The IUE User Interface*

Daryl T. Lawton, David Dai, MaryAnn Frogge, Warren F. Gardner, Heather Pritchett, Andrew T. Rathkopf, Ian Smith and the IUE Committee†
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract

We describe the design and initial prototyping of the user interface of the DARPA Image Understanding Environment (IUE) and tools for documentation, tutorials, and publication that will facilitate the use and adoption of the IUE.

1 Introduction

The user interface of the Image Understanding Environment (IUE) is intended to provide flexible, simple, and powerful tools for exploring data, algorithms, and systems [Mundy and others, 1992]. The general principles of object oriented design used in developing the IUE object hierarchy and programming constructs have also been applied to the interface: abstraction over common operations to provide a small number of interface objects which can be freely combined by a user. The interface has been designed to have a consistent interaction with IUE objects and their semantics, especially the abstraction in the IUE object hierarchy. Thus, the display and browsing operations are sensitive to the class similarities for objects such as images, image registered features, and spatial objects. Using and becoming comfortable with the interface hopefully will not involve understanding a large numbers of unrelated things.

An equally important part of the user interface is what it does not develop. The IUE user interface must leverage extensively off of existing (and emerging) interface and graphics packages and standards. The interface needs to be supported by ongoing and future developments in software environments and graphical user interfaces. This is critical for the long term use of the IUE. We can depend on continuous advances in all these areas that the IUE will need to take advantage of in terms of capabilities and cost.

To realize this, the interface is being developed in terms of three levels. The Graphics Level is the underlying "machine independent" package for display and graphic operations which tell the screen what to do. Examples would be X, GL, OpenGL, and PHiGS. The Interface Support Level involves packages for the creation and rapid prototyping of user interfaces and related tools which are built on top of graphics level software. This also includes the tools found in the selected software development environment such as editors and debuggers. The Image Understanding Environment User Interface (IUEUI) Level consists of the interface objects specialized for image understanding. This includes such things as object displays, plotting displays, several types of browsers, and structures for describing the interface context. The IUEUI consists of a small set of objects which can be freely combined. The specifications of these objects is relatively independent of the other two levels although much of the current prototyping activities are directed towards understanding how to best realize the functionality of the IUEUI objects with respect to these two levels, especially for accessibility and limiting the eventual cost of the IUE for users.

The basic functional components of the IUE interface are:

- Displays: These deal with mapping spatial objects and images (or sets of spatial objects and images) onto two-dimensional display windows. There are types of displays for displaying images and image registered features, for plotting functional relations between attributes and components of spatial objects; and for displaying surfaces.

- Browsers: These deal with presenting textual and symbolic information about objects. There are different types of browsers for operations such as inspecting the values in a spatial object, for performing interactive queries with respect to databases and sets of objects, and for inspecting relational graphs and networks.

- Interface Context Descriptors: These are for describing the state of the interface and interface objects. Examples are such things as the current color-look-up table for a given display; the current

*This research is supported by the Advanced Research Projects Agency of the Department of Defense and is monitored by the U. S. Army Topographic Engineering Center under contract No. DACA76-82-C-0016
†The members of the IUE Committee are: Tom Binford-Stanford; Terry Boult-Columbia; Bob Haralick, V. Ramesh-U. Washington; Al Hanson, Chris Connolly-Umass; Ross Beveridge-Colorado State; Charlie Kohl- All; Daryl T. Lawton-Georgia Tech; Doug Morgan-ADS; Joe Mundy-GE; Keith Price-USC; Tom Strat-SRI

289
display window or browser; and links between interface objects which describe related views. This information supports intelligent default behaviors.

- **Command Language and Command Buffer:** A user can control his interaction with objects using an interactive command language. The commands can be used in code and to create scripts. This also provides a complete description of the functionality of the user interface.

- **Simplified, programmable access to Graphical User Interface (GUI) objects:** This is intended to provide programmer access to several of the objects commonly found in Graphical User Interface Construction Kits such as knobs, sliders, text buffers, menus. These can then be used in applications and to extend the interface.

With the exception of GUI objects, we now look at each of these in more detail. Our focus here is on the functional components of the user interface and their attributes.

## 2 Object Displays

Object Displays are for viewing and interacting with objects by mapping them onto two-dimensional display windows. This involves nearly all IUE objects: images, curves, regions, object models, surfaces, vector fields, etc. Object displays involve a wide range of actions such as displaying an image and image registered features; displaying networks of objects such as stereo images, multi-resolution pyramids, image sequences (and this can involve having several linked windows for the different images); cycling through displays of the different components; or mapping the different components onto different planes of the display buffer and combining the images through transparency or color addition); display of models and predicted segmentations as overlays; and interactively inspecting and manipulating displayed objects and applying operations to them.

There is a strong relation between spatial objects and displays. Most IUE objects are expressed as relations between sets. In displaying an object, values from one set are used to specify a position in a display window and the corresponding values from another set are used to specify an attribute such as intensity, color, overlay, transparency effect, etc., that is displayed at the position. A basic example is an image, where one set is described by the indices of the coordinate system of the image and the other by the color or intensity values associated with the particular image coordinate. A discrete curve is a mapping from integer indices onto two-dimensional positions with respect to an image coordinate system. Displaying a curve as an overlay on top of an image, involves mapping two-dimensional positions along the curve onto window positions using the same transformation that was used for the display of the image. The color/intensity of the display at these points can be based upon registered values associated with the curve (such as approximated curvature). For example, a user might want to display an intensity image in 8-bits of grey-level intensity and then overlay extracted curves on top of this with the display of curvature values along the curve mapped onto 8-bits of red intensity.

We refer to the operations that are involved with specifying the mapping onto a screen position the position methods. These involve operations such as panning, zooming, perspective transformations, and warping operations. The machinery for this comes directly from the coordinate system transformation methods. The operations that involve mapping onto a particular window value are called the value methods and these involve such things as setting up the CLUT, point-mapping functions (functions applied to the value at an object position prior to display), transparency effects, etc.

The basic processing flow for displays is shown in figure 1. Displays take place with respect to a context which involves such things as the current object, the current description of the transformation from an object to the display window, the current color look-up table, links between IUE objects, and several other attributes. Several display operations involve setting up these contextual attributes. Displaying an object, such as an image or image registered features, involves iterating over the object and applying the specified position methods to determining the position in a window at which to generate a display and also applying the specified value methods. In interactive processing, a selection operation is performed with respect to the display context to return a value at a selected location. Graphics are also done with respect to the display context. (The processing flow in figure 1 is idealized in several respects. Many display operations don't involve iterating over an object but manipulating the color look up table and display buffer directly. Rendered objects or displays which involve warping or interpolation are more naturally expressed as iterating over the display window itself. Displays can also involve a discrete sampling of other objects using square pixel neighborhoods).

The object display methods are organized into the following classes:

- **Value Methods:** These are methods that control how values in the specified object(s) get mapped onto screen: attributes such as color and intensity; how to configure planes in the screen buffer for the display of color images; how many panes to use for overlays; particular functions and conditions to apply to object values prior to display. This includes operations such as overlays, mapping onto different color bands, transparency, and others.

- **Position Methods:** These are methods that control how positions in specified object(s) get mapped onto a display window. This includes operations such as panning, zooming, perspective views, and some types of warping.

- **Display Window Attributes:** These involve controlling attributes of the display window such as position, size, attributes of the title bar, event handling for the mouse; and resizing operations.

- **Link Methods:** For linking different displays (and browsers and GUI widgets). Examples are window to window zooming, display of stereo and pyramidal
images in multiple images. This involves creating links and specifying the operations and transformations associated with links between interface objects.

**Interaction Methods:** These involve interaction and manipulation of displayed object(s) in a display. Operations include object selection, recovering object positions and values from a mouse click, applying functions to selected objects.

**Graphics Methods:** Display registered graphics for drawing lines, text, three-dimensional objects, etc.

**History Methods:** Many objects can be mapped to the same display window. These are methods to coordinate displays over time, such as cycling through an image sequence, playing an animation of displays.

**Archiving Methods:** Printing an object display, writing an object display to file, writing an object display to video tape.

### 2.1 Value Methods and the CLUT

Value methods are used to specify how values in an object are mapped onto display window attributes such as color and intensity. There are several types of methods for this. **CLUT-segment methods** involve setting up a color look up table (CLUT). They involve creating named segments and associating some number of bits with each segment, such as specifying 8 bits for red, 8 bits for green, and 8 bits for blue. **CLUT-value methods** involve associating color and intensity values with CLUT indices. These operations can be applied to CLUT-segments. For example, the red component of the color look up table can be mapped onto red values in several different ways: by linear interpolation between specified shades of red or by a spline through specified color values. **Overlay methods** involve setting up overlay planes. Overlay planes can be displayed and cleared separately of underlying intensity displays. **Object-mapping methods** deal with taking object values and mapping them onto color table indices. For example, the CLUT-segment for red intensity could be set for a linearly interpolated 255 shades of red but the actual object values in an object could range from values such as -1000 to 2000. The **Linear object-mapping method** specifies to linearly interpolate from this range of object-values to the available shades of red. There can be a linear mapping from the object onto color table indices, but the color table may be set up for a non-linear mapping onto actual intensities displayed on the screen. **Value-function methods** are user specified functions that are applied to specified object values to map them onto CLUT indices. Examples are conditional expressions that determine what value to map a particular object value onto. In Lisp this is straightforward. In C and C++, it involves a run-time interpreter which we want to be of as minimal complexity as possible. Other value methods deal with transparency, blinking, and logical operations on bitplanes.

### 2.2 Position Methods

**Position methods** are used to specify mappings from spatial objects and images onto two dimensional display windows. They specify where to display something in a display window. For example, position methods are used to specify pan, zoom, and scaling parameters to relate an image to a display window.

The position methods and transformation networks used by the interface are defined by the coordinate-transforms used in the IUE. The interface generally requires transforms of images and image registered objects onto display windows and simple types of interpolation and sampling. More complicated mappings, such as image warping and the generation of rendered objects for a specific sensor use methods from the sensor and scene classes and image packages for warping. These are either used to generate an image which is then displayed or the object specific display methods can be invoked through the interface.

The coordinate transformations and networks are very important for the interaction methods. In this case, the user indicates some position in the display window and then the mapping from the object to the display window is inverted so that the corresponding values and position in the object can be determined and accessed.
The object display is able to do this for images and invertible geometric transformations. For others, such as interacting with a potentially complex, rendered solid, methods from the other classes, such as the sensor and scene class, are needed. It is also possible that there are other representations of a rendered scene, such as an image-registered depth map that contains pointers to all the surfaces that project to a given pixel, ordered by depth that can be generated to simplify the interactive processing.

There is often hardware supported pan and zoom for images that should be accessible through the position methods, even though this is machine dependent.

2.3 Interaction Methods

The Interaction Methods are for interacting with and inspecting objects through the context associated with a display, such as the current object to display transformation. The methods associated with this are built on top of the event-handling mechanisms of the supporting environment. Interacting with an object through a display involves using the position mapping from the object to the window. This is straightforward if the mapping is invertible and there is no interpolation or warping. This is usually the case for images and image registered objects. It can also involve geometric intersection using the ray of projection corresponding to a selected window position. For other objects, such as closed analytical surfaces, the reverse mapping is more complicated and involves general spatial object methods that need to be accessed by the interaction methods.

The current object to be interacted with can be explicitly specified or selected. Selection can require disambiguation if there are multiple overlapping objects or complex spatial objects. The user may be required to use a spatial index image (an image of pointers to objects which occupy a given position) or use geometrical data base operations in the IUE. Both are potentially expensive and don't reflect operations specific to the interface but are general IUE spatial data base operations that need to be invoked through the interface to return the selected object(s) and object position from the object to display mapping.

There can be a variety of interaction devices (minimally it should specify a screen location), but we are assuming a mouse with at least two distinguished buttons and text-input from the keyboard. In the interactive mode:

- The current position of the mouse is stored in variables for the \( (\text{mouse-x}, \text{mouse-y}) \) of the current display window. Associated with this is the corresponding positions \( (\text{current-position}) \) and values \( (\text{current-values}) \) in the specified objects. The default is to only deal with geometry only to inverting coordinate system transforms and not sampling or interpolation relative to the objects. Since several objects can be interacted with at the same time, these lists will consist of lists of positions and values.

- When the mouse is clicked, the values for the window position, the corresponding object positions and values are stored in separate lists. Interactively selected Functions can then use items in these lists as parameters.

- A user can associate functions and command-sequences with keys and mouse-states so the functions can be called interactively. The functions are stored in a table index by mouse-state or keyboard-event. The functions can be a sequence of specified interface commands and can be interactively applied to the values in the different lists. Functions are selected using keyboard input (numbers) or mouse-state (mouse-down, mouse-up, mouse-drag for the left, right, and, if it is available, the middle mouse button). Function selection may also be based upon a count of the number of mouse-clicks for specified mouse button.

- There may be a default spatial index associated with a display window. This is memory intensive but can help with a lot of the interactive operations, especially the selection operations.

2.4 Graphics

Often a user will want to perform graphical displays of text, two dimensional graphics, and three dimensional graphics. Examples are annotating a display, indicating where some action is occurring (the position of an epipolar line, translational flow paths, etc.), projecting a wire-frame of a model onto an image. Much of this functionality will come directly from an existing graphics packages that the IUE will utilize. The graphic displays need to take place in three different modes:

- they can occur in the coordinate system of the display window. In this case display only occur with respect to the window coordinate system.

- they can occur in the coordinate system of a displayed object, such as drawing a line with respect to the inverse mapping from window to object coordinates

- generating corresponding IUE objects. Thus, in drawing a line in image coordinates, an corresponding instance of an IUE line object would be produced. When the wire-frame model is displayed, each line-segment and junction would be created as an object in the IUE. This is very useful for producing data for testing routines. This mode can be coupled with the interactive processing mode to allow for the interaction creation of data. This maybe restricted to relatively simple objects such as polygons, curves, and so forth.

2.5 Types of Object Displays

We have distinguished four types of object displays:

- The image or pixel display is for viewing images and image registered features.

- The local graphics display displays objects by mapping their values onto parameterized graphic objects such as lines and cubes. Examples are displaying vector fields and edges.
• The surface display is for displaying objects that get mapped onto mesh or rendered surfaces.

• The plot display is for displaying functional relations between objects. Examples are one-dimensional, two-dimensional, three-dimensional graphs, histograms, scattergrams, and views of functions and tables.

These different types are distinguished by specific methods but all inherit a large number of similar methods from the general display class. For example, overlay operations are similar for a surface display and for an image display, although they can look quite different. (In one case they appear like drawing in solid colors in image registered coordinates on top of a displayed image and in the other it would be rendering the colors onto a displayed surface).

2.5.1 Local Graphics

Local Graphic Displays are a subclass of object displays which map object values onto parameterized graphics, such as a line, a square, a perspective view of a cube, Chernoff faces, or a user-specified function. A common example is a vector display which will map each component from a pair of image onto the x and y components of a vector. Using the general display methods, the vectors can be displayed as an overlay on top of an image or through indices in a CLUT. For visualizing three-dimensional attributes in register across an image, the user can display unit cubes with their orientation computed from the specified components of the display object. The graphic display can be a piece of graphics code which will be positioned to the projected location of the pixel.

There will be specialized local graphic displays for vectors and different types of edges because of their heavy usage. It will be possible to display the horizontal and vertical edges in the cracks between pixels or to place a single edge at the center of a pixel with its orientation determined by the specified components objects.

2.5.2 Plot Displays

There are several different types of plot displays: one-dimensional, two-dimensional, three-dimensional graphs, histograms, scattergrams, perspective views of functions and others. Examples of plot displays can be found in several data visualization packages and mathematics packages such as Mathematica and GNUPlot [Wolfram, 1991]. In using such packages in the IUE, it is important to bear in mind cost limitations on bundled software and potential problems with data compatibility and speed. Plots also need to be compatible with the general display methods for such things as

• mouse interaction methods: for selecting a position in a graph and then having access to the domain point and the range point. An example is interactive segmentation from a histogram displayed as a plot

• links between plot displays and other types of displays

• most of the view transformations for such things as scaling and zooming

• overlaying plots in different colors

We are currently exploring the use of the plotting capabilities in GNUPlot to be used for use in the IUE. It is essentially free and all the source code is available.

3 Browsers

Browsers are used for interacting with text-based or symbolic descriptions of objects. They are used for actions such as queries over set of objects, determining and inspecting relationships between objects, process monitoring, and inspecting values in an object. The browser and browser-related classes are being designed so they can readily be built on top of existing interface construction kits.

There are two general types of browsers: Field-Browsers and Graph-Browsers of which only field browsers are currently being implemented. Field browsers are built from component objects which are found in several GUIs:

• A field appears as a rectangular box which can be filled with text, icons, colors, colored text, text in particular fonts, or user-specified graphics. Fields can have actions associated with them when they are selected or a user changes the values in them.

• Fields can be organized into connected horizontal or vertical field groups where each field is a unique index in the Field Group. The fields in field groups will generally have different objects displayed in them. An example comes from the object registered browser where a field group can correspond to a display of registered values from different objects. For better visualization, these can be displayed in different colors, fonts, etc., in addition to their position in the field group. A field group can also have a distinct boundary.

• Field Groups can be organized into field matrices where in each group as a unique index set in the field matrix. Objects and sets of objects can be mapped onto the matrix.

• Field Matrices can be scrollable as a way to control the mapping of an object (or object set) onto the Field Matrix

We distinguish between four types of field browsers which inherit from the general Field browser class:

• Object-Registered Browser: This contains values extracted from a spatial object, such as the intensity values in some square neighborhood of an image. Depending on the dimensionality of the object (or relationships between component objects), this can be presented as a one-dimensional array, a two-dimensional Array, or multiple two-dimensional arrays and describe curves, images, image sequences, pyramids.

• Set/Database Browser: This is presented as an array of fields. Each row of fields corresponds to
selected attributes of a particular object and each column corresponds to common attributes over the set (or database) of objects. An example would be browsing the database which describes the current active objects in the IUE to find the most recently created image from some operations.

- **Object Attribute Browser**: Each row corresponds to the value of an attribute for an object. This is used for inspecting a single object.

- **Hierarchical Browser**: Useful for text based inspection of graph structures and trees. When an item is selected, the related items (along some relational dimension) are displayed in the next column.

The methods associated with browsers are very similar to those with displays, suggesting a more general IUE Interface object class. The position methods for browsers involve how an object (or set of objects) gets mapped onto the fields of a browser. For object registered browsers, these are essentially the same as with displays (see figure 2). The fields are analogous to pixels in a display window, although they can be filled with textual information. For Database browsers the position methods specify how objects are mapped onto rows of the browser and how attributes are mapped onto columns (see figure 4). The position methods for mapping from graphs and networks onto a hierarchical browsers involve keeping track of different paths through a network and nodes and arcs that have been traversed. Browsers can also be linked to browsers, displays, and user specified interface widgets. The following examples have been implemented using the FORMS GUI kit on SGI (Overmars, 1991).

### 3.1 Object Registered Browser

The object registered browser is used to inspect the values in a neighborhood of a spatial object. A common example is inspecting the image values about a selected point. It is very much like the display of a spatial object in a display window, but instead of the values being mapped onto window positions and screen intensities and colors, values are mapped onto field locations and general field attributes such as colored text in specified fonts, colors, and icons. The attributes of and the specification of the mapping from an object onto an object registered browser is shown in figure 2. A set of spatial objects are mapped onto a matrix of fields in the object registered browser. This mapping involves several parts: a coordinate transform from the N-dimensional spatial objects to the two-dimensional object registered browser; the type of interpolation to be performed if the mapping doesn’t involve discrete values; what to do when browsing beyond the boundaries of the spatial object. Also shown is a navigation tool to interactively access position methods to position the object registered browser with respect to a set of spatial objects. The browser is linked to a display window which shows the position of the browser with respect to the bounding rectangular prism of the spatial object. This display would be updated when the browser is moved with respect to the spatial objects. Figure 3 shows an implemented two-dimensional Object Registered Browser in which is displayed two images and the computed difference of the two images, each in separate fields. Each image is displayed in a different color and the field containing the difference image uses the background color to encode the magnitude of the difference.

### 3.2 Set/Database Browser

The set/database browser is for inspecting the attributes of a set of objects. It enables interactive queries can be performed via the browser. This is especially useful for keeping track of instances of objects (an object selected in a Set/Database browser should probably default to the current object so it could be displayed immediately). There are two structures used for describing the mapping from a database onto the browser. One is the set of selected attributes which correspond to the columns. The other is the current set of items which satisfy a query and the indices into the first and last element of this set which are displayed in the corresponding rows of the browser (see figure 4). Figure 5 shows an example using the Set/Database browser to inspect a set of line segments and then to sort them by slope.
3.3 Object Attribute Browser

An Object Attribute Browser is for inspecting the attributes of an object or several objects with the same types of attributes. It uses an ordered list of object attributes to determine which attributes of the object to display. Figure 6 shows an example of a Object Attribute Browser applied to the attributes of an image object.

3.4 Hierarchical Browser

A hierarchical browser is for inspecting graphs and network objects. Instead of one large field-matrix, it consists of linked Nx1 field-matrices. Each column corresponds to a set of nodes. When a node is selected, the types of relations (arcs) are displayed in the current arc browser. When a type of arc is selected, the nodes with that type of relation are displayed in the adjacent (right) column. Several structures are used to describe (and update) the mapping from the graph onto the successive browser columns. The current node is the most recently selected node. The current path is stored as well as the nodes that have been visited. Figure 7 shows a hierarchical browser applied to an instance of a polyhedral mesh.

4 Interface Context

There are several data structures for describing the context of the interface. These are used for intelligent defaulting and for saving the state of the interface. These include:

- Object-Display-Mapping: Structures which describe the mapping from an object onto a display. This includes the viewing transformation between an object and display window, the value-mapping of how the object is displayed and a reference to a particular CLUT.

- Object-Browsing-Mapping: Structures which describe the mapping from an object(s) or database onto a browser.

- Display Context: Structures which describe the current context for a display for such things as the current window, the current object, the current object display mapping, the current display command, the current mouse-selected object position and value, the lists of interactively selected object values and positions. For example, if neither a display or an object is specified, it will default to the most recently used.

- Browse Context: Related structures for browsers. Such things as the current browser, the current data base, query history, and others.

- History: The sequence of display or browsing actions for a particular window or browser are saved.
and can be reaccessed and used for creating animations. In addition which objects have been displayed or browsers is also stored.

- Default layouts for windows and browsers: The desired layout of windows and browsers can be saved and be available to a user when he starts using the IUE. Users may prefer different interfaces (arrangement and instantiation of the basic interface objects) depending on the task or level of sophistication.

- Object Display Links: A structure which describes the concatenation of a display or browsing operation between IUE interface objects.

The context description is an extension to the underlying context usually provided by the graphics level. It should be possible to read and save context descriptions.

4.1 Links
Links support operations such as window to window zooming, displaying the same object from different views or using different value-mappings, and controlling displays using interactive widgets like sliders and knobs. Linked displays are useful for displaying composite data such as stereo image pairs or pyramids. When something happens in a parent display (or browser), another display will perform an action using information from the parent display. The action can be a display operation operation or executing a sequence of commands associated with the link, such as a set of commands from the interactive command language.
The mapping between a spatial object and a display window in one window can be concatenated with the display specification in another. A common example is using one window to zoom onto the display in another or using one window to display a selected portion of another (Panning and Zooming are so common they will be directly supported via an interactive tool).

We have specified constraints on links to avoid many complex and pathological things that can happen. Linked displays and browsers are only updated when a display action is performed, not when changes are made to the displayed object. Individual links are bi-directional, but no cycles are allowed in the graph formed from all of the links between IUE interface objects.

5 Command Language

Users will be able to specify all interface actions through an interactive command language and be able to access all the functionality of the interface. Display operations can be performed interactively through the command buffer. The command language will have intelligent defaults and abbreviations (such as displaying the current window if none is specified). In addition, the commands will be be usable in non-interactive code for creating scripts and general display routines. All of the functionality of the interface is accessible through an interactive command language which encompasses the overall functionality of the interface.

A concern with the interface command language is that it becomes another language that people will need to memorize. This is not an issue for development in Lisp since the display operations can be called interactively like any other function, but it is a significant issue with C++. We intend for the command language to be as simple as possible, with a limited syntax. Most arguments are specified via keywords and correspond directly to interface object methods. There are also defaults for command specification. And the IUE will probably eventually support intelligent prompting to complete the commands. The general syntax is

\[
\text{IUE-interface-object object-set [keyword arguments]}^* \\
\]

For example,

\[
[\ast w1* \text{ image1 :p}] \\
\]

means to display image1 using a pixel-type display in window \*w1* using the current display context associated with the display in window \*w1*. The brackets are used to indicate separate commands. If the last display operation was of type :p in \*w1*, then only:

\[
[\text{image1}] \\
\]

needs to be specified. More detailed examples are presented below.

An important operation for displaying spatial objects is the ability to apply functions to objects prior to displaying or interacting with them. These operations almost always don't involve creating a new object. An example is manipulating the underlying color look up table to perform a thresholding operation. In this case, there is no thresholded image object produced, only what is displayed in a window. This goes by many names in different systems such as Pixel Mapping Functions, Dynamic Color, Generalized Color Look-Up Tables.

There are two aspects to such functions. First, there are limitations on the types of functions that should be specified for application to an object when it is displayed. Operations such as zooming, panning, manipulating the color look up table, specifying which planes in the display buffer are used, and simple point-wise algebra with limited conditional evaluation, are very useful and will be supported. But, it doesn't make sense for operations such as generalized warping or detailed processing over a neighborhood or generalized intersection to be done by via interface commands. Second, there are also language specific aspects for specifying function application to objects prior to display. In Lisp, it is straightforward to pass lambda expression or closures which are applied to each position or value prior to display. In C, this requires a library of standard functions and an interpreter.

In the actual operation of the IUE, it is not necessary that all interactions take place through this command language: some will be invoked by menus and special keys and refer to the current display context. An important part of the design of the IUE interface entails how commands (and which commands) are mapped onto menus and other interactive devices. This is especially important since the interface will support a wide community of users ranging from naive users who are interacting with hardened applications to developers. Naive users may want lots of interactive devices such as specialized menus while experienced users will want more powerful, abbreviated commands. Advanced users will become very adept at shortcuts that should be provided.

5.1 Examples

The following presents some examples of specified display operations using the command language.

\[
[\ast w1* \text{ image1} \\
\text{:p} \\
\text{:linear 0 128 *screen-min* *screen-max*}] \\
\]

This would display to window \*w1* using the current defaults. The range of object values from 0 to 128 are linearly mapped onto the range of values \*screen-min* and \*screen-max*.

\[
[\text{image :p}] \\
[\text{edge-image :overlay red}] \\
\]

An image is displayed in the current window using a pixel-type display. The edge image is then overlayed on
top of this. Wherever the edge-image is equal to 0 nothing is displayed in the red overlay plane and wherever the edge-image is equal to 1, the corresponding pixel is set in the red overlay plane.

\[\text{:overlay-colors (red, green, blue, violet)}\]
\[\text{image :p :value-function}\]
\[\quad (\text{if (image.value > 10) red blue})\]

The first command tells the current display to use the specified overlay colors. The second will display red in the overlay plane at a screen pixel corresponding to an image pixel if the image value is greater than 10, otherwise it will display blue.

\[\text{spatialIndexImage :p :value-function}\]
\[\quad (\text{if (label-image.value = NULL)}\]
\[\quad 0\]
\[\quad (\text{length (spatialIndexImage.value}))\]
\[\quad :\text{linear 0 20 0 *screen-max*}]\]

This function displays a spatial index image (an image where each pixel contains a list of all the objects which occupy that pixel). The value function determines the number of objects in this list and the linear function maps this onto available screen intensities.

\[\text{image1 image2 :p :value-function}\]
\[\quad (\text{image1.val - image2.val})\]
\[\quad :\text{linear -20 20 *min* *max*}]\]

This function displays the difference between two images. Other common value functions would be for type conversion and display histogram transforms. The user can also specify functions in the interactive mode to be applied to the values in the different queues. For example:

\[\text{image :i}\]
\[\quad :1 [\text{p :overlay-plane clear}]\]
\[\quad [\text{p image :value-function}\]
\[\quad (\text{if (image.value > object-values[1])})\]
\[\quad \quad \text{red blue})]\]

The user has selected an image location with a mouse click and the corresponding queues have been filled with the window and object positions and values. Thereafter, when the user hits the terminal key 1, the overlay planes will be cleared and all image locations with a value greater than the value at the selected image location will be displayed in red, otherwise blue, in the overlay planes. \text{image.value} is a dummy variable that refers to the current value in image which is being displayed. \text{object-values[1]} refers to the value selected using a mouse click in the display window and stored in the object-value queue. \text{red blue} refers to globally defined overlay colors. Recall that the \text{value function} specifies the operation to be applied to an object value to map it onto a screen intensity or color.

\[\text{[link *w1* :zoom 2 2 :pan 50 50]}\]

This links *w1* to the current window and concatenates a zoom and a pan transformation.

\[\text{RegionDB :p}\]
\[\quad :\text{positions RegionDB.locations :values RegionDB.textureMeasure :linear 0 100 *min* *max* :red-8}]\]

This says to display the RegionDB in the current display window with the positions coming from the locations attribute of the regions in the regionsDB and the values by taking the Region DB texture mappings and using a linear mapping from these onto screen intensities in 8 bits of red.

\[\text{[*w1* histogram :plotid ]}\]
\[\text{[*w2* image :p]}\]
\[\text{[*w1* histogram :i}\]
\[\quad :1 [\text{min = object-values[1].x}]\]
\[\quad [\text{max = object-values[2].x}]\]
\[\quad [\text{[*w2* image :p}}\]
\[\quad \quad :\text{value-function}\]
\[\quad \quad (\text{if ((image.value > min) &&\}}\]
\[\quad \quad (\text{image.value < max))})\]
\[\quad \quad \text{blue red})]}\]

This is an example of plotting used for interactive histogram segmentation in which the interaction methods lets us click on the axis of a plotted function to returns the x coordinate and the y-value of the displayed object and then use these values to specify peaks in a histogram. Here the user has plotted a histogram in *w1*. He then selects the range of values by clicking on the displayed histogram. The current-object-value contains the x and y value from the displayed histogram. These are stored in the local values min and max. When the user hits the key 1, the selected range of values are displayed in the blue overlay plane in *w2*.

6 Additional Features

Even though our focus has been on developing the core functionality of the user interface, there are several other features that have been considered for use with the interface. Some of these can be built on top of the interface objects and operations described previously. These are important candidates as packages and libraries to augment the core IUE.
One important area involves interactive task management tools. Examples can be found in the data-flow editor in the Cantata portion of Khoros and the Task editor in KBVision. Another area that several people feel is important is developing graph browsers are for the display of graphs and networks, generally representing object or values as nodes and using links to describe relations. Graph browsers can have difficulties when trying to display several nodes with arbitrary relations between them in that the connections between the nodes can begin to obscure the over all display. A typical use would be for the display of a constraint or coordinate transform network.

There are probably hundreds of nice interactive controls for displays and visualisation that exist in different environments, such as interactively manipulating the object-value to screen-intensity function by interactively shaping a function; selecting color look-up tables; modifying color look-up tables; interactively building display commands using templates or command browsers; floating palettes of interactive drawing tools; etc. In general, such tools can be very useful, but it is extremely important that there be a consistent look and feel with different applications that are based on the IUE. This will be partially achieved by depending on the underlying graphical user interface to supply the basic interface objects.

Other useful interface tools are:

- Interactive Selection and Modification of color look up tables and display mapping functions; cycling through different color look up tables.
- A dialog box for setting up system defaults and initializing characteristics of the IUE: initial layout, font selected, level of expertise, etc.
- Access to and Integrated use of Established Visualization Packages: There are several data visualization products and it would be nice to have a modular interface to these.
- Mensuration tools: Such things as rulers, grid overlays, and the use of multiple cursors mark of distances and points of reference. These probably can be built on top of basic interface capabilities and the display of IUE objects (in particular, the display interaction methods and IUE objects such as bit-mapped regions, line-objects).
- Interactive Object Creation (Draw Objects): It should be possible to create object interactively. This is useful for creating idealized data for testing and development. This should be supported by the display interaction methods and access to the instantiation methods associated with spatial objects.
- Incorporating Hardware Accelerators: So the interface and the IUE in general can modularly incorporate different hardware accelerators.
- Display Buffer Optimization: The display buffer itself is a short term memory for manipulating the view of a displayed object. A useful feature would be routines to directly access the display buffer or performing specific display operations in ways optimized for particular types of displays.

7 Status - Implementation Trade-offs

We are currently prototyping many different parts of the user interface to complete the functional specification and to answer basic implementation questions about choices regarding GUIs and user interface toolkits. This will help to simplify the job of the eventual integrating contractor. For reasons of rapid development, current implementations are taking place in C and C++ on Silicon Graphics Computers using the GL graphics library, Motif, and the FORMS user interface toolkit. We have been able to put the general display object and the different browsers and hope to use these as initial browsers and displays specialized for use with the Data Exchange Format. We are exploring extensions to GNUPlot so it is compatible with the methods associated with the general display class and can provide an inexpensive plotting package. We are also evaluating OPENGL as a possible machine independent graphics package.

8 User Facilitation Tools

The IUE will be supported by on-line documentation and tutorials. The tools for implementing these will also be available for enhanced communication and publication by scientists and developers who use the IUE. While there is significant activity in developing documentation and hypermedia toolkits, they remain largely machine dependent with no clear standardization. We are developing a simple documentation tool called Knowledge Weasel (KW) which is based on Lucid Emacs 19 and existing media editing tools.

Knowledge Weasel (KW) is a presentation and authoring system designed to support annotation using several different types of media. A simple analogy for KW is reading a book or attending a lecture and being able to make diverse types of comments and annotations on the material. In reality, such unrestricted annotations and comments made with respect to real books and lectures could create a significant mess (especially if made by several different people), so in developing KW we have extended this simple metaphor in several ways. The first is to provide a general format for annotations that can include several different types of media. An annotation is a common record structure wrapped around instances of different types of media such as text files, sound, drawings, postscript files, GNU-plots, code running in the GDB debugger, and others. Annotations are implemented much as a property lists in LISP with attributes and values and are displayed as buttons with an associated region of support. When an annotation is selected it performs an operation specific to the type of annotation selected. Annotations are created using existing media editing tools for operations such as recording a sound, drawing packages, calls to other branched processes, grabbing a portion of the screen. The second extension has been to develop different types of navigation, organization and presentation tools to keep users from feeling overwhelmed with a great deal of possibly
irrelevant information. Users can prune the set of annotations that they want to deal with and also how they are displayed. Annotations are structured to make possible intelligent processing, perhaps eventually including rule-based processing for automatic presentation and "ferreting" of information (hence the name).

We are implementing KW on Lucid Emacs 19 which is in turn based on the X window system. Lucid's implementation of Emacs Lisp provides primitives for handling display attributes such as windows, fonts, and colors. Lucid Emacs version 19 has a built-in Lisp interpreter for Emacs Lisp and this Lisp variant provides a wide variety of primitives that are useful for manipulating text, processes, and/or files. It is available via anonymous FTP on the Internet, and is also the basis of a commercial product. Knowledge Weasel is chiefly written in Emacs Lisp but some parts, such as the part which interacts with the operating system's lock daemon (lockd), is in C and communicates via pipes with the Emacs Lisp portion of the implementation.

Figure 8 shows an example using some of the current features of KW. A user is reading some text about histogram equalization from a text file in Emacs. He has selected some annotations for display (these could be comments from other users or references to other materials). One annotation corresponds to bringing up the corresponding code and then executing it step-by-step in the GNU debugger. One nice thing about the intergration with GNU-tools and Emacs is that it is possible to directly annotate running programs for step by step commentary. The user has selected the button "View Histogram" which is associated with a GNUPlot-type annotation. Annotations are displayed in a larger font of text (which is colored). The actual display of annotations is controlled by a user. Annotations can be conditionally displayed and mapped onto different colors and fonts.

We have begun using an initial version of KW to develop an on-line version of the Low Level Vision course taught at Georgia Tech. We also plan to use it as part of a computer vision algorithms course where students will select a paper from the literature, implement the corresponding algorithms and use KW to develop a tutorial presentation of the paper.

8.1 CD-ROM
A significant instance of technology transfer is the DARPA IU Proceedings and workshop. For the next meeting, we hope to enhance this by having the workshop proceedings available on CD-ROM, and integrated with the Data Exchange Format, a documentation and browsing tool such as Knowledge Weasel, and, possibly, the IUE itself. This will enable an extraordinary type of paper which includes data, code, additional references, animations, and extensive annotations and cross-references.

References
