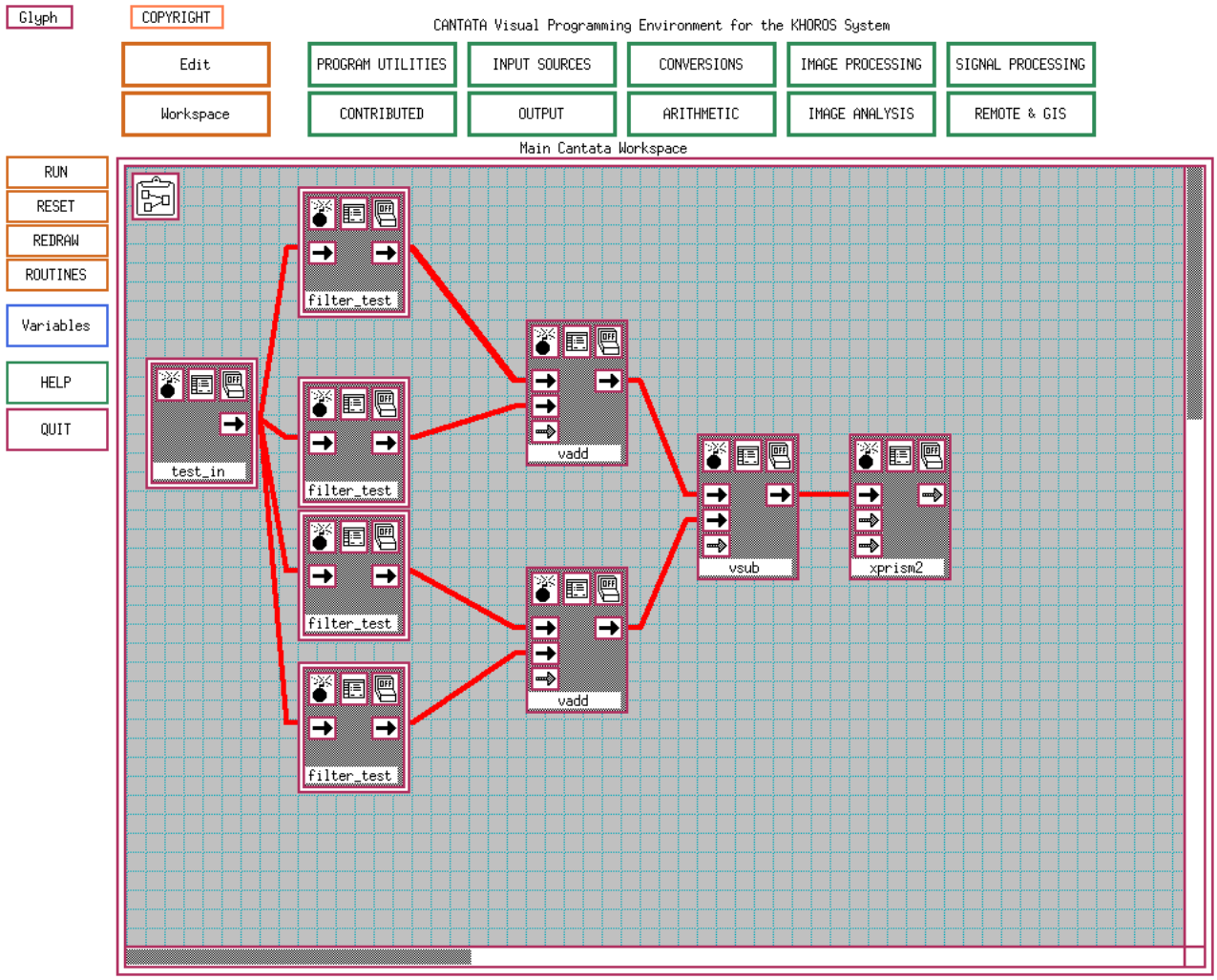

Figure 1

All Khoros components come together in a program called “Cantata,” Khoros’s visual-programming environment. Cantata represents a network of connected vroutines and xvroutines as a dataflow diagram (see Fig 1). Each program in the network is represented as a glyph containing the program’s name, arrows showing the program’s input and output connections, and three iconic buttons along the glyph’s top. The leftmost button, displaying the image of a bomb with a lit fuse, removes the glyph from the network. The middle button, displaying the image of a form, expands the glyph to its full graphical user interface. When the glyph corresponds to an xvroutine, the interface is identical to the one appearing when you run the program stand-alone. For vroutines, the interface contains widgets for specifying the parameters that you would specify on the command line when you run it outside Cantata. The rightmost button is the on/off switch.

In Khoros, you construct dataflow networks by selecting programs from pull-down menus accessed via the green buttons above the Cantata window; you then see the program glyph, which you can place anywhere on the Cantata grid. You construct red flow lines connecting the glyphs by selecting the source glyph’s output arrow and the destination glyph’s input arrow. Black arrows indicate connections that are required, while gray arrows indicate the ones that are optional.

When you run the network, Khoros stores the output of each process in a temporary file, subsequently specified as input for the downstream processes. As you run each command, Khoros echoes the command line you have thus constructed. Each gray glyph turns black as it begins execution and white as it completes it. The output and input

arrows of each successful connection are marked with dots, indicating the creation of a temporary file. Because Cantata links processes via temporary files instead of pipes or sockets, you can easily execute the network manually, one process at a time. This is the purpose of the on/off switch: by flipping the switch, you can execute each glyph in turn. In the default, data-driven mode, you must select glyphs in the correct order to guarantee that the necessary input files have been computed. In demand-driven or responsive-execution mode, you can choose any glyph and Khoros will execute all the upstream glyphs that compute input needed by the glyph you choose.

The drawback of using temporary files occurs when you run networks requiring great amounts of disk access. Cantata attacks this problem by implementing data caching via shared memory. Most of the systems running Khoros do support shared memory, although setting Khoros up to use it requires a little preparation and tuning; the size of the shared memory block, for example, is an important tuning parameter. The cache mechanism is controlled by environment variables and auxiliary programs, and is thus available not only within Cantata but also from the command line and within scripts.

Because it includes glyphs for programming constructs such as loops, if-then-else, and merging datastreams, Cantata is a complete programming environment. It also contains glyphs used to store comments or to execute any Unix command not already part of Khoros.

One common complaint surrounding dataflow-programming environments is that they are complex. Cantata addresses this point through its streamlined design, which makes learning the dataflow paradigm easier, especially if you are already familiar with the concept of connecting Unix programs via pipes or temporary files. It also lets you transfer knowledge in the other direction: you can learn Khoros by starting with Cantata and, from the trail of commands printed as Cantata executes processes, learn to piece the units together into scripts you can run independently of Khoros.

Nonetheless, as a new user you must deal with a vast array of available Khoros routines and the accompanying difficulty in browsing the lists and selecting proper functions. Cantata alleviates this problem somewhat by organizing the programs into a three-level hierarchy. The first level consists of the ten green buttons above the Cantata window. Each button brings up a pop-up menu of no more than seven choices. When you make your choice, Cantata brings up a panel containing widgets selecting a specific glyph and its parameters. For example, to find the “vadd” routine, which sums two inputs, you select the “arithmetic” button, and Khoros displays a menu listing “unary arithmetic,” “binary arithmetic,” “logical operations,” and “matrix algebra.” If you select “binary arithmetic,” Khoros brings up a panel featuring six different binary arithmetic operators. This panel has two help buttons, one providing general help for the binary arithmetic category and summarizing the available operators, the other giving specific help on the selected operator.

Another way in which Cantata helps manage complexity is by letting you collapse entire networks into subroutines and represent them by a single glyph. Each of the “test_in” and “filter_test” glyphs in the digital signal-processing example shown in Figure 1 is in fact a subnetwork consisting of more than a dozen glyphs.

Support for Software Development

It is clear that the designers of Khoros put a lot of thought and planning into its software development capabilities. Additionally, in releasing Khoros for free distribution, the University of New Mexico has provided not only a well-crafted suite of programs and libraries but also a good deal of support for users who want to extend Khoros and integrate their contributions for distribution to the rest of the Khoros community.

To build a new Khoros vroutine or xvroutine, you start with “composer,” the environment’s interactive graphical user-interface editor. Although it is not as sophisticated as commercial user-interface development environments, composer lets you interactively design the panes of a user interface with control over the types, positions, and geometries of the application’s widgets. One result of a composer session is the user-interface specification read at run time by the application or by Cantata.

Composer also operates as a computer-aided software engineering tool, assisting you with structure. Composer’s program-specification portion consists primarily of about two dozen buttons, each selecting a portion of program code or documentation for editing. The details of integrating a new routine into Khoros are hidden, so when you are programming you need to deal with only those portions of code that actually perform work.

Once the user-interface and program specifications are complete you can use “conductor” to generate the source code for a graphical-user-interface program (xvroutine) or “ghostwriter” to generate the source code for a command-line program (vroutine). Both programs write the source code for the library routine and the program, the library and program manual pages, and the makefile compiling the code. (In fact, Khoros builds atop the “imake” program, which is part of the X Windows release, so it writes the appropriate “imakefile.”) Khoros also provides “kinstall,” a source-code configuration and management tool capable, among other things, of executing the appropriate “makes” on remote machines to create the binaries for several different architectures in one step. You can execute all these tools—conductor, ghostwriter, and kinstall—via buttons within composer, and, once you have correctly configured the Khoros system, you can design, code, maintain, compile, and install new routines also all from within composer.

Performance

I ran all my Khoros 1.0 tests on a 16 Mbyte Sun SPARCStation 1, running Sun’s Open Windows X environment. By downloading the Sun binaries directly from UNM’s anonymous ftp site, I avoided compiling the entire release; all operated reasonably. Performance, especially within Cantata, was far superior to that of version 0.9. As for the Cantata editor itself, I tested selection, placement, and hook-up of glyphs, and found it uniformly responsive, even within networks containing dozens of glyphs.

My configuration had the entire Khoros release stored on a remote machine and accessed via NFS, but used my workstation’s local disk for storage of the temporary files used for linking modules. As a result, performance was limited more by the time needed to load and execute each program than by operation of the modules or communication between them.

For example, the digital signal-processing network shown in Figure 1, which generates signals and performs numerous operations on them to produce the plot shown in Figure 2, consists of over 60 nodes and takes 59 seconds to run from start to finish. The last 12 seconds is spent executing the xprism2 program, the network’s only glyph that produces output. Running the same network with a 1 Mbyte shared-memory cache did not reduce the execution time significantly, since more disk I/O time was spent loading the programs than connecting between them.

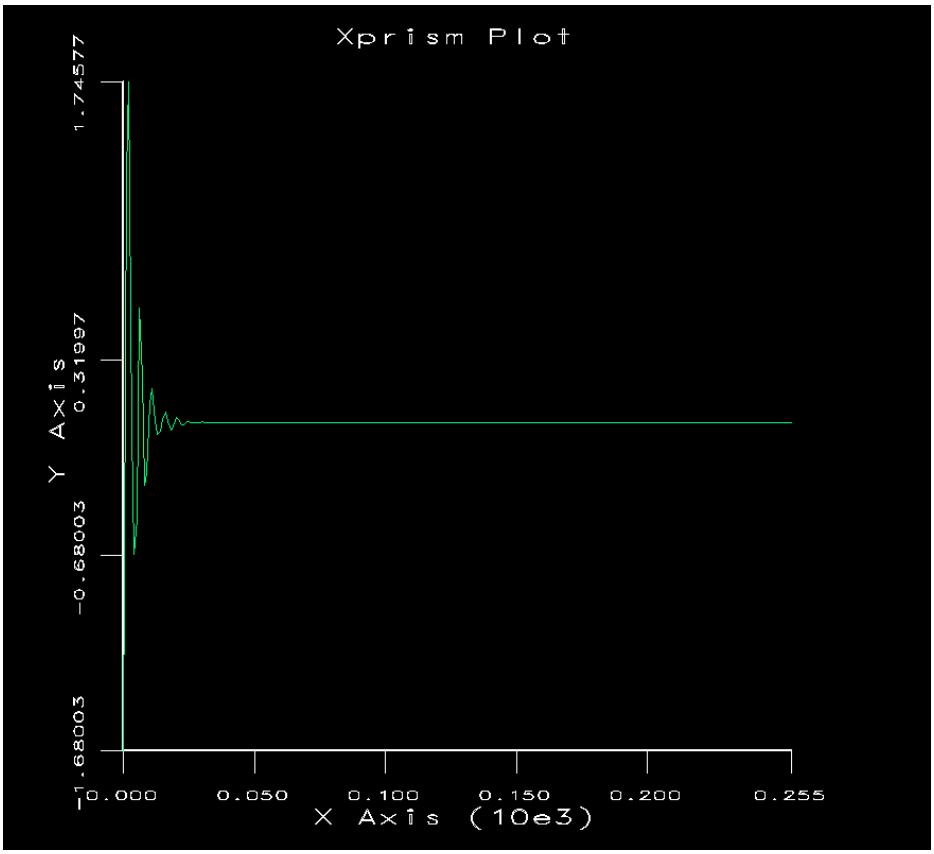

Figure 2

While Khoros is reasonably mature, small bugs and design flaws are scattered throughout the package. The colormap-editor problems made for difficult operation, and bugs in the interaction between concert and Cantata pushed a couple of Cantata-display sessions running on separate workstations out of sync. In fairness, the Khoros team is aware of the concert-Cantata problem and mentions it in the documentation.

For the most part, however, I had no problems exploring Khoros. The Khoros team maintains an electronic mailing list for reporting bugs and exchanging information between users, and through it I found a steady stream of useful information and work-arounds.

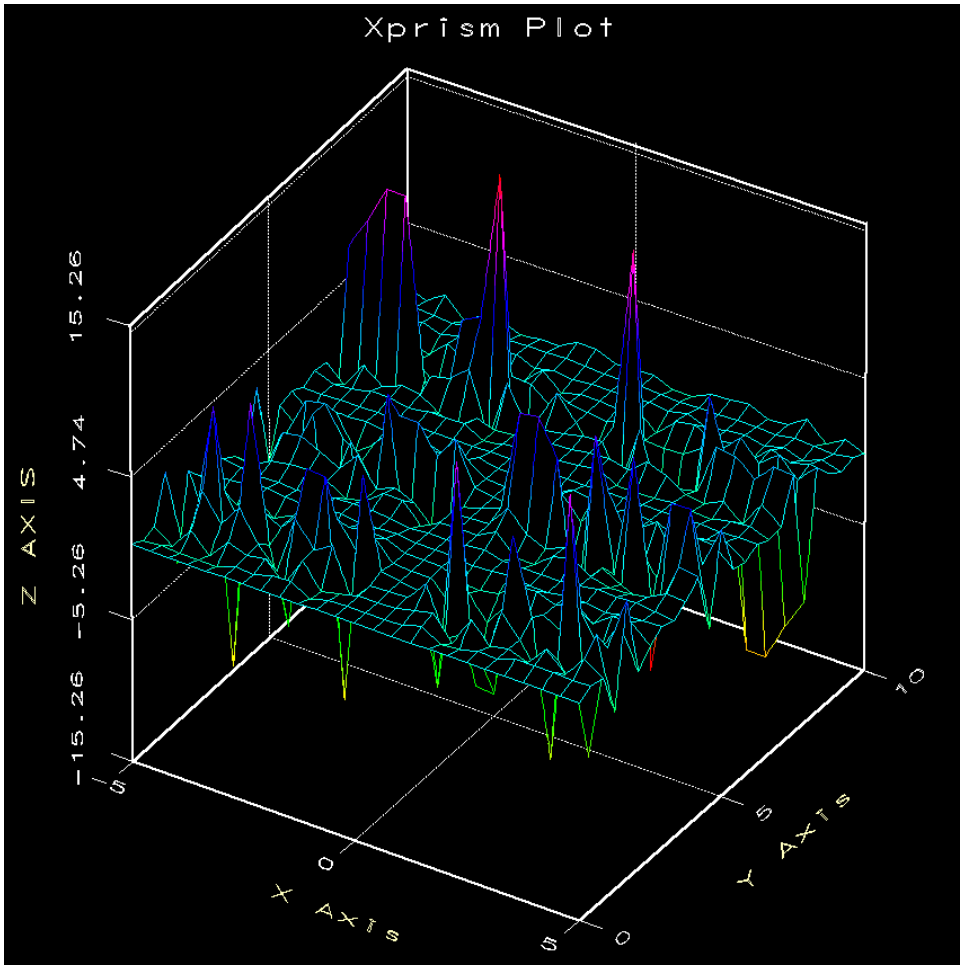

Figure 3

Evaluation

When considering a package as extensive and extendable as Khoros, you must consider both what *can* be done with the system and what *has* been done. With Cantata, users have constructed visual interfaces for the LINPACK/EISPACK libraries, an image-processing library, a digital signal-processing library, and a remote sensing/geographical information system. Each of these domains has been dealt with exhaustively.

Yet Khoros is not a complete scientific visualization toolkit. While it strongly supports image processing, it does not support either surface or volume rendering. Khoros also lacks tools for interactively viewing three-dimensional objects and for computing such common visualization tools as isosurfaces or particle trajectories in a fluid flow.

Nevertheless, Khoros is an excellent environment for developing such packages, and its no-cost availability to the scientific visualization community promises imminent and continuing creation of more tools. Research labs looking to develop environments specific to their needs would do well to consider building on top of Khoros rather than starting from scratch.

“If I have seen further,” said Newton, “it is by standing on the shoulders of giants.” Members of the Khoros team have demonstrated that this principle holds doubly for software development, and by contributing this gigantic effort to the research community, they have, hopefully, raised all our sights a little higher.