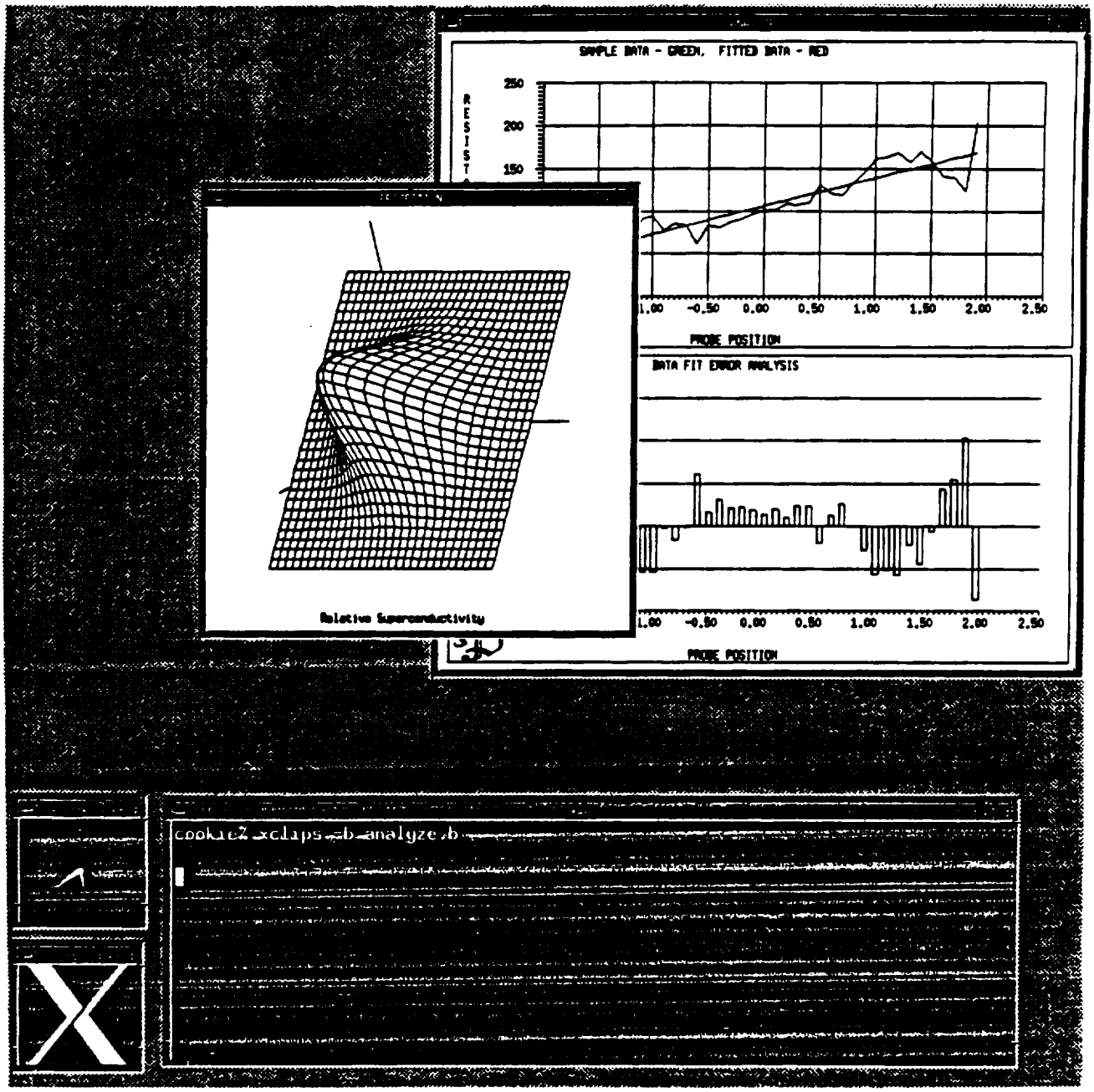
Constructing Complex Graphics Applications With CLIPS and the X Window System

Ben M. Faul
TRW Defense Systems Group
Carson, CA 90746



1. ABSTRACT 1

2. INTRODUCTION 1

3. APPROACH 2

4. DATA ACQUISITION AND MANIPULATION 2
4.1 Inter-process Communications Data 2
4.2 Language Interface For Inter-process Communications 4
4.3 Mathematical Data 5
4.4 Language Interface For Data Analysis 5

5. GRAPHICS 6
5.1 Two Dimensional Graphics 6
5.2 Three Dimensional Graphics 6
5.3 Language Interface For Graphics 6
5.4 Definition of XCLIPS Windows 7
5.5 Printing 11
5.6 XCLIPS Interface to the UNIX X Window System 11

6. XCLIPS APPLIED TO A REAL PROBLEM 12
6.1 Problem Definition 12
6.2 System Architecture 13
6.3 Demonstration 14

6.4 XCLIPS Programs 18
6.4.1 ANALYZE Source Code 18
6.4.2 PROJECTION Source Code 20
6.4.3 XCLIPS Extensions Glossary 21

7. CONCLUSIONS 22

8. FURTHER READING 23

1. ABSTRACT

This article will demonstrate how the artificial intelligence concepts in CLIPS used to solve problems encountered in the design and implementation of graphics applications within the UNIX-X Window System environment. The design of an extended version of CLIPS, called XCLIPS, is presented to show how the X Window System graphics can be incorporated without losing DOS compatibility.

Using XCLIPS, a sample scientific application is built that applies solving capabilities of both two and three dimensional graphics presentations in conjunction with the standard CLIPS features.

2. INTRODUCTION

The CLIPS language provides most of the control functions required for building expert systems. Two areas of the language identified that could use improvement are in the areas of advanced graphics presentation and data analysis functions.

To apply CLIPS to the solution of very complex scientific or business applications, the language requires extensions to handle extended data analysis and graphics presentations problems normally encountered in these systems.

In designing extensions to the CLIPS system to handle these kinds of problems, a survey of several scientific and presentation graphics systems was done to determine the new features.

The survey of these other systems yielded the following capabilities that would be most desirable in the extended CLIPS expert system shell:

Inter-process communications - Many problems are better solved by the ability to use a server/client architecture.

2D & 3D Charting/Graphics - A picture is worth a thousand words.

Printing of Charts/Graphics - Hard-copy is needed, in order to publish the charts and graphs.

Data Smoothing - Reduces noise in a set of experimental data.

Curve Fitting - Polynomial and cubic splines curve fitting to a set of values.

Simultaneous Equations - Solves systems of linear equations.

The remainder of this document describes the philosophy of how CLIPS was extended to incorporate these new features and how well the resultant XCLIPS performs in solving a non-

trivial problem.

3. APPROACH

The design approach for extending the CLIPS language involves two distinct tasks.

The second first involved designing a graphics system for XCLIPS to use. While the X Window System was chosen as the graphics sub-system, linking XCLIPS directly to X would obviate the expert systems from ever being used on DOS; the X Window System is not available on DOS, nor is it ever likely to be available on DOS.

The second task involved linking the XCLIPS language to the data analysis algorithms and graphics sub-system. For the most part, interfacing XCLIPS to these sub-systems follows the method defined in section 2, in the CLIPS 4.3 Advanced Programming Guide. However, the interface to some of the data analysis functions requires the use of vectors and matrices as parameters. Because the data types of standard CLIPS are not convenient for representing matrices, the language had to be extended in a non-standard manner.

4. DATA ACQUISITION AND MANIPULATION

Advanced data handling capabilities required by XCLIPS fall into two categories: inter-process communications and the mathematicics based tasks such as curve-fitting and the like.

4.1 Inter-process Communications Data

Many applications are better implemented as separate cooperating entities - using a server/client architecture.

A familiar server/client architecture may be found in large database management systems. Typically, the only program that actually updates the database is the "server" process. The user "client" programs communicate their requests for processing to the server, that handles the requests and returns the appropriate responses. In such a way, access to the database is maintained through a single process.

In the XCLIPS system, the inter-process mechanism used is the TCP protocol. Using TCP, an XCLIPS program may communicate directly to any process within the same machine, or any process on any machine that the user can access via the local or wide-area network.

In order to open a communications path between programs, the caller and receiving programs first have to be ready to make connections. The two programs that will be communicating agree beforehand which communications channel (or "socket") will handle the call. The program that will be called prepares to receive by making a function call to place the socket into the "accepting state". The function that places the program into the accepting state returns immediately with status indicating whether any other program is ready to communicate. In this

way, the program can continue processing, without the need for waiting for a connection to complete.

Periodically the accepting program checks the status of the socket to find out if a connection has been accepted.

The program that wishes to place a communications call to another program specifies the address of the program to be called. This address consists of the Internet name and socket number. This action puts the calling program in the "opening state".

When the opening program makes the function call to open the communications socket, the function waits for the call to complete before returning; however, if the call does not complete within 5 seconds the call returns with an error.

When the open completes, the opening program gets a return code indicating success. Also, the called program, which is periodically checking the socket for a completed call likewise gets a return code indicating that the communications channel is now open for communications.

Once established, bi-directional communications is as easy as reading and writing to a file.

To promote efficiency, when a socket is read by a rule in XCLIPS, the socket read returns immediately, whether or not data is actually available. The socket read call returns the number of bytes it read. When the return code is greater than zero, data is ready to be processed.

To demonstrate how easy implementing inter-process communications within XCLIPS programs can be, consider the following rules for sending and receiving messages across a network.

In the rules defined below, the process listens on socket 3000; when successful, the socket descriptor 1 is used for reading a message from the network and then printing it on the terminal.

```
(defrule listen "Listen for network open"
        (not (socket opened))
        =>
        (if (> 0 (NetAccept 3000 1)
        then
                (assert (socket open)))

(defrule read-socket "If data in socket, then print"
        (socket open)
        =>
        (bind ?string (NetRead 1))
        (if (neq ?string "")
        =>
                (printout t ?string t)))
```

In the following rules the process opens a connection to a process on machine "shasta" at socket

3000 (the previously described rules). Once the connection is open the "write-socket" rule reads from the terminal and sends the message to the other process on machine shasta.

The "read-socket" rule of the other process reads the data sent by the "write-socket" rule and then prints this data on the terminal.

```
(defrule setup "Setup the call"
        (not (call setup))
        =>
        (GetHostByName "shasta")
        (assert (call setup)))

(defrule check-socket "Check socket for open success"
        (not (socket open))
        (call setup)
        =>
        (if (> 0 (NetOpen 3000 1))
        then
                (assert (socket open)))

(defrule write-socket "Write to socket"
        (socket open)
        =>
        (NetWrite (read) 1)))
```

These two programs may be on the same machine, on different machines on the same local-area network, or on different machines separated across the world on a wide-area network.

4.2 Language Interface For Inter-process Communications

In the UNIX environment, the programmatic interface to the TCP layer is done through file descriptors. However, in a DOS system, TCP sockets are separate from the file descriptors. Because this bifurcation of file/socket descriptors is a given on DOS, in the spirit of keeping DOS and UNIX versions of XCLIPS equivalent, this bifurcation of file/socket descriptors is retained in the UNIX version. Note that while file I/O on both DOS and UNIX is of the blocking variety, Network I/O on XCLIPS is of the non-blocking type.

As can be seen in the XCLIPS programs of the previous section there are several new language constructs introduced. Actually, this network capability is accomplished by the introduction of only five new commands to the language.

(GetHostByName) - Identifies the program to be called, by its Internet address.

(NetAccept) - Place a specified socket in the "accept" state. (Listen for a call.)

(NetOpen) - Place a specifies socket in the "opening" state. (Place a call.)

(NetWrite) - Send data to the other program.

(NetRead) - Receive data from the other program.

The ability of XCLIPS rules to communicate across a network, in a transparent, real-time fashion opens up new vistas for CLIPS applications.

4.3 Mathematical Data

The XCLIPS language includes many functions (over 75) for easily handling and analyzing large volumes of data. Section 2 of this document details the kinds of functions available for data analysis.

4.4 Language Interface For Data Analysis

All of the data analysis functions involve operations on floating point arrays or matrices. While CLIPS has a vector data type, it is not suitable for handling large amounts of data, nor are these vectors shareable across rules.

To accommodate easier handling of single and two dimensional arrays, as well as for the ability to share this kind of data across rules, two new data types are introduced -- Vector (single dimension) and Matrix (two dimensions). These new data types are accessed by name as string variables. The new data types have their own actions for assigning and evaluating elements.

As representative of the class of data analysis functions available in XCLIPS, the curve-fitting functions are briefly discussed below:

In the curve-fitting section of the XCLIPS language there are three functions available.

(PolyCurveFit) is a function that fits a polynomial with linear coefficients to a dependent - independent variable set of data.

(CubicSplines) is a function that fits a set of polynomial equations to a discrete set of data.

(CalcSpline) is a function that will calculate the cubic spline interpolation of a y-given value given an x-value of the cubic splines coefficient matrix calculated by the function CubicSplines.

These curve fitting functions are representative of the power and flexibility of the functions available within XCLIPS. For sample uses of these functions refer to Section 6.4.

5. GRAPHICS

5.1 Two Dimensional Graphics

The XCLIPS language provides both plot and chart graphics, as well as object oriented drawing. Graphical representations are often the best method for conveying information derived from a mathematical analysis; the pictorial representation of a sine wave carries more information to the reader than an equation or columns of numbers.

There are approximately fifty functions available for 2D graphics. The following table details the kinds of features available. The extensions were written in a machine-independent manner, all of these graphics functions are available under both DOS and UNIX versions of XCLIPS.

Automatic Axes and Scaling
Automatic Grid Drawing
Line Plotting
Bar Plotting
Contour Plotting
Pie Charting
Patterning
Text Printing
World <-> Real Coordinate Translations
Color Selection
Object Oriented Drawing

5.2 Three Dimensional Graphics

Building on the 2D graphics capabilities, XCLIPS implements 3D projections using 2D functions. There are thirty 3D graphics functions available in XCLIPS. The following table summarizes the capabilities available in the language:

World <-> Actual Coordinate Translation
Concatenation
3D Rotation
Perspective Selection
3D Scaling
Color Selection
Solid Drawings

5.3 Language Interface For Graphics

To facilitate an XCLIPS product portable to both DOS and UNIX, XCLIPS uses an arbitrarily defined that is neither specific to DOS or UNIX. The XCLIPS language interfaces to this arbitrary window system. In this manner, the language is independent of the native DOS graphics

or the X Window System based graphics. The graphics commands include both low-level (draw line, point, etc.) to very high-level (auto-axes generation, draw contour, draw 3D in 2D projection, draw object and the like) commands.

5.4 Definition of XCLIPS Windows

The window system used internally by XCLIPS is an arbitrary one designed to be portable to both the DOS and UNIX operating systems.

The DOS version of XCLIPS works on Pcs using CGA, EGA, VGA, and Hercules graphics cards. The XCLIPS programs are independent of the graphics card used in the PC. Of course, color application's output is converted to black and white on monochrome displays; nonetheless the XCLIPS application still run. On the DOS screen up to 10 "windows" may be created by the application. Each of these windows is separately accessible by the XCLIPS program. The windows may or may not overlap as the programmer desires. These windows are accessed with a "world" coordinate system, defined by the user program.

In the UNIX-X Window environment, XCLIPS creates an X window that corresponds to the DOS screen. Within this X window, up to 256 sub-windows (instead of 10) may be created by the XCLIPS program. If the user program desires, the resolution of the XCLIPS window may correspond to a resolution found under DOS on CGA, EGA, VGA or Hercules graphics adapters. However, the XCLIPS program may select a base window to be of any size that the X Window System display can support. The UNIX based XCLIPS program uses the same base color scheme as the DOS system uses. However, the XCLIPS program may utilize all the colors available to the X Server, if the developer so desires; but, such programs are not backwards compatible under DOS. Even though the UNIX based extended XCLIPS has higher resolutions, more colors, virtual memory in its favor, the XCLIPS programs will still run under DOS, subject to DOS's memory restrictions.

To demonstrate how well the arbitrary window interface works, consider the following XCLIPS rules that describes a wire-frame house in a 3D perspective as displayed on a DOS screen and a UNIX X Window.

```
(defrule main "Initialize the system"
        (not (system initialized))
        =>
        (dInit3)
        (Init3D 6)
        (SetWorldCoordinates -10 -10)
        (SelectColor 3)
        (WorldScale 1 2)
        (WorldRotate3 10 0 1)
        (Persp 15)
        (assert (system initialized))
        (assert (draw house)))
```

```
(defrule draw-house "Draw the wire-frame house"
        ?rem <- (draw house)
        =>
        (retract ?rem)
        (SelectColor 15)
        (Move3Abs 1)
        (Line3Abs 1 -1)
        (Line3Abs 1 -1 -1)
; right side
        (Line3Abs 1 -1)
        (Line3Abs 1)
        (Move3Abs -1)
        (Line3Abs -1 -1 -1)
; left side
        (Line3Abs -1 -1)
        (Line3Abs -1)
        (Move3Abs 1)
; front top
        (Line3Abs -1)
        (Move3Abs 1 -1)
        (Line3Abs -1 -1)
; front bottom
        (Move3Abs 1 -1)
; back top
        (Line3Abs -1 -1)
        (Move3Abs 1 -1 -1)
; back bottom
        (Line3Abs -1 -1 -1)
        (Move3Abs 1)
        (Line3Abs 0 1.5 1)
; roof
        (Line3Abs -1)
        (Move3Abs 1 -1)
        (Line3Abs 0 1.5 -1)
        (Line3Abs -1 -1)
        (Move3Abs 0 1.5 1)
        (Line3Abs 0 1.5 -1))
```

The following two figures show the results of the XCLIPS programs running under both DOS and UNIX. In Figure 1, the DOS screen is displayed. In Figure 2, the UNIX screen is shown, running the exact same program.

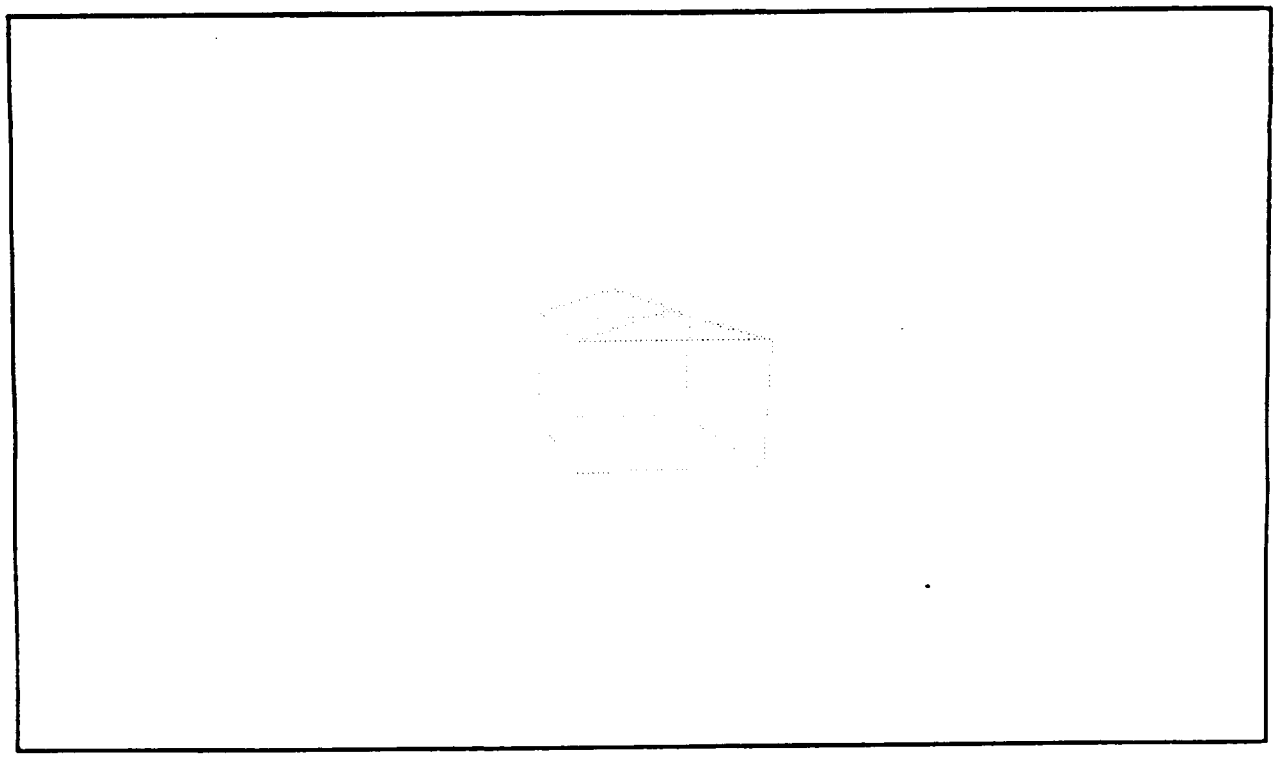


Figure 1, Screen Dump of DOS Version of "3D House".

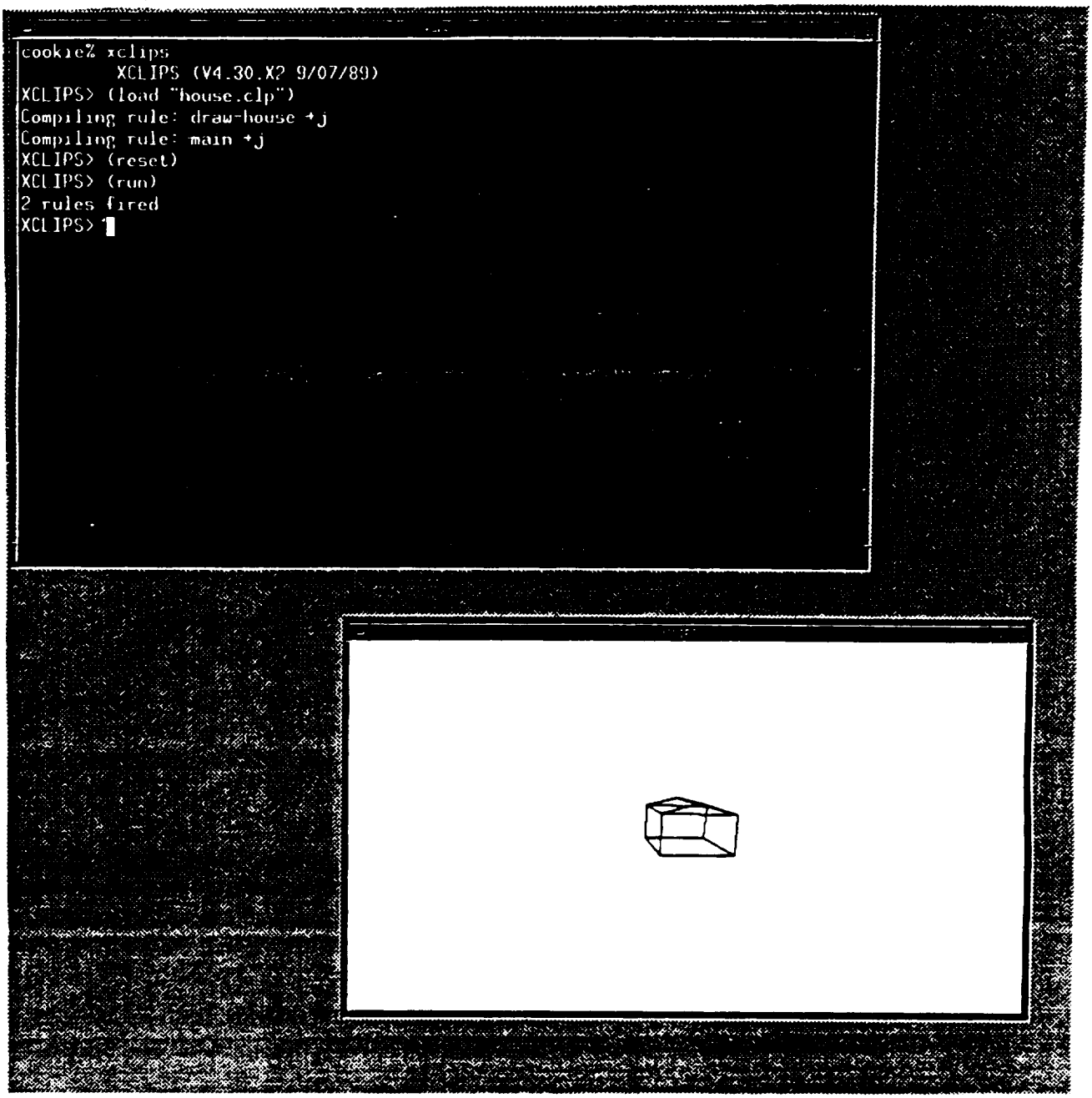


Figure 2, UNIX Version of "3D House".

As Figures 1 and 2 show, by using an arbitrary windowing system the XCLIPS programs are easily made machine and operating system independent.

5.5 Printing

Users of graphics systems need hard-copy output as well as screen outputs. Since a range of printers would be used by any given set of users, XCLIPS supports some of the more popular printers. The language interface to the printer drivers is via a single call, with parameters used to inform XCLIPS which printer is selected, where to spool the output, and landscape/portrait modes.

Table 1. Printers Supported by XCLIPS
EPSON MX, FX, LQ Dot Matrix Dot Matrix Printers.
Toshiba Dot Matrix Printers
HP Laser Jet Printers
Plotters implementing Hewlett-Packard Graphics Language (HPGL).

A future enhancement will include Post-Script support, as this output format is readily becoming the standard for publishing.

5.6 XCLIPS Interface to the UNIX X Window System

The graphics sub-system used by XCLIPS is the X Window System. Because the X Window System is divided into two distinct parts, with all of the device dependent code isolated in the server, XCLIPS is inherently machine-independent.

XCLIPS utilizes the Xlib programming library for all its graphics requirements. Xlib provides all of the primitive graphics capabilities needed by XCLIPS; however, since Xlib calls are very low-level, a separate library called "seglib" was created that supplies high level functions to the XCLIPS language, such as auto-axes, bar and pie charting, etc., that are of more interest to the expert system user.

Seglib is organized using a layered approach, making it usable with the Microsoft "graph.lib" library under MS-DOS and on UNIX under the X Window System. XCLIPS expert systems utilizing graphics capabilities work without modification on either UNIX (using X) or DOS (using standard DOS graphics).

In Figure 3, the operating system independent graphics architecture of XCLIPS is described. Notice that the top layer, the XCLIPS language interface and high-level graphics, is common across both the DOS and UNIX versions of XCLIPS. The middle layer, also common to both DOS and UNIX versions, is an interface to the machine dependent graphics layer (bottom layer). The middle layer is divided into to two parts. The top half of the middle layer is an arbitrary graphics system that communicates to a graphics library with a compatible calling sequence to Microsoft's "GRAPH.LIB" library. The bottom half of the middle layer is an implementation specific module depending on which operating system is being utilized. On DOS, this bottom half is merely a coupling to GRAPH.LIB. On UNIX this bottom half is a module that translates GRAPH.LIB calls to X-Window System Xlib calls.

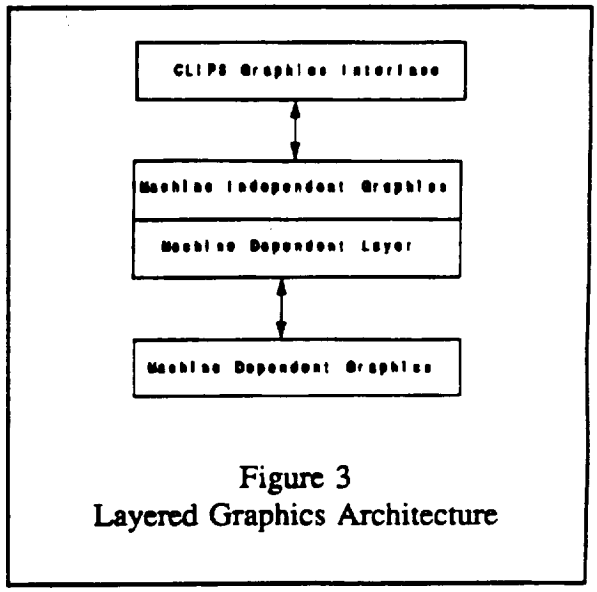


Figure 3
Layered Graphics Architecture

The bottom layer is operating system specific. On DOS, this layer is merely the Microsoft graph.lib library. On UNIX, this bottom layer is the Xlib X Window System library. Using this architectural approach, XCLIPS remains true to its operating system independence heritage.

Note that this DOS to X Window System library interface at level-2 has the potential to allow DOS programs written to the Microsoft library to be easily ported to the UNIX/X Window System interfaces.

6. XCLIPS APPLIED TO A REAL PROBLEM

6.1 Problem Definition

To demonstrate the effectiveness of applying non-procedural languages, such as XCLIPS, to solving graphics related problems, a fairly sophisticated application is described, and then implemented. This XCLIPS system will demonstrate the uses of the 2D and 3D graphics described in Sections 6.1 and 6.2, the inter-process communications mechanisms described in Section 5.1, and the curve-fitting data analysis functions described in Section 6.3.

The demonstration system will be a simulated resistance/superconductivity analysis station. This analysis station will have the following features:

Table 2. Capabilities of the Superconductor Analysis Station
Obtains resistance samples from the surface of the object being analyzed.
In a split window display the resistance samples on the object. This display will show the actual samples along with a prediction of the resistance curve, and a running computation of the prediction confidence.
In a separate window, display a 3D projection of the resistance found across the object's surface. The X and Y axis represent the surface of the object and the Z axis represents the resistance at that point on the object.

6.2 System Architecture

There will be two instances of XCLIPS running on the UNIX computer that communicate via inter-process communications mechanisms.

The first XCLIPS system is called "ANALYZE". This expert system is responsible for communicating with the resistance probe, over a TCP socket. ANALYZE will also handle all computations involving data gathering and analysis (curve fitting) as well as all 2D graphs). Also, ANALYZE will handle any user input.

The second XCLIPS system is called "PROJECTION". This expert system is responsible for generating the 3D projection of the object's resistance. The data for the 3D projection will be a contour map. This contour map is send to PROJECTION by the ANALYZE expert system.

The following figure describes the processing architecture of the complete resistance analysis work-station.

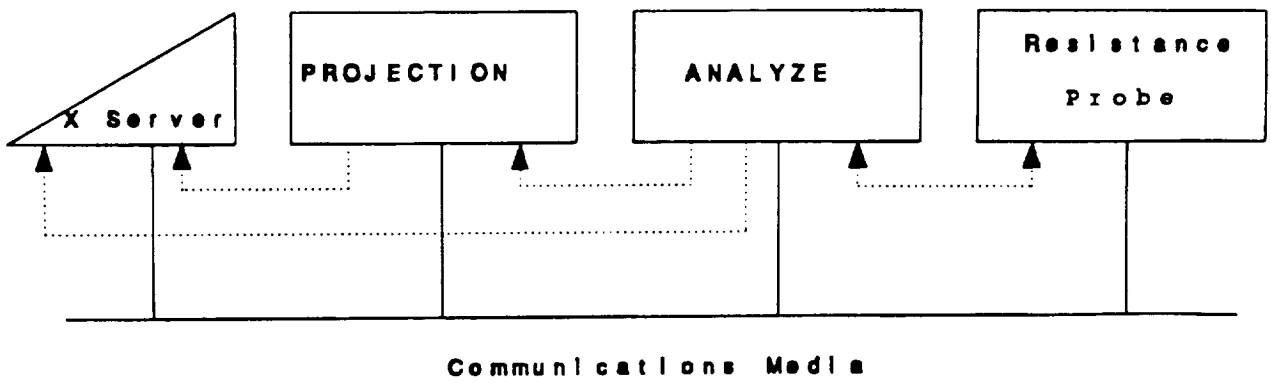


Figure 4, Superconductor Analysis Work Station Architecture

As can be seen in Figure 4, the central server is the process called ANALYZE. ANALYZE controls input and output to the probe, and the PROJECTION system. The graphics output of both ANALYZE and PROJECTION is sent to an X Server.

Due to its inherent server/process architecture, all the processes could be on the same machine, or each process could be on a different machine across a network. This transparent distributed architecture is flexible, without burdening the user with having to know the specifics of how the system operates.

When the analyst starts the session, the ANALYZE expert system is started. ANALYZE then establishes connections with the resistance probe. Once the connections are successfully started, ANALYZE then requests user input as to where to place the probe. Once ANALYZE has the coordinates to analyze it sends the appropriate information to the probe. When ANALYZE receives the data from the probe it displays the information in its 2D windows as line-plots. When the analyst wants to view a 3D projection of the object's resistance the user pushes an icon with the mouse button. ANALYZE then starts the PROJECTION expert system, establishes a TCP connection with it, and passes the data to be displayed as a contour map.

PROJECTION then computes the 3D projection of the contour map and displays the projection.

6.3 Demonstration

The resistance analysis work-station is started by typing "xclips -b analyze.b" at the UNIX shell prompt.

After ANALYZE begins running, the coordinates of the object are sent to the probe. After the

necessary data is retrieved from the probe, the graphs are then presented. In the figure on the next page, the 2D chart of the resistance of the material under test along with a prediction of the material's resistance appears in the upper window. In the lower window the error analysis of the predictions appear as a line-plot.

Notice the "printer" icon in the lower left hand side of the upper graph window, and the Gothic "P" icon in the lower left hand side of the lower graph window. Depressing the mouse button while the mouse rests on the "printer" icon causes the rule to fire that prints the window on a dot matrix printer. Depressing the mouse button while the mouse cursor rests on the Gothic-P icon starts the PROJECTION expert system.

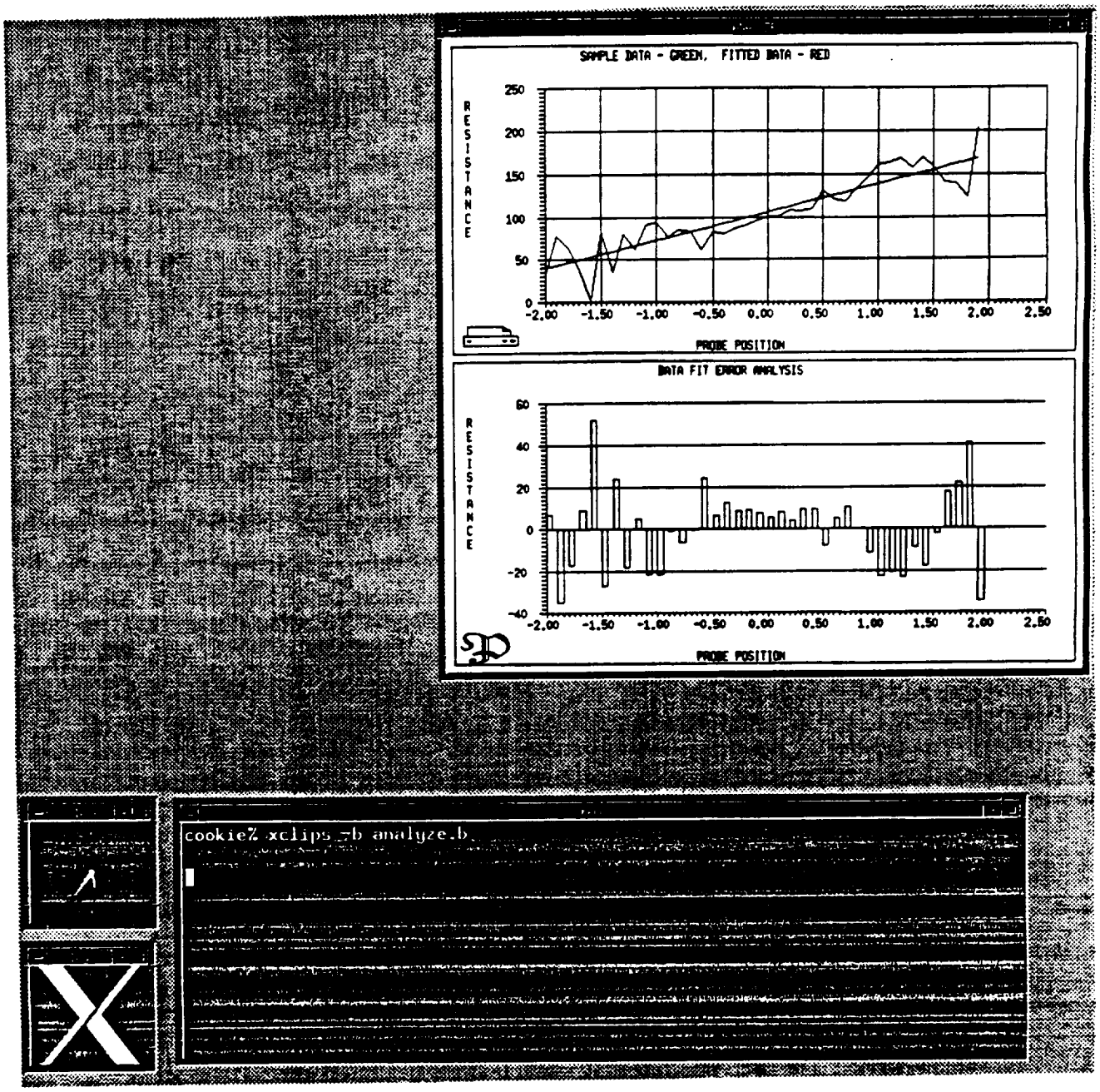


Figure 5, ANALYZE (2D) Display

The following figure shows the result of depressing the Gothic-P icon, that results in the 3D projection of the contour map.

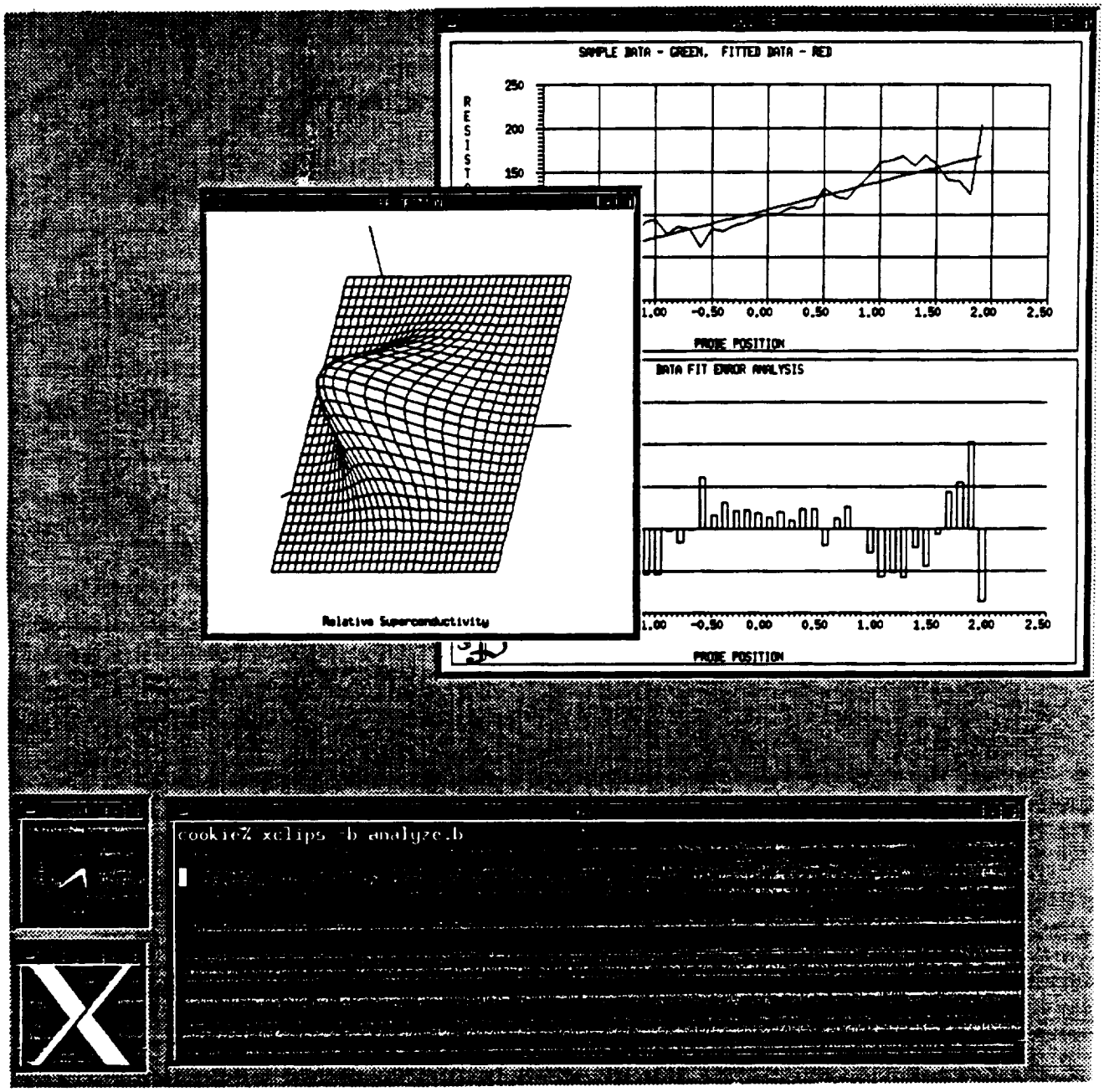


Figure 6, ANALYZE (2D) and PROJECTION (3D) Display

6.4 XCLIPS Programs

In this section the two expert systems source code is listed in sections 6.4.1 and 6.4.2. In Section 6.4.3 a glossary of the XCLIPS extensions used in this article are presented.

6.4.1 ANALYZE Source Code

```
(defrule initialize "Initialize the ANALYZE expert system"
        (not (system initialized))
        =>
; allocate storage
        (Vector "xdata" 50)
        (Vector "ydata" 50)
        (Vector "indvar" 50)
        (Vector "depvar" 50)
        (Vector "coef" 50)
        (Vector "coefsig" 50)
        (Vector "yest" 50)
        (Vector "resid" 50)
        (Vector "numobs" 1)
        (Matrix "contour" 50)
; connect to probe & get data
        (GetHostByName "probe")
        (bind ?test 0)
        (while (= ?test 0)
                (bind ?test (NetOpen 3000 1)))
        (NetWrite 1 "-2 0 2.5 0")
        (bind ?test 0)
        (while (= ?test 0)
                (bind ?test (NetRead 1)))
        (Assign "numobs" 0 ?test)
        (XTitle "ANALYZE")
        (InitSEGraphics 600600)
        (assert (system initialized)))
        (InitSEGraphics 6)

(defrule display-and-fit ""
        (system initialized)
        =>
        (bind ?numobs (Evaluate "numobs" 0))
; fit data to 1st order polynomial
        (NetRead 1 (Address "depvar"))
        (NetRead 1 (Address "indvar"))
        (PolyCurveFit "indvar" "depvar" ?numobs
                "order" "coef" "yest" "resid" "coefsig")
        (SetCurrentWindow 3)
        (BorderCurrentWindow 2)
        (SelectColor 6)
        (SetAxesType 0)
        (AutoAxes "xdata" "ydata" ?numobs 1)
        (LinePlotData "xdata" "ydata" ?numobs 3 0)
```

```
        (SelectColor 3)
        (TitleWindow "SAMPLE DATA - GREEN,  FITTED DATA - RED")
        (TitleXAxis "PROBE POSITION")
        (TitleYAxis "RESISTANCE")
         (bind ?i 0)
         (while (<= ?i ?numobs)
                  (Assign "ydata" ?i (Evaluate "yest" ?i))
         (bind ?i (+ ?i 1)))
; draw the curve
        (LinePlotData "xdata"  "ydata"  ?numobs  4 3);
        (DrawGrid 10)
         (assert (display errors))
         (Register (Transform(PutObject "printer") 0 0) "print"))

(defrule display-error analysis
         (display errors)
         =>
        (SetCurrentWindow 3)
        (BorderCurrentWindow 2)
        (SelectColor 6)]
        (SetAxesType 0)
         (bind ?i 0)
         (while (<= ?i numobs)
                  (Assign "ydata" ?i (Evaluate "resid" ?i))
                (bind ?i (+ ?i 1)))
        (AutoAxes "xdata" "ydata" ?numobs 1)
        (BargraphData "xdata" "ydata" ?numobs 0.05 1)
        (TitleWindow "DATA FIT ERROR ANALYSIS")
        (TitleXAxis "PROBE POSITION")
        (TitleYAxis "RESISTANCE")
        (DrawGridY 10)
        (assert (watch mouse))
         (Register (Transform(PutObject "Gothic-P") 0 0) "project"))

(defrule watch-mouse "Watch the mouse, and do what it says"
         (watch mouse)
         =>
         (while (= 0)
                  (if (= (MouseHit) 1)
                  then
                           (GetMouse) (bind ?object (Analyze (Pick)))
                           (if (eq ?object "print")
                           then
                                    (ScreenDump "/usr/ben/spool" "epson-lq" 1 1 0))
                           (if (eq ?object "project")
                           then
                                    (system "xclips -b project.b &")
                                    (GetHostByName "shasta")
                                    (while (eq (NetOpen 3000 2) 0))
                                    (NetWrite 2 50)
                               (NetWrite 2 15)
                                    (NetWrite 1 "contour_map")
                                    (bind ?test 0)
```

```
                (while (= ?test 0)
                        (NetRead 1 (Address "contour")))
                (NetWrite 2 (Address "contour"))))))
```

6.4.2 PROJECTION Source Code

```
;*************************************************************
;       Draw 3D From Contour Plot
;*************************************************************

(defrule create-function "Create the contour map"
  (create function)
  =>
  (Vector "elements" 2)
  (while (= (NetRead 1 (Address "elements") 0))
  (bind ?num (Evaluate 1 "elements"))
  (bind ?alloc (Evaluate 2 "elements"))
  (Matrix "contour.x" ?alloc ?alloc)
  (Matrix "contour.y" ?alloc ?alloc)
  (Matrix "contour.z" ?alloc ?alloc)
  (NetRead 1 (Address "contour.x"))
  (NetRead 1 (Address "contour.x"))
  (NetRead 1 (Address "contour.y"))
; draw the contour map
  (Vector "pv.x" 5)
  (Vector "pv.y" 5)
  (Vector "pv.z" 5)
  (bind ?lower (- 1 ?num))
  (bind ?i (* -1 (- ?num 1)))
  (while (<= ?i ?num)
        (bind ?j (* -1 (- ?num 1)))
        (while (<= ?j ?num)
                (Assign "pv.x" 0 (Evaluate "contour.x" (+ ?i ?lower) (+ ?j ?lower)))
                (Assign "pv.y" 0 (Evaluate "contour.y" (+ ?i ?lower) (+ ?j ?lower)))
                (Assign "pv.z" 0 (Evaluate "contour.z" (+ ?i ?lower) (+ ?j ?lower)))

                (Assign "pv.x" 1 (Evaluate "contour.x" (+ ?i ?num) (+ ?j ?lower)))
                (Assign "pv.y" 1 (Evaluate "contour.y" (+ ?i ?num) (+ ?j ?lower)))
                (Assign "pv.z" 1 (Evaluate "contour.z" (+ ?i ?num) (+ ?j ?lower)))

                (Assign "pv.x" 2 (Evaluate "contour.x" (+ ?i ?num) (+ ?j ?num)))
                (Assign "pv.y" 2 (Evaluate "contour.y" (+ ?i ?num) (+ ?j ?num)))
                (Assign "pv.z" 2 (Evaluate "contour.z" (+ ?i ?num) (+ ?j ?num)))

                (Assign "pv.x" 3 (Evaluate "contour.x" (+ ?i ?lower) (+ ?j ?num)))
                (Assign "pv.y" 3 (Evaluate "contour.y" (+ ?i ?lower) (+ ?j ?num)))
                (Assign "pv.z" 3 (Evaluate "contour.z" (+ ?i ?lower) (+ ?j ?num)))

                (Assign "pv.x" 4 (Evaluate "contour.x" (+ ?i ?lower) (+ ?j ?lower)))
                (Assign "pv.y" 4 (Evaluate "contour.y" (+ ?i ?lower) (+ ?j ?lower)))
                (Assign "pv.z" 4 (Evaluate "contour.z" (+ ?i ?lower) (+ ?j ?lower)))
```

```
                (PolyFill3D "pv.x" "pv.y" "pv.z" 9 4 5)
                (bind ?j (+ ?j 1)))
        (bind ?i (+ ?i 1)))
  (assert (watch mouse)))

(defrule watch-mouse "Watch the mouse for click, exit if found"
  (watch mouse)
  =>
  (while (eq 0 (GetMouse))
        (Service))
  (CloseSEGraphics)
  (exit))

(defrule init "Initialize Comm Port, Draw the Axes"
  (not (system initialized))
  =>
  (while (= (NetAccept 3000 1) 0))
  (XTitle "PROJECTION")
  (Tinit3)
  (Init3D 6)
  (SetWorldCoordinates -10.0 -10.0 10.0 10.0)
  (WorldRotate3 20.0)
  (WorldRotate3 -45.0 1)
  (SelectColor 15)
  (Draw3DAxis 10)
  (assert (system initialized))
  (assert (create function)))
```

6.4.3 XCLIPS Extensions Glossary

The following list summarizes the XCLIPS extensions used in this presentation.

Address - Returns the address of the object specified.
Assign - Assign a value to matrix or vector at index specified.
AutoAxes - Draw axes from data.
BarGraphData - draw a bargraph from specified data.
BorderCurrentWindow - Border the current window.
CloseSEGraphics - Close the graphics window.
Draw3DAxis - Draw the 3D axis from specified values.
DrawGrid - Draw a grid in the window.
Evaluate - Return a value from a vector or matrix at specified index.
GetHostByName - Given a host name, initialize the network parameters.
GetMouse - Get the mouse position.
Init3D - Initialize 3D graphics routines.
InitSEGraphics - Initialize the 2D graphics routines.
Line3Abs - Draw a line from current 3D point to specified 3D point.
LinePlotData - From specified data, draw the plot.
Matrix - Create a 2D matrix.
Move3Abs - Move to specified point in 3D.

Move3Abs - Move to specified point in 3D.
NetAccept - Accept connections from the network.
NetOpen - Place a call on the network.
NetRead - Read data from a process across the network.
NetWrite - Write data to a process across the network.
Persp - Set the 3D perspective.
Pick - Determine if an object is registered at this mouse position.
PolyCurveFit - From specified data, create a curve.
PolyFill3D - Draw a 3D image from a contour map.
PutObject - Put specified object in window.
Register - Register an object at specified position.
ScreenDump - Print the window contents.
SelectColor - Select current color by an index.
SetAxesType - Set the axes type to use on subsequent calls.
SetCurrentWindow - Set operations to point into specified window.
SetWorldCoordinates - Set the world coordinates as they relate to the screen.
TitleWindow - Place a title on the current window.
TitleXAxis - Place a title on the X axis in current window.
TitleYAxis - Place a title on the Y axis in current window.
Transform - Transform world coordinates to real.
Vector - Allocate storage for a one dimensional array.
WorldRotate3 - Rotate a point in a 3D space.
WorldScale - Scale the window by specified world coordinates.
XTitle - Title the window for use by the X Window System window manager.

7. CONCLUSIONS

XCLIPS is readily applicable to solutions that require graphical expressions. The XCLIPS rule is a very flexible control mechanism for handling icons, mouse and keyboard devices, and drawing simple to very complex pictures.

XCLIPS based graphics solutions to very complex problems tend to be straight-forward and compact. With the addition of communications support, transparent distributed XCLIPS applications are as easy to build as monolithic systems.

The one area where XCLIPS is difficult to apply to data intensive problems is in the area of performing complex computations. If the language employed an operator precedence grammar, this difficulty would be eased, but the language would become less uniform.

8. FURTHER READING

A good understanding of the general graphics principles used in extended CLIPS is contained in the following reference: Newman, M.N., Sproull, R.F., *Principles of Interactive Computer Graphics*, McGraw Hill Book Company, New York.

For information on graphics programming in the PC environment under MS-DOS, two books are especially helpful: Wilton, R., *Programmer's Guide to PC & PS/2 Video Systems*, Microsoft Press, Redmond Washington and in Microsoft Corporation (1987), *Microsoft C 5.1 Optimizing Compiler Run Time Library Reference*, Microsoft Press, Redmond Washington.

The following book was used a reference for X Window System graphics programming: Nye, A. (1988), *Xlib Programming Manual Vol. 1*, O'Reilly & Associates, Sebastapol CA.

The following book describes the algorithms used for solving systems of simultaneous equations and polynomial curve fitting: Chapra, S.C., Canale, R.P. (1985), *Numerical Methods for Engineers*, McGraw-Hill Book Company, New York.

The following document describes advanced programming topics in CLIPS: Giarratano J.C., *CLIPS Reference Manual*, Johnson Space Center, Houston TX.