The following paper was originally published in the
Proceedings of the Fourth USENIX Tcl/Tk Workshop
Monterey, CA, July 10-13, 1996

Hypertools in image and volume visualization

Pierre-Louis Bossart
Lawrence Livermore National Laboratory

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

Hypertools in image and volume visualization

Pierre-Louis Bossart

Lawrence Livermore National Laboratory
Non-Destructive Evaluation Section
L-416, 7000 East Avenue
Livermore, CA 94550, USA
Phone: 510-423-9350 Fax: 510-422-7819 E-mail: bossart@redhook.llnl.gov

Abstract

This paper describes our experience in image and volume visualization at the Lawrence Livermore National Laboratory. After an introduction on visualization issues, we present a new software approach to the analysis and visualization of images and volumes. The efficiency of the visualization process is improved by letting the user combine small and reusable applications by the means of a machine-independent interpreted language such as Tcl/Tk. These *hypertools* can communicate with each other over a network, which has a direct impact on the design of graphical interfaces. We first describe the implementation of a flexible gray-scale image widget that can handle large data sets, provides complete control of the color palette and allows for manual and semi-interactive segmentation. This visualization tool can be embedded in a data-flow image processing environment to assess the quality of acquisition, preprocessing and filtering of raw data. This approach combines the simplicity of visual programming with the power of a high-level interpreted language. We show how hypertools can be used in surface and volume rendering and how they increase the interaction efficiency by performing complex or tedious tasks automatically. One biomedical application is presented.

1 Introduction

Visualizing images or volumes helps extract qualitative information and quantitative measurements from raw data sets. Visualization software is thus becoming critical in virtually every domain of engineering. However, despite the number of packages available from commercial vendors and from the public domain, it is extremely difficult to find a package that fulfills the needs of a research laboratory dealing with large data sets. Indeed, all the packages are implemented based on the following scheme.

- First, the file formats need to be decoded, and the raw data read in. The data can then be formatted or extracted. This step includes operation such as subsampling, interpolation or dimensional reduction, e.g when a slice is extracted from a 3D volume.
- Next, the data are normalized, typically to 8- or 16-bit integers, and displayed in a window after the colors have been allocated. Using predefined color look-up tables, stretching the histogram, reducing the image dynamic range and providing a colorbar help identify the relevant features of the data set.
- Numerical values, extracted data or regions of interest can then be written back into a file.
- At any point, the user may interact with a graphical user interface (GUI) or issues commands to an interpreter, e.g. to change the color look-up table, look at pixel values, etc.

Practical experiments showed that none of the all command-line oriented (VIEW, IDL, PWave, Matlab, etc), dataflow-oriented (AVS, Explorer,..) or "self-contained" (Analyze, 3DVIEWNIX, etc) visualization packages could handle large data sets whose size exceed both the size of the memory and swap space. The user is thus compelled to manually extract smaller pieces of the data sets, which can be time-consuming and inefficient. It is thus mandatory to tightly link the data extraction and data visualization steps, typically by reading and visualizing one slice at a time instead of loading a 3D volume in memory before visualizing its 2D slices.

The efficiency of interactive visualization is also limited by the GUI design, which can almost never be customized by the user. Even when the source code is available, adding or removing features is very difficult if not impossible, as the GUIs are implemented as monoliths of hundreds of thousands of lines. However, the user needs to control the way the colors are allocated, e.g., interactive thresholds, linear and non-linear colormaps. Similarly, extracting profiles, histograms or non-rectangular regions of interest make quantitative measurements possible. Off-line data transforms or extractions are in our opinion too cumbersome in a research environment. In addition, unsupervised automatic segmentation performs poorly when the data are noisy. The alternative, that is the use of manual and semi-automatic segmentation techniques, is however limited by the lack of flexibility of most graphical interfaces.

The considerations above led us to implement a new visualization tool, geared to large data sets. Since we could not afford, nor had the experience required to write a self-contained application in X-Motif, our approach was to divide the GUI into small reusable components by relying on Tcl/Tk. Tcl/Tk is now used by thousands of users in every domain of graphical and engineering applications. Indeed, Tcl/Tk provides simple ways to "glue" different modules together and it can be extended easily, in contrast to other GUI builders.

However, the use of Tcl/Tk in the signal and image processing community is scarce. Two main reasons can explain this situation. First, efficient data management mechanisms and number-crunching capabilities in image processing are generally believed to require high-performance languages, in contrast to Tcl which only handles character strings. Next, the photo image widget was not designed for interactive visualization and its flexibility is very limited, mainly because image processing applications are not the main focus of Tcl/Tk developers.

In this paper, we describe the implementation of a new gray-scale image widget. By focusing on data management and color allocation problems, we were able reach a level of performance which compares favorably with other image processing and visualization packages. A transparent overlay mechanism provides a link to manual and semi-interactive segmentation techniques. We will show how this widget helped us build VISU, a flexible VISUalization software described below. VISU is made up of several standalone applications which communicate which each other over the network. The power and flexibility of these *hypertools* in image and volume visualization will be described with some examples. A biomedical application will be presented.

2 The *pict* gray-scale image widget

As it can be guessed from the name, the *pict* gray-scale image widget is based on the *photo* image widget. After few experiments, it became clear that the *photo* widget was not appropriate for our application. First, this widget can only handle 8- or 24-bit color images, and it does not provide any mechanism to visualize floating-point images. Next, the colors are allocated statically and cannot be changed dynamically by the user. Thus, we decided to implement a new widget by focusing on data management and color allocation.

2.1 Data management

Since we wanted to support byte, short, integer or float-point types, we modified the data structure, in order to remove the fields related to color management and dithering. The raw data is allocated in a block of memory and can be accessed by using the data type information. The master structure also provides a pointer to a block of byte data, corresponding to the normalized raw data. The contrast can be increased or decreased by setting the dynamic range of the raw data. This feature proves most helpful when comparing two floating-point images whose dynamic range is different. The pixel values can be queried, and the result string contains the actual value, for example a floating-point value.

The Khoros1 and VIEW (local LLNL format) file formats are now supported. The GIF and PPM readers were modified in order to read only one color band. Readers/writers for the SUN Raster file format are provided for both the photo and the pict widget. Raw binary files can also be read from a file (or a channel if the code is linked against Tcl7.5) by specifying the dimensions, the data type and the number of bytes corresponding to the header. In all cases, only one slice is read at a time, which decreases the memory requirements dramatically. However, volume rotations and transpositions need to be done off-line. Support will be added in the near-future for HDF, netCDF and ACR-NEMA file formats.

Profiling the Photo source code showed that memory management was fairly inefficient, as a lot of time is spent copying blocks between different addresses. The TkPictPutBlock and TkPictPutZoomedBlock routines were rewritten to make sure memory blocks are duplicated only when necessary.

2.2 Color allocation

The human visual system cannot see more than 60 shades of gray, which makes color management for gray-scale images much more simple than for color ones. Since we wanted to change the colors dynamically and use predefined colormaps, we chose to display the images using an 8-bit PseudoColor Display. This requirement is in our opinion fairly minimal. Besides, our experience proved that dithered images cannot be compared accurately.

In order to allow for fast array transformations, the colors are allocated from a contiguous set. For example, the palette can be inverted quickly by reversing the color indices. The images are displayed in false colors by choosing from a variety of prefined look-up tables. The histogram can be stretched or thresholded to produce a binary image. Furthermore, the colors can be allocated from shared, default or private colormaps. Changing the colors of one shared colormap will affect all the images that share it. This feature allows the user to visualize the same image displayed with different colors, or to compare the result of two different thresholds.

One additional benefit is that the color allocation can be used to display "semi-transparent" overlays. This is a feature that was found very useful in our interactive segmentation work. The user can for example "paint" on the image, draw polygons, Bezier or free-form curves, and yet guess the gray-level values. The overlays can be saved as a mask image. Let us point out that this feature enables the user to extract non-rectangular regions of interest. Alternatively, a mask image can be overlaid on top on the active image, in order to check an off-line segmentation over, remove spurious pixels or compare two data sets. The user can interactively combine overlays with logical operations (or, xor, and, etc..) by changing the Graphic Context. The *pict* widget provides an interface to advanced image processing routines [1, 2], and the overlays can also be used in semi-interactive segmentation [3], where a coarse initialization is specified; the segmented result can then be displayed. Examples in Figure 1(a)-(d) show two cross-sections of a CT volume displayed with different look-up tables. The concept of transparent overlays is described in Figure 2(a)-(c). The use of overlays in semi-automatic segmentation is presented in Figure 2(d)-(f).

3 VISU: a volume VISUalization application

In this section, we describe how the *pict* widget is incorporated in VISU, a user-friendly volume visualization extension of *wish*. In addition to the low-level widget commands, we provide a set of default scripts and high-level commands which make the life of the average user easier. In order to remove any learning curve for LLNL users, the syntax of these commands reproduces that of VIEW, a general signal processing package used in our group. For example, the following commands will read the first slice of volume "hand", display the image, and then display the tenth slice.

```
% rdfile hand_0.sdt f
% disp f
% rdslice f 10
```

The simplicity of these commands enables the user to write his/her own set of macros. In addition to the command line, VISU also provides a graphical interface to most of the high-level commands. In order to reduce the GUI design, we chose to provide only one Control Panel (see Figure 3) and a one-to-one mapping between images and windows. At a given time, one image is considered "active". An image becomes active when the user clicks on it, and its window title is changed. The mouse location and pixel values displayed in the Control Panel correspond to those of the active window. Similarly, moving the slice scale will result in another slice being displayed; the dynamic range of the images can be typed in two entry boxes.

The Palette Menu (Figure 4) lets the user change the colors with the mouse. Four scales can be moved in order to choose the low and high thresholds. The pixels whose values are between the low and high threshold appear white, and the rest appear black. As soon as the scale is changed, the look-up table is updated. Visualizing the actual values of the thresholds instead of normalized values proved very helpful. For example, in our CT applications, the user can see where the attenuation value exceeds a threshold. Predefined look-up tables can be loaded by clicking on one of the radiobuttons. A color tool makes it possible to stretch the colormap in a non-linear way. The graphical interface allows the user to change the intensity and each of the RGB channels independently to create their own look-up table.

In the Overlays Menu (Figure 5), the active mask can be chosen and overlaid onto the active image. The user can choose how to combine the overlays by setting the overlay Graphic Context with the mouse. For example, the intersection of two binary masks can be seen and saved into a new image. We also provide a graphical interface to our segmentation routines.

The GUI can be customized within minutes without recompiling any code. For example, displaying several images side-by-side could be done by packing them in the same canvas, instead of different windows. This flexibility enables us to design the best GUI and to take into account the requirement of a specific imaging application. In most cases, the user will be able to configure the GUI himself.

Releasing the source code on the Internet[1] helped test VISU on various platforms and operating systems we did not have access to. The *configure* tool generates Makefiles automatically. While it is still dependent on Xlib, VISU can be ported to Windows and Mac-OS without too much effort, to become a machine-independent visualization software. About 500 anonymous ftps were logged on our server. Let us remark that this figure may appear small, but most of the VISU users have very specific needs and would probably not use Tcl/Tk at all if this gray-scale image widget did not exist.

4 Hypertools

So far, the features of VISU we described are fairly standard, and can be found in other less flexible visualization packages. However, the Tk library makes it possible to use our widget to build visualization *hypertools*, defined in John Ousterhout's book [4] as "stand-alone applications which can communicate with each other and be reused in ways not foreseen by their original designers". Indeed, it came as a surprise to us how easily the `send` command can be used by visualization tools based on the *pict* widget in peer-to-peer or master/slave relationships. In order to demonstrate the power of these hypertools, we take several examples.

4.1 Image visualization

We mentioned several times in this paper the importance of normalizing floating-point images in the same way. Typically, this can be done by querying the minimum and maximum value of each image, and by computing the absolute minimum and maximum of all the images being displayed. Setting the dynamic range has to be done each time an image is updated; for example, in volume visualization, the dynamic range is likely to be different for each slice. Moreover, visualization tools may run on different systems and display the images on the same screen. In summary, choosing a correct dynamic range is a tedious time-consuming task. However, the `send` command makes things simple, as the following script demonstrates:

```
proc set_range {} {
   # initializations
   set l [winfo interps]
   set min {}; set max {}; set visu_list {}

   # list visu applications
   foreach k $l {
      if { [string match "visu*" \
            [lindex $k 0] ] } {
         lappend visu_list $k
      }
   }

   # query dynamic range of active images
   foreach k $visu_list {
      lappend min \
         [send $k {$curr_img getmin}]
      lappend max \
         [send $k {$curr_img getmax}]
   }

   # compute the global minimum and maximum
   set fmin [lindex $min 0]
   set fmax [lindex $max 0]
   foreach k $min {
      if { $k<$fmin } {
         set fmin $k
      }
      if { $k>$fmax } {
         set fmax $k
      }
   }

   # set the new dynamic range
   foreach k $visu_list {
      send $k {$curr_img range} $fmin $fmax
   }
}
```

In this script, the dynamic range is broadcast to all the visu applications. Other possibilities include showing the pixel values at the same mouse location in different images, so as to compare images on a pixel-by-pixel basis. Using the visu scripts, this option could be written as the following command.

[1] ftp://redhook.llnl.gov/pub/visu

```
foreach k $visu_list {
    send $k {set x $x}
    send $k {set y $y}
    send $k {set pix [$curr_img get $x $y]}
}
```

Instead of sending values, it is possible to send commands, and for example to visualize the same slice in different data sets by broadcasting the command `rdslice $curr_img $curr_slice`. A direct application is the creation of simultaneous animations of volumes located in different file systems and accessed over the network.

4.2 Image profiles and histograms

Visualization of image profiles along an arbitrary line is an invaluable tool to evaluate the presence of artifacts in reconstructed images. For example, in X-ray CT, beam-hardening produces "cupping" artifacts that can easily be detected by extracting a profile. Similarly, the accuracy of the reconstruction techniques can be assessed by visualizing the profiles of sharp edges. Typically, low-pass filtering and noise elimination smooth and spread the transitions.

Along the same lines, visualizing a histogram helps understand the statistics of an image and the distribution of its gray levels. In the case where the histogram is made up of several modes, the objects can typicaly be segmented out by applying different thresholdings to the data.

Although most image visualization packages provide 1D signal viewers, our experience proved that they cannot be easily customized by the user and are not flexible enough.

In our applications, the 1D signals are extracted by issuing a command to the *pict* image widget and sent to a BLT graph. Tcl/Tk handles only character strings and relies on the X protocol to communicate between applications. In addition, the profiles are generally made of less than 1000 samples, so that the overhead introduced by the communications is minimal. This feature allows us to let the user configure the graphical interface. Figure 6 shows an example where a profile is extracted interactively and then sent to a 1D signal viewer. The main benefit of this approach is that the user can dynamically create and extract new profiles or histograms of images viewed across the network, combine and compare them without having to save these one-dimensional signals in files, and transfer them by ftp.

4.3 Data-flow environments

In our biomedical application, all the data preprocessing and segmentation is implemented in Khoros, a data-flow image processing environment. Khoros provides a network editor; the input data undergo a series of transformations each time a network node is executed by the scheduler (Figure 7). This data flow environment helps the user design an imaging application, as the output of each node can be visualized by connecting it to a visualization module (other packages such as AVS, Explorer, LabView use the same paradigm). However, dynamically setting the range of all the images in a data flow environment is next to impossible. Similarly, writing a macro or a loop in a command-line interface sometimes makes more sense than editing a network of nodes.

These two problems were solved by implementing a Khoros module that executes a VISU script. This approach results in several important benefits. First, it compensates for some of the drawbacks of VISU, e.g. by providing simple ways to extract arbitrary slices. Basically, we rely on modules provided in the Khoros distribution, or write our own modules to filter and transform the raw data. Next, the power of both command-line and data-flow environment can be combined: a command-line interface is provided by a remote controller derived from the `rmt` example of the Tk distribution, which can communicate either with a specific application or send messages to all of them. As a result, the user can rely on the power of visual programming, while being able to rely on a more traditional command-line interface. To our knowledge, this feature is not provided by any other visualization package. Finally, the embedded visualization modules can communicate with each other. For example, the same profile could be extracted in different images and displayed in the same 1D signal viewer. Let us remark that the same approach could be easily implemented in other data-flow environments such as AVS or Explorer.

4.4 Volume visualization

As described before, VISU can display slices of a 3D volume. Another way to visualize a volume is to extract isosurfaces, either by using the 3D Marching Cubes algorithm [5] or by reconstruction of a 3D surface from 2D contours [6]. Visualizing a surface helps understand the spatial distribution of objects in 3-space, in contrast to visualizing 2D slices. Almost every visualization package provides these tools, but they are almost never combined or used simultaneously.

Several other approaches make use of Tcl/Tk in volume visualization. Schroeder, Lorensen and Martin recently released vtk[2], a visualization toolkit which can be used either by writing C++ programs, or through Tcl/Tk scripts. Similarly, Lacroute [7] released VolPack [3], a volume rendering library coupled with a Tcl/Tk-based graphical interface.

These two toolkits could be used in conjunction with VISU in order to build a volume visualization package by relying on the `send` command. This approach results in two main benefits.

First, the different visualization modules can be combined easily, even if they are not executed on the same machine. Typically, it will be possible to highlight a point of the surface and to see the intersecting slice. Conversely, viewing a new slice will automatically change the coordinates of this highlighted point. The color tools provided with VISU will also be used in order to choose isosurface values, or the transfer functions in volume rendering. The same graphical interface can also be used to set the viewing parameters in surface and volume rendering. It is our belief that the combination of these visualization techniques will help identify more efficiently the relevant features of volumetric data sets.

Next, provided that the interface between hypertools does not change, the user does not need to know how the surface or volume rendering techniques are implemented. As a result, the best visualization modules can be chosen by the user. For example, vtk supports an abstract rendering engine and can be slower than the tkSM widget [4], which provides easy access to the OpenGL and Mesa libraries. In the case where the software modules were not linked against Tcl/Tk, they can still be used and executed from the Tcl/Tk environment.

5 Conclusion

Although Tcl/Tk is widely distributed in engineering and graphical applications, its use in imaging and visualization is recent. In this paper, we described a new gray-scale widget and its use in visualization of large data sets. Although it relies on a interpreter, its speed compares favorably with typical X-Motif self-contained applications, as we implemented an efficient management of data and colors. Designing high level commands and combining hypertools helps perform tedious tasks automatically, thus increasing the interaction efficiency. We plan in the near-future to enhance our image processing library and to port the code to Windows and Mac. The Tcl7.5 sockets will also be used to develop distributed visualization applications for the Internet.

[2] http://www.cs.rpi.edu:80/~martink

[3] http://www-graphics.stanford.edu/software/volpack/

[4] http://www.isr.umd.edu/ ihsu/tksm.html

6 Acknowledgments

This research was performed under the auspices of the U.S. Department of Energy, contract No. W-7405-ENG-48 and LDRD grant No. 96-ERI-003, with the support of H. Martz and K. Hollerbach. J. Pearlman, M. Abramowitz, E. Nicolas, J.P. Hebert, M. Cody and D. Garrett helped improve VISU by providing invaluable bug reports and suggestions.

References

[1] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *I.E.E.E. Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, June 1991.

[2] L. Vincent. Morphological grayscale reconstruction in image analysis: Applications and efficient algorithms. *I.E.E.E. Transactions on Image Processing*, 2(2):176–201, April 1993.

[3] P-L. Bossart. Détection de contours réguliers dans des images bruitées et texturées : association des contours actifs et d'une approche multiéchelle. Thèse, Institut National Polytechnique de Grenoble, Octobre 1994.

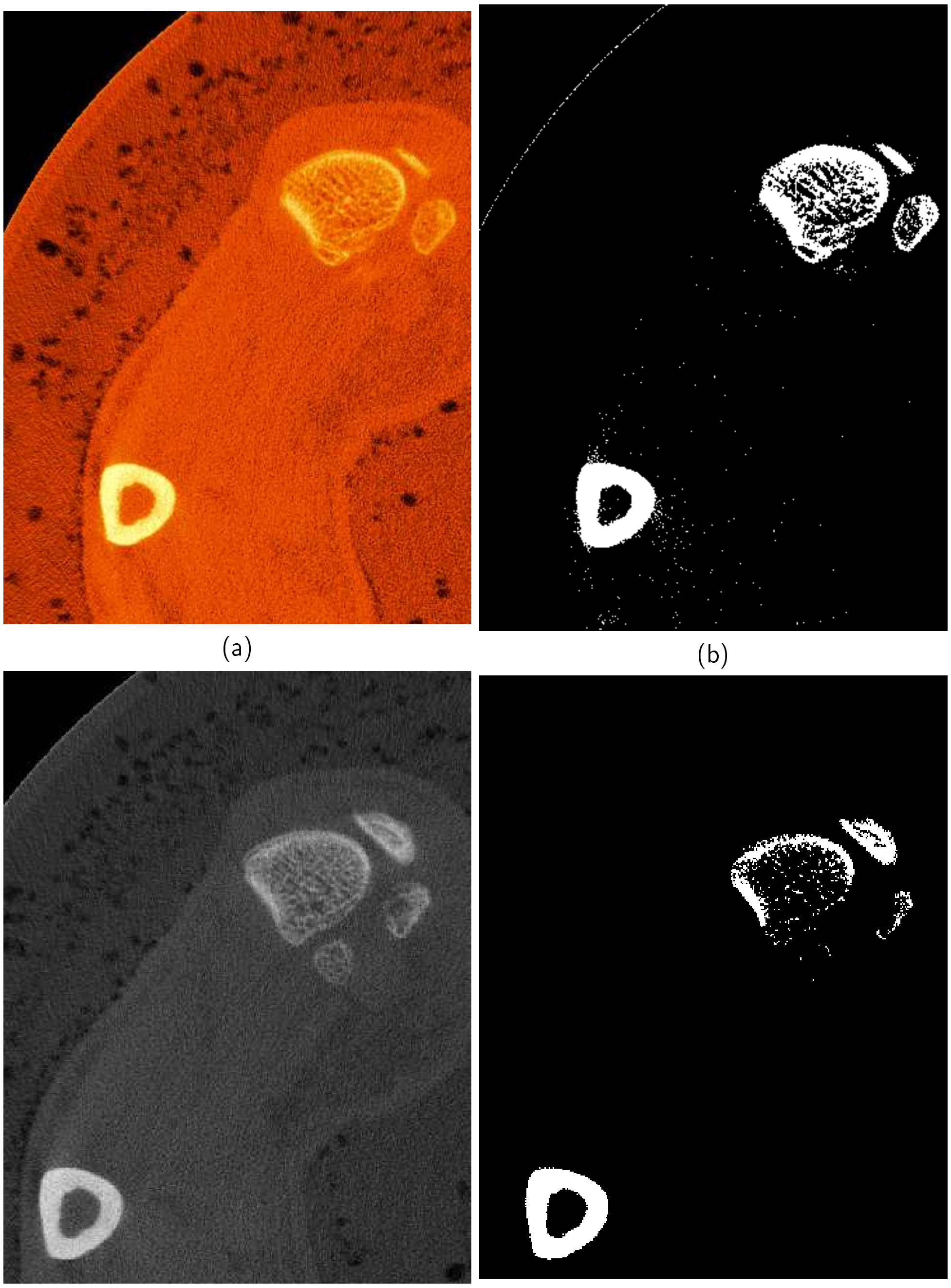
[4] J.K. Ousterhout. *Tcl and the Tk toolkit.* Addison-Wesley, 1994. ISBN 0-201-63337-X.

[5] W.E. Lorensen and H.E. Cline. Marching Cubes: a high resolution surface extraction algorithm. *Computer Graphics*, 21(3):163–169, 1987.

[6] B. Geiger. *Three-Dimensional Modeling of Human Organs and its Application to Diagnosis and Surgical Planning.* PhD thesis, Ecole des Mines de Paris, 1993.

[7] P.G. Lacroute. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation.* PhD thesis, Stanford University, 1995. also Technical Report: CSL-TR-95-678.

7 Figures



(a) (b) (c) (d)

Figure 1: Cross-sections of a thumb and index near joint: (a) corresponds to slice 308 displayed with a 'ct' private colormap (b) is also slice 308 with a shared colormap (c) corresponds to slice 314 with a 'gray' private colormap (d) is slice 314 with a shared colormap. In this example, the private colormaps are used as a reference while the colors are changed simultaneously in the two shared colormaps.

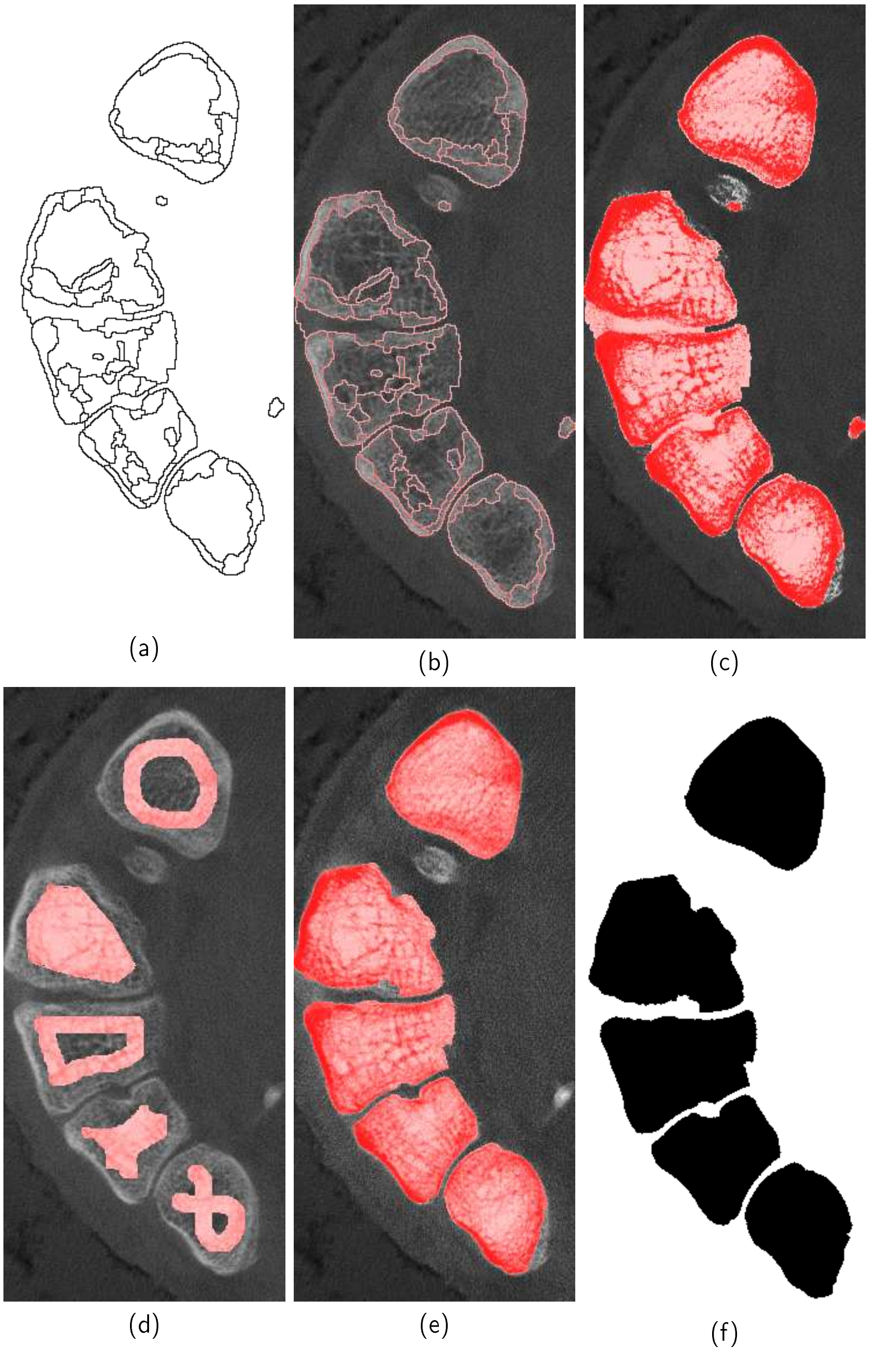


Figure 2: Cross-section of a hand: (a) contours from an off-line segmentation. (b contours overlaid on the image. (c) Regions created from these contours. (d) Coarse manual initialization of overlays. (e) Interactive segmentation result after region-growing. (h) Resulting mask.

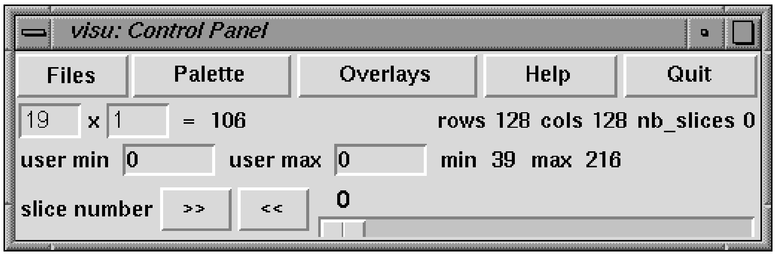


Figure 3: Control Panel

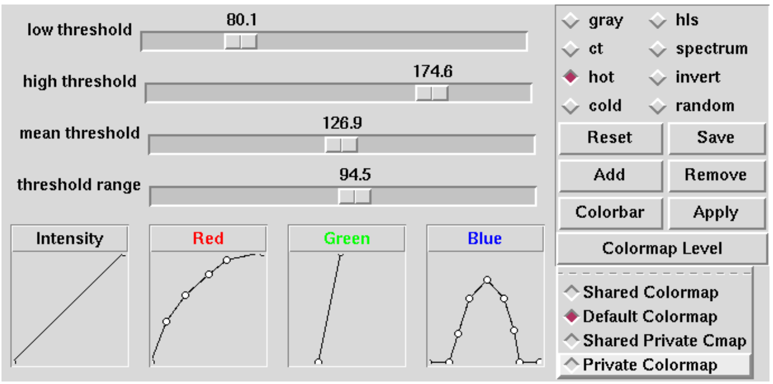


Figure 4: Palette options

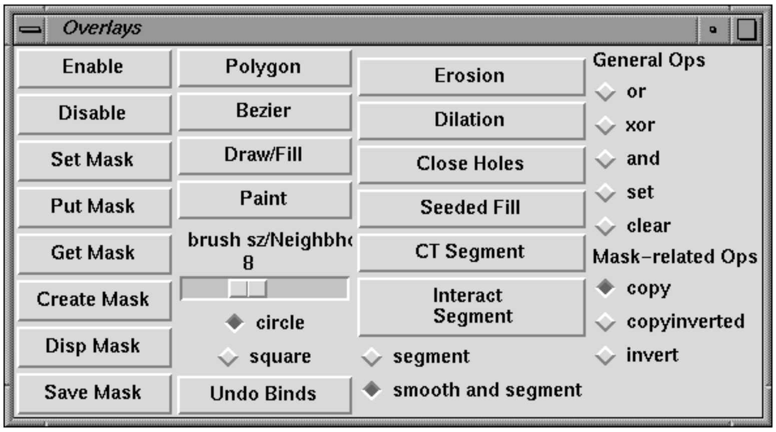


Figure 5: Overlays options

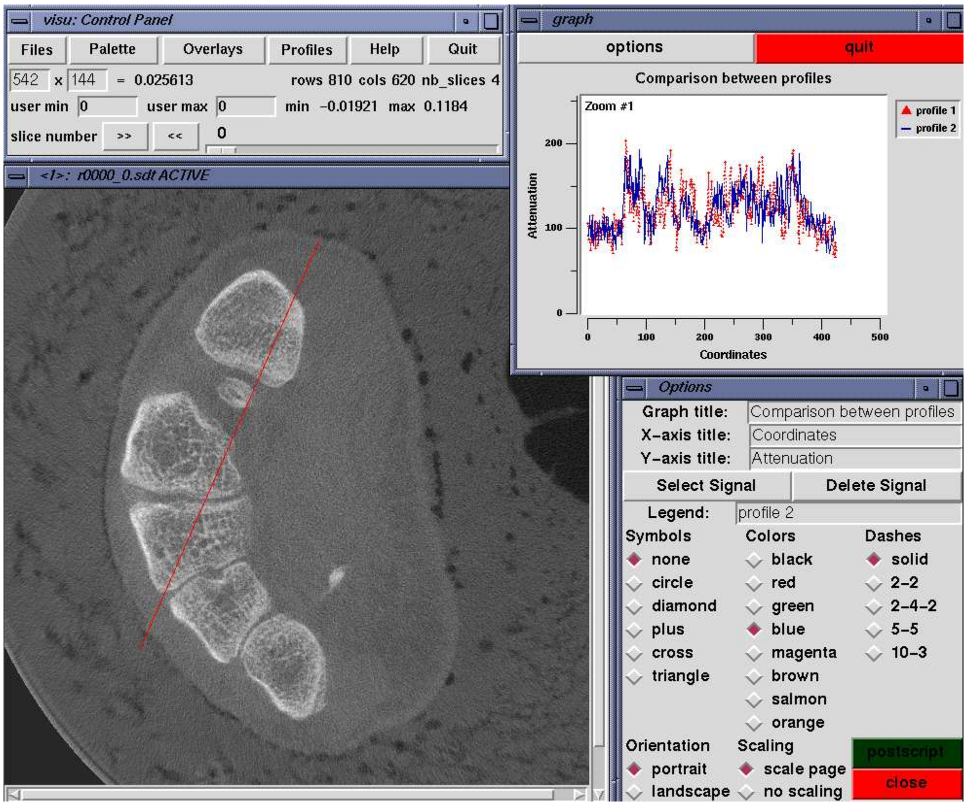


Figure 6: The profiles extracted in VISU are sent across the network to a graph viewer

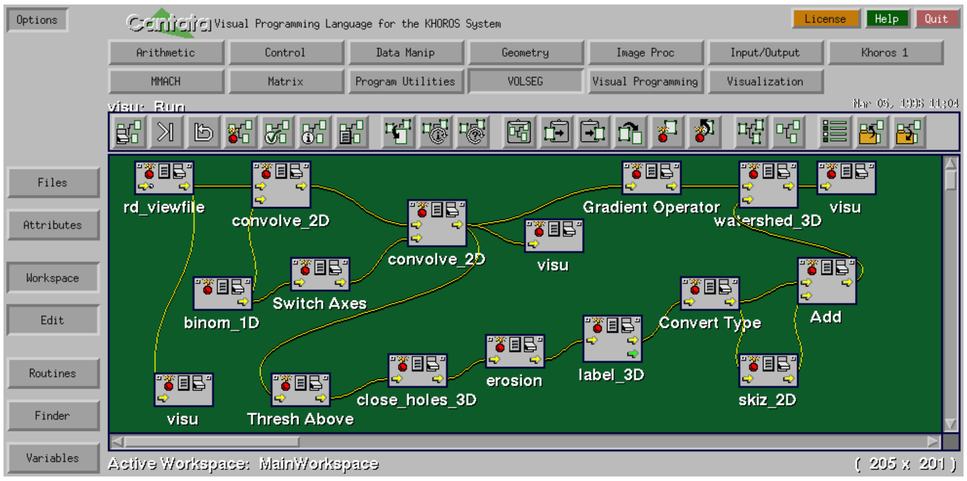


Figure 7: Khoros data flow environment