THE KHOROS SOFTWARE DEVELOPMENT ENVIRONMENT FOR IMAGE AND SIGNAL PROCESSING

Konstantinos Konstantinides and John R. Rasure †

ABSTRACT

Data flow visual language systems allow users to graphically create a block diagram of their applications and interactively control input, output, and system variables. Khoros is an integrated software development environment for information processing and visualization. It is particularly attractive for image processing because of its rich collection of tools for image and digital signal processing. This paper presents a general overview of Khoros with emphasis on its image processing and DSP tools. Various examples are presented and the future direction of Khoros is discussed.

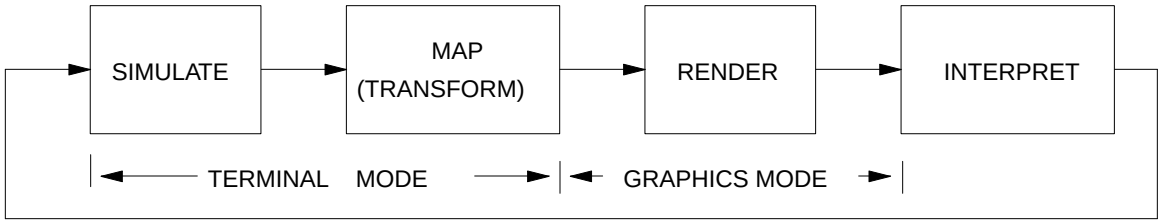
Internal Accession Date Only

---

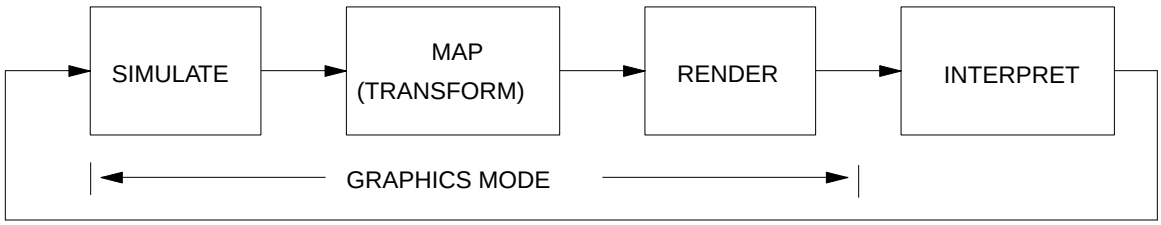
† K. Konstantinides is with Hewlett-Packard Laboratories, J. Rasure is with the Dept. of Electrical Engr., Univ. of New Mexico Albuquerque.

1. Introduction

The repetition of a simulation process with different parameters until a satisfactory solution is found, known as iterative processing, is a key element of the scientific process. For example, in image processing of noisy satellite data, one may iterate through the use of various image filtering kernels before the received images can be considered noise-free and can be processed with other imaging techniques, such as image enhancement or pattern recognition



a)



b)

Fig. 1: Iterative scientific processing. a) Traditional approach, b) Using an application builder.

algorithms. Fig. 1a shows a block diagram of a traditional iterative scientific processing. It consists of four main steps: (a) *computer simulation*, which may also include data input and data processing, (b) *mapping*, where the simulation data are transformed into a displayable form, (for example, data may be mapped to a color palette) (c) *rendering*, where the data are plotted or displayed on a computer screen, and (d) *interpretation.* In a traditional setting, simulation and mapping are executed in "batch" mode on a main-frame or super-computer using a terminal interface and only the rendering step is performed interactively on a graphics terminal or workstation. Data renderers, such as Plot3D [1] , and the Personal Visualizer [2] provide great flexibility in viewing data, but have no control on the simulation and mapping process.

Advances in workstations, graphics, and visual languages now allow for the complete scientific cycle to be executed interactively in "graphics" mode (Fig. 1b). The user can visually program and control both the data viewing parameters and the system variables. Such visual programming environments are usually called Application Builders. In general, an application builder is a visual environment that allows a user to: (a) create a block diagram of his application by graphically connecting system "blocks," (b) graphically control input, output and simulation variables, and (c) visualize the data. Examples of application builders include apE [3] [4] and AVS [5] for the processing of volumetric data (for example, data generated from simulations in fluid dynamics), HP VEE for instrument control and data processing [6], and Gabriel [7] for data communication and signal processing.

In this paper we present a general overview of Khoros, a visual environment developed at the University of New Mexico, with special emphasis on its tools for image and signal processing. Khoros was originally developed for research in image processing, but now it is being used as a research and development tool in a variety of scientific applications, including geographical imaging systems (GIS), medical imaging, and distributed processing. Because of its rich collection of signal and image processing routines, Khoros is particularly attractive as a prototyping tool in research in image processing.

Section two presents an overview of Khoros and its tools for image processing and analysis. The Khoros tools for digital signal processing are discussed in section three. Other tools in the Khoros environment are presented in section four. We conclude with a short discussion on the future features of Khoros and a summary.

2. Khoros Overview

Khoros is an integrated software development environment for information processing and visualization. Khoros has been under development at the Department of Electrical and Computer Engineering at the University of New Mexico since 1987. It includes a visual programming language (*cantata*) [8], code generators for extending the visual language and adding new application packages to the system, an interactive user interface editor, interactive image display programs, surface visualization, an extensive library (over 260 routines) of image processing, numerical analysis, and signal processing routines, and 2D/3D plotting packages. Besides its use in research in image processing, pattern recognition, GIS, and other related fields, Khoros may also be a valuable tool for teaching image and signal processing.

2.1 Cantata

Fig. 2 shows a snapshot of *cantata*, the visual programming environment for the Khoros system. Cantata is a general purpose programming language based on the data flow paradigm. It includes support for conditionals, iteration, and subprocedures. Fig. 2 shows a block diagram of a simple image processing application and output results at various stages of execution. Each element of the block diagram is called a *glyph*. Glyphs are placed on the cantata

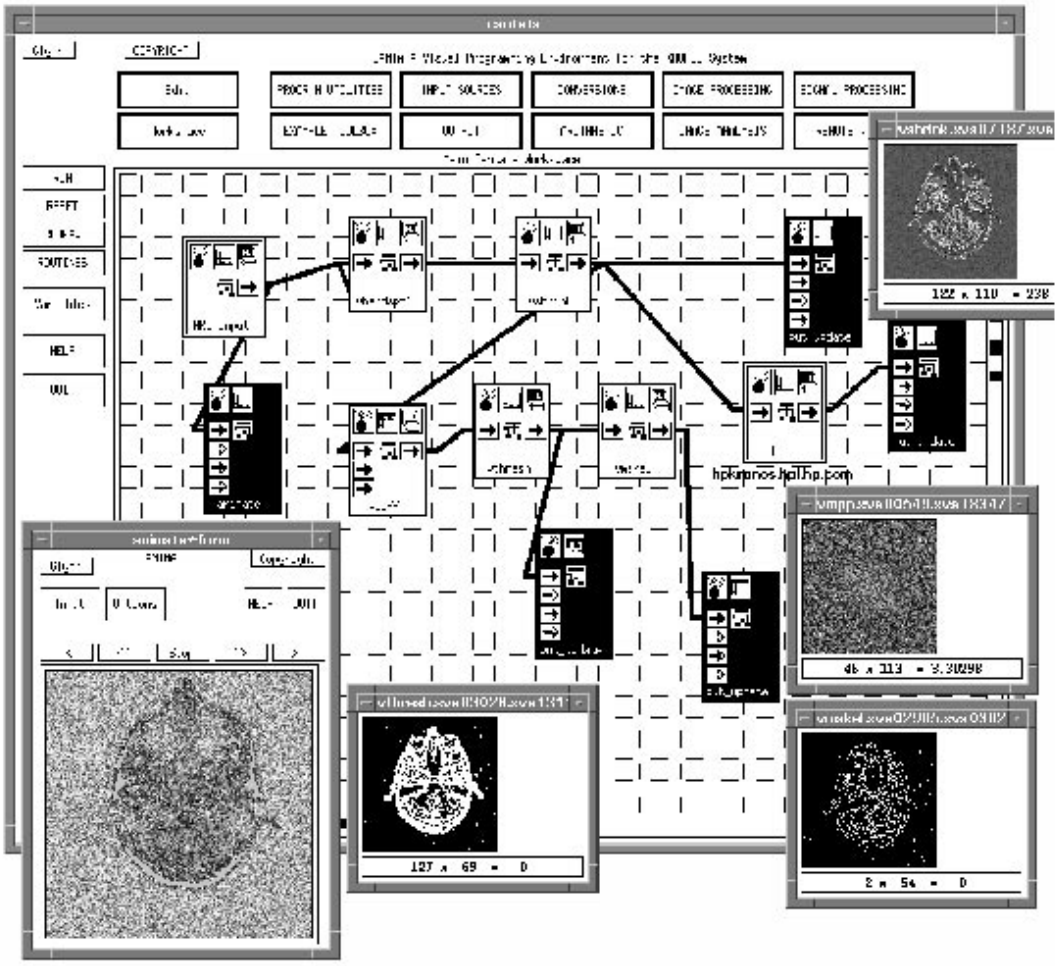


Fig. 2: Block diagram of an image processing application in Khoros.

canvas by selecting the appropriate routines either from the top menus (INPUT SOURCES,

OUTPUT, IMAGE PROCESSING, etc.), or from a list of all available routines from the ROUTINES menu at the left of the *canvas.* Glyphs are connected together by clicking at the appropriate input/output glyph arrows.

In this example on edge detection, input (from *MRI_input*) is a multiband MRI image, where each band represents a slice of the MRI scan of a human head. Glyph *vbandspt1* selects one of the MRI slices. That image is subsampled using the *vshrink* routine and is displayed with the *put_update* glyph. (Subsampling is not necessary in edge-detection, but it is done here so that all output images can fit nicely on the screen.) After differentiation (*vdiff*) and thresholding (*vthresh*), the edges of the input slice are displayed again using *put_update*. Glyph *vmskel* computes the skeleton of its input so that a refined set of edges can be obtained. The *FFT* glyph computes the logarithm of the norm of the 2-D FFT of the subsampled scanned image. The original multi-band image is also input to the *animate* glyph. The *animate* routine can be used to either browse through the various bands of the input image or to animate them. Glyphs communicate via temporary files or shared memory, depending on the system architecture and the user's preference. The entire workspace, the data flow, and intermediate results can be saved and restored for later use.

From Fig. 2, on top of each glyph there are three iconic buttons. A click on the *bomb* icon removes the glyph from the cantata canvas. A click on the *on-off* switch, turns on/off that application, and a click on the middle (text) icon opens a window that displays information and control options for that glyph. The tree-like icon, in the middle of a glyph, appears only if distributed processing is enabled. A click on that icon allows the user to specify the machine that will execute that glyph. For example, in Fig. 2, module *FFT* will be executed on host hpkronos.hpl.hp.com. The rest of the glyphs will be executed on the local host.

2.2 Glyph Control

A major difference and advantage of Khoros from other application builders such as apE and AVS is that a single workspace is used for both data flow and module control. This feature is particularly attractive in cases were many similar modules are used. When a separate workspace is used for control, it is often difficult to remember the correspondence among control and execution modules. In Khoros, the control panel (or graphical user interface (GUI)) for each glyph is enabled by clicking on the middle "text" icon. For example, Fig. 3 shows the control panels of the threshold glyph *vthresh*. The threshold routine is part of the *segmentation* set of image processing routines. A list of all available routines in this set is shown in Fig. 3. For the threshold operation, one can control the lower and upper threshold levels (set here at 128 and 255 respectively), and can define the value of the non-zero Pixel level. On-line help is provided via the "HELP" buttons.

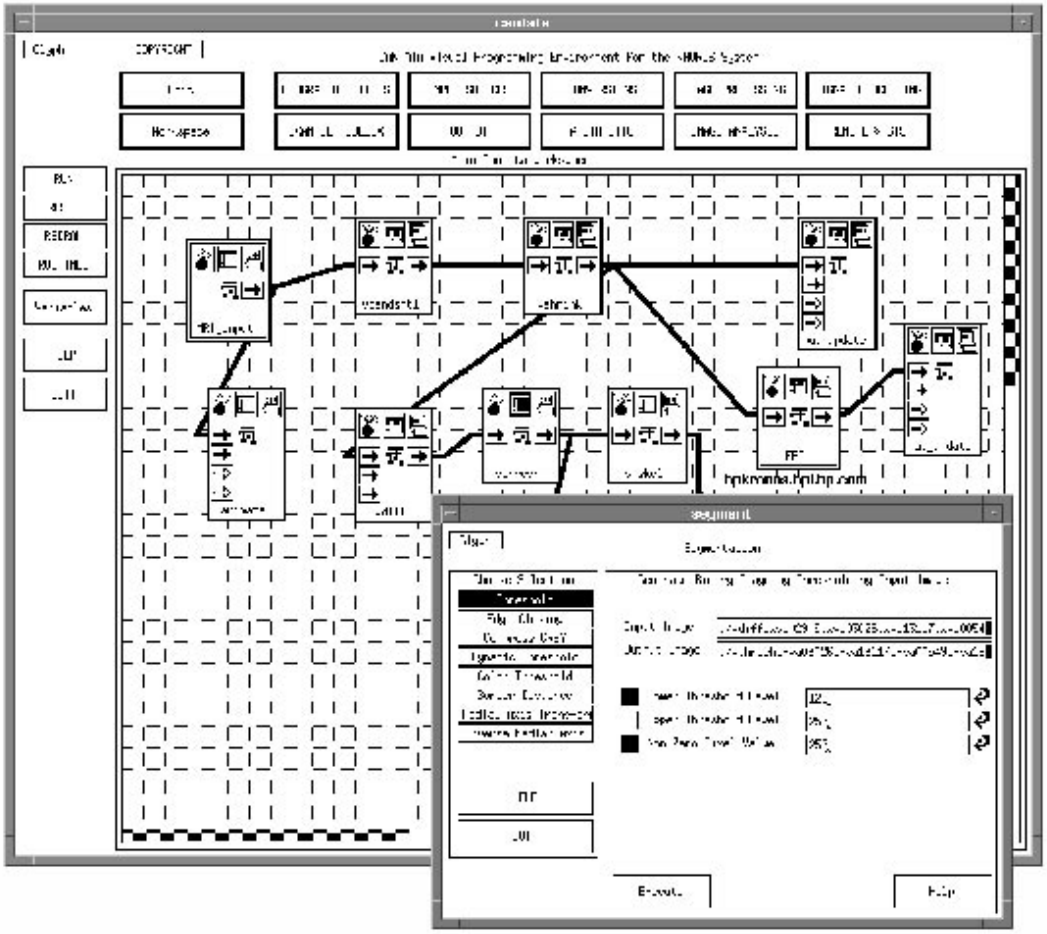


Fig. 3: Graphical user interface for the threshold glyph.

2.3 Data Types

Khoros uses a unified data format for all its modules [9] (Vol. II, Chapter 1). A single format facilitates data interchange among the Khoros routines and the development of new modules. The Khoros format (VIFF) includes information for an application to properly interpret the data and perform basic error checking. The VIFF data structure is used for both 1-D and 2-D data processing and the visualization of 3-D data. It has a 1024 byte header, followed by (optional) map and location data, and the image data. The header provides information on data storage and interpretation, data location (implicit or explicit), and the color space. The VIFF header is extensible for future applications such as multi-dimensional data processing and 3-D data rendering. Because of the wide range of existing image data formats (i.e, GIF, TIFF, raster, PBM, etc.) Khoros includes many data conversion routines for easier processing of existing data.

2.4 Program Hierarchy

Cantata allows an hierarchy of workspaces so that the visual complexity of large data flow graphs can be reduced. Multiple glyphs can be combined into another cantata workspace. Then that workspace can be used as a regular cantata glyph. Fig. 4 shows the structure of the *MRI_input* glyph used in Fig. 2. It consists of two *input* modules and the *count_loop* and *vbandcomb* modules. The purpose of this sub-procedure is to combine a set of MRI scanned images into a single "multiband" image, where each band represents an MRI slice. This

This figure is not currently available!

Fig. 4: Block diagram of the MRI_input subprocedure.

example also shows the use of global variables and loop control within cantata. Module *vbandcomb* combines two (or more) input images into a single image. The output image has as many image bands as the sum of bands in its inputs. Denote by head.*i*.img (*i*=0, 1, 2, ..., N), the i-th slice of the MRI scan. The flow diagram in Fig. 4 evaluates the following loop:

```
vbandcomb -i1 head.0.img -i2 head.1.img -o head.out.img
for i=2 to N do
  vbandcomb -i1 head.i.img -i2 head.out.img -o head.out.img
done
```

In Fig. 4, one of the input glyphs (the input to *count_loop*) points to head.0.img and the other to head.*i*.img. The final output is available either from the output of *vbandcomb* or from the top right output of *count_loop*.

2.5 Execution Scheduling

A visual language process is analogous to an operating system process. It is a running visual program with a state and data. The functionality and the execution of the operators, and the characteristics of the data that an operator works on vary dramatically from one visual language system to the next. The differences can be attributed to the fact that each system is trying to optimize the visual language process for a specific application domain or computing architecture.

In cantata, there are three separate phases of the process that are re-evaluated continuously: translation, scheduling, and dispatching. The translation step is an interpretation of the visible network of connections and glyphs into a recursive netlist. The netlist contains such information as whether a glyph is a source or a sink, whether valid data is available at the input of a glyph, and whether the parameters have been modified. The netlist information is used as input into the scheduler. The scheduler can be run in a data driven or demand driven (responsive) mode. In either case, glyphs are scheduled if either their parameters have been modified or data at their inputs is invalid. All glyphs that are scheduled are then dispatched as local or remote Unix Processes †. The dispatcher uses input from the visual language to determine both the location of execution and the method of data transport (file, socket, shared memory).

2.6 Tools for Image Processing

Khoros provides library routines or executables for a large range of image processing applications, including basic pixel-arithmetic, low-level image processing, and image analysis. Routines are available either as "stand-alone" programs that can be used from a terminal or in cantata, or as library calls that can be linked by the user during the development of new code. Table 1 shows a partial list of the routines available now for image processing. These routines are divided into three main classes: Arithmetic, Image Processing, and Image Analysis. The Arithmetic class includes all the pixel-level arithmetic and logical operations. The core of image processing routines (filtering, geometric manipulation, and transforms) is in the Image Processing class. Finally, the Image Analysis class includes various routines in segmentation, feature extraction and classification.

3. Tools for Digital Signal Processing

Although Khoros was originally developed as an image processing environment, it has a rich selection of tools for digital signal processing. Hence, it can also be used as a powerful prototyping tool for DSP applications. Fig. 5 shows a cantata environment with a simple filtering application. A sinusoid signal (generated in *dgsin*) and white Gaussian noise (generated in *dggauss*) are added together in *vadd*. The noisy signal is passed through a low pass filter (*dfilter*) designed in *dfiltlp*. Finally, the noisy and filtered signals are displayed in both the time and frequency domains using *xprism2*. The frequency response of the signals are evaluated in the composite *FFt-norm* modules. Each *FFT-norm* module consists of the *dfft1d* glyph, which computes a 1-D complex FFT, followed by the *dmpp* glyph which evaluates the logarithm of the norm of its input data.

The displayed plots show the outputs of this flow graph for a low-pass Butterworth filter with cutoff frequency at 0.1 Hz and rejection frequency at 0.4 Hz. The left plot shows the noisy

---

† Unix is a registered trademark of AT&T.

Table 1
Khoros Routines for Image Processing

ARITHMETIC	
Unary Arithmetic	Scale, Normalize, Invert, Clip, etc.
Binary arithmetic	Add, Subtract, Multiply, Blend, etc.
Logical Operations	And, Or, Xor, Shift.
IMAGE PROCESSING	
Spatial Filters	Sobel, Median, 2-D convolution, Edge Extraction, etc.
Morphology	Erosion, Dilation, Skeletonization, etc.
Transforms	FFT, Hadamard.
Frequency Filters	LPF, BPF, HPF, Band-Reject, Inverse, Wiener Restoration, etc.
Geometric Manipulation	Shrink, Rotate, Transpose, Interactive image warping, etc.
Subregion	Extract, Insert, Pad, etc.
IMAGE ANALYSIS	
Segmentation	Threshold, Medial Axis Transf., etc.
Feature Extraction	Shape analysis, Region Matching, Fractal Analysis, Texture Extraction, etc.
Classification	K-means, Labeling, LRF-classifier, etc.

(dotted line) and filtered data (solid line) in the time domain, and the right plot shows the corresponding data in the frequency domain. The Z-transform of the low-pass filter is computed by the filter-design glyph *dfiltlp*. In designing the filter, the user needs to provide the type of the filter (Butterworth, Chebychev I or II), the sampling frequency, the cutoff and rejection frequencies, and the passband and stopband gains. Similar glyphs also exist for band-pass and high-pass filters. Table 2 shows a partial list of the available Khoros routines for DSP. The list includes data input generation routines, transforms, filter design tools, DSP operations, and matrix operations.

3.1 Interaction

The major advantage of application builders is the ability to easily view data and interact at different stages of the data flow graph, and the ability to selectively add, delete, and control modules. Thus, one can experiment with any part of the iterative process. For example, in the flow graph of Fig. 5 one can change the filter design type (in *dfiltlp*) from Butterworth to Chebychev with a click of the mouse, and then see the updated plots. Designs could even be compared side by side. Similarly, one could change the input signal or noise parameters and

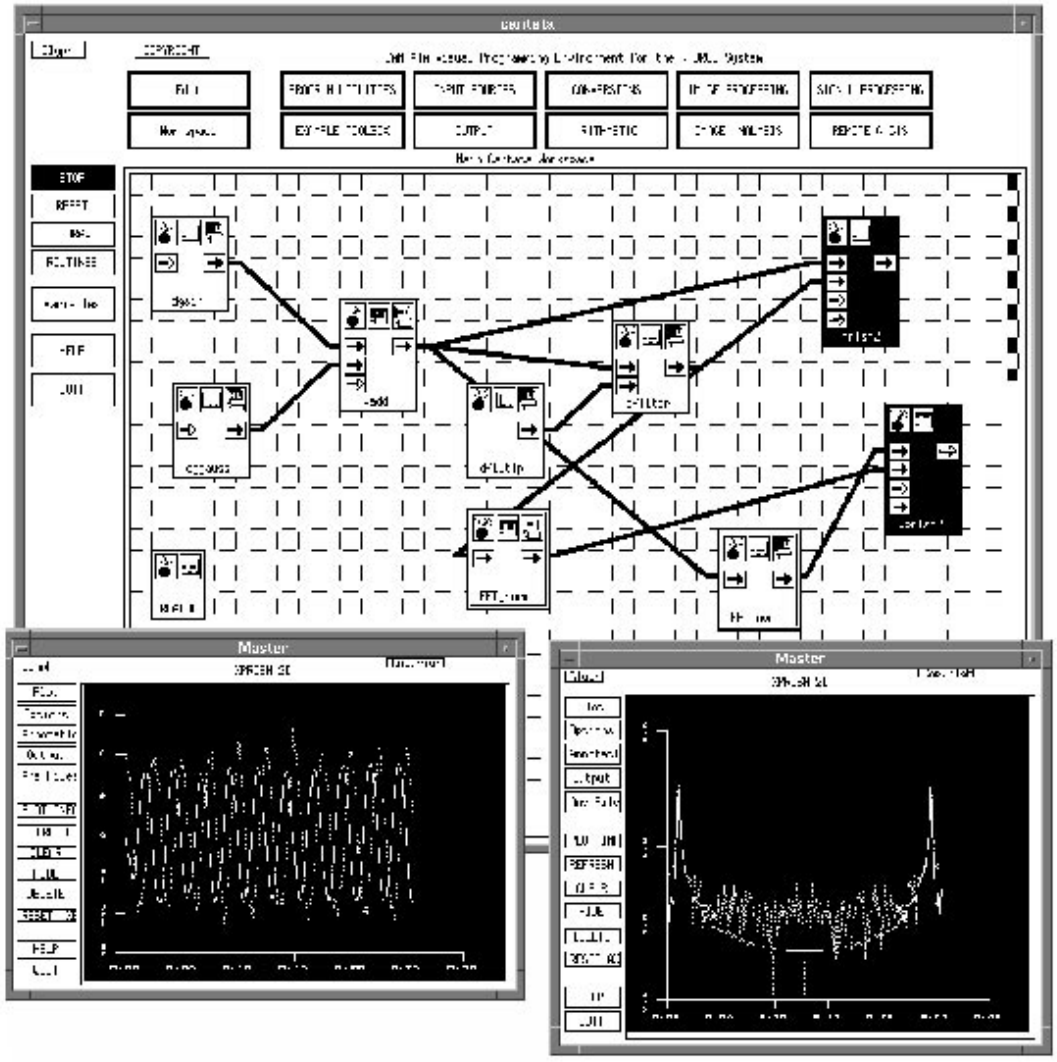


Fig. 5: Block diagram of a DSP application in Khoros.

evaluate the performance of a specific filter design.

Table 2
Khoros Routines for DSP

INPUT	Sinusoid, Pulse, Noise (Gaussian, Rayleigh)
MODIFY SEQUENCE	Normalize, Subsample, Scale, Window, Extract, Insert, etc.
TRANSFORMS	FFT/IFFT, Hartley, Hadamard.
FILTERS	LMS Transversal, LMS Lattice, Butterworth, Chebychev I or II, etc.
DSP OPERATIONS	Autocorrelation, Cross-Correlation, Spectral Estimation, etc.
MATRIX OPERATIONS	SVD, LU Eigenvalues/Eigenvectors, etc.

4. Companion Utilities

In addition to the programs for image and digital signal processing, Khoros provides many other interactive tools. *Editimage* is an interactive image editor, and can be used to extract information on an image, edit it, change its colormap, or annotate it with text and graphics. With *Viewimage* one can combine imagery data with elevation data to create a 3-D image of a terrain. This module is particularly useful in GIS applications.

In many applications, such as the evaluation of medical images by doctors located at different sites, it is important that the users of a visualization system can share and interact on the same data simultaneously. Khoros allows that capability through *Concert*. All Khoros interactive applications inherit the ability to broadcast and receive user events. Concert is the program that coordinates multiple copies of the Khoros application. For example, the command `concert -command editimage -d2 remote_machine.company:0` will start two copies of *editimage*, one locally and one remote. However, both the local and the remote users will have equal control of each others screens. Using concert, the X events that correspond to a user event, such as a button click or the motion of a scroll bar, are broadcast to all the other copies of the application that concert started. The result is that the local copy of the application can receive events from both the local and the remote user.

The same X event broadcasting technique can be used to create journal files of a user session. Instead of broadcasting to another application, the events are sent to a file. This file can then be used as the source of user events for an automatic playback of the recorded session.

As mentioned before, Khoros is not just a visual programming and data visualization environment, but a complete development system. Regardless of the number of available

modules in any visualization system, users always want to incorporate their own versions of existing modules or new ones. Khoros provides support for both the generation of code and the user interface for new modules. *Composer* is an interactive editor of User Interface Specification (UIS). It allows the user to interactively customize the user interface of an application [10]. Finally, *Ghostwriter* is a program development toolkit that allows the user to integrate the user interface created by Composer, new code and documentation, and code from the Khoros libraries, into a new module and into the Khoros environment.

5. Future Directions of Khoros

There is much excitement about the capabilities of Visual Development Environments both for PC's and Workstations [11], [12]. However, the popular view of visual development is really only a small and incomplete component of the overall software development process. One main focus of the Khoros project is to merge two currently separate visual programming paradigms: the direct manipulation of the graphical user interface development and the data flow visual programming (cantata). This merger will create a more complete visual programming environment and in some cases will allow an end-user to write complete programs without using any textual (i.e. C) code. This visual programming environment will be augmented by visual CASE tools for program maintenance and installation and will be integrated into existing software development products such as those offered by CenterLine and Hewlett-Packard.

Another emphasis of Khoros is distributed computing. The applications that are built using Khoros tools should run as a network application, utilizing resources as appropriate. The concert program and the distributed computing capability in cantata are examples of this. In general, the intention is to broaden the applicability of Khoros as a software environment for data processing and visualization.

6. Summary and Conclusions

This paper presented a general overview of Khoros, with emphasis on its tools for image and DSP processing. Khoros is a very powerful prototyping tool for a variety of applications. It is being provided free to the public, as an Open System, via anonymous ftp, and it is being supported on a large array of hardware platforms, from workstations to super-computers. Its tools allow visual data-flow programming, data visualization, and the easy integration of new modules. The current version (1.5) lacks the 3-D rendering capabilities of other application builders (i.e. apE, AVS), but compensates with its excellent support for a large number of image processing functions. Present workstations have excellent graphics capabilities, but they may lack the processing power required for compute-intensive applications. The Khoros capability for distributed processing allows users to interface to a computer network of (probably) heterogenious machines under a single visual environment.

7. Acknowledgments

We would like to acknowledge that Khoros is the outcome of an on going team effort by the Khoros development group at the University of New Mexico.

REFERENCES

1. State of the Art in Data Visualization, ACM SIGGRAPH '89 Course notes.
2. *The Personal Visualizer User's Guide*, Hewlett-Packard, 1990.
3. D. S. Dyer, "A dataflow toolkit for visualization," IEEE Comp. Graphics and Applications, Vol. 10, No. 4, pp. 60-69, July 1990.
4. M. V. Wettering, "apE 2.0," Pixel, pp. 30-35, Nov./Dec. 1990, Pixel Communications, Inc.
5. C. Upson et al., "The Application Visualization System: A Computational Environment for Scientific Visualization," IEEE Computer Graphics and Applications, pp. 30-42, July 1989.
6. *HP VEE-Engine and HP VEE-Test Reference*, Hewlett-Packard, P.N. E2100-9003, Apr. 91.
7. E. A. Lee, W-H. Ho, E. E. Goei, J. C. Bier, and S. Bhattacharyya, "Gabriel: A design environment for DSP," IEEE Trans. on ASSP, Vol. 37, No. 11, pp. 1751-1762, Nov. 1989.
8. J. Rasure and C. Williams, "An integrated data flow visual language and software environment," Journal of Visual Languages and Computing, Vol. 2, No. 3, Sept. 91, Academic Press.
9. *Khoros Programmer's Manual*, Univ. of New Mexico, 1992.
10. D. Argiro and J. Rasure, "An X windows based application programming system," Xhibition 92: A window on distributed computing, San Jose, 1992.
11. Charles Petzold, "The Visual Development Environment: More than a pretty face?", PC Magazine, pp. 195-236, June 16, 1992.
12. Steven Mikes, "Visual Programming Tools for X," X Journal, Vol. 1, No. 5, pp. 50-62, 1992.