The Image Understanding Environment: Overview

J.L. Mundy and IUE Committee*  
mundy@crd.ge.com  
G.E. Corporate Research and Development  
Schenectady, NY 12309

Abstract
The Image Understanding Environment (IUE) project is a five year program, sponsored by DARPA, to develop a common software environment for the development of algorithms and application systems. The ultimate goal of the project is to provide the basic data structures and algorithms which are required to carry out Image Understanding (IU) research and to develop IU applications. This paper provides an overview of the IUE and an update on the status of the project.

Introduction
What is the IUE?
The Image Understanding Environment (IUE) is an object-oriented software system which provides the basic data structures and operations required to implement Image Understanding (IU) algorithms. These data structures and operations are based on the classical mathematical abstractions used in IU research, such as pointsets, transformations and topology. The primary purpose of the Image Understanding Environment (IUE) is to facilitate exchange of research results within the IU community. The IUE will also provide a platform for various demonstrations and tools for DARPA applications. These demonstrations and tools will become a primary channel for IU technology transfer. The IUE will also serve as a conceptual standard for IU data models and algorithms. The availability of standard implementations for algorithms will facilitate performance evaluation of new techniques and to track progress in algorithm improvements. The IUE is designed to support evolution and testing of IU techniques and provide an efficient programming environment for rapid prototyping.

History of the IUE
In late 1989, Rand Waltzman of DARPA, then manager for Image Understanding programs conceived and developed a new program called I4US. The decoding of this acronym is Intelligent Integrated Interactive Image Understanding. The name has since been shortened to IUE, for Image Understanding Environment.

The IUE program was announced at a meeting for DARPA Principal Investigators in Scottsdale, Arizona at the end of February, 1990. The project goal, as announced by Rand, was a five year program to design and implement a common software environment for the development and demonstration of image understanding algorithms and techniques.

Rand Waltzman convened and chaired three meetings during the 1990-1991 period to develop an consensus in the IU community about the requirements for the IUE. A number of teams were established to suggest specific application scenarios and propose skeleton architectures for the IUE. In April 1991, these team reports were reviewed, and the IUE committee was formed from representatives of each team. In June 1991, Oscar Firschein replaced Rand Waltzman as the IU program manager at DARPA.

Since April 1991, the IUE committee, along with the DARPA IU Program Manager, has met eight times to develop the design and produce a number of documents which specify the design as well as an IU data exchange standard. In September 1992, the committee provided a draft version of the IUE design to DARPA to provide the basis of a solicitation to select the IUE contractor. On January 8th 1993, the IUE BAA was published in the Commerce Business Daily by the contracting agent, the Topographic Engineering Center.

The basic structure of the IUE is captured by a partial summary of the class hierarchy shown in Figure 1. The figure illustrates the central idea of the IUE design and its relationship to mathematical concepts. The following sections provide a brief summary of the IUE design.

Image
The IUE image object class supports many forms of image data, from intensity images to color images to complex composites such as pyramids. IUE images fall into one of two subclasses: simple or composite. Simple images have two primary dimensions, x and y, with possible additional dimensions such as color and/or time.
Figure 1: A partial view of the IUE class hierarchy. There are hundreds of class definitions in the current IUE, and this figure represents only a sketch to illustrate the general nature of the hierarchy.
A simple image can be defined as a mapping from $Z \times Z$ to $V$, where $Z$ is the set of positive and negative integers and $V$ is the set of allowable pixel values:

$$I : Z \times Z \rightarrow V$$

$I$ may be abstractly viewed as a discrete function of two variables $I(r, c)$ where $r, c \in Z$. For computational reasons it is desirable to restrict the domain of the mapping to a specific subset of $Z \times Z$, usually a rectangular bounded region of the plane. The set of pixels comprising the restricted domain is often called a region-of-interest (or ROI) and, by extension, the logical image defining the characteristic function is also called an ROI. Note that there is no restriction on the connectedness of the pixels in this restricted domain; that is, it is not a true region as defined in region segmentation and may consist of a set of completely isolated pixels. In the IUE, there are two ROIs which affect the domain. While both are called ROIs, their semantics are slightly different. Every image in the IUE has an ROI associated with it, called the image-ROI. This ROI may be explicitly represented as a logical image having the same extents and index set as the original image or it may be implicitly represented as the entire image. In addition, each method has an optional parameter which specifies an ROI, called the in-ROI. The set of pixels comprising the pixel domain is obtained by intersecting the image-ROI and in-ROI.

Simple-images can map onto different slices of the same underlying pixel data, thus avoiding redundant copies in many common situations. To illustrate, an RGB image is represented as a 3 dimensional shared-array. Associated with it are three 2 dimensional arrays: slices corresponding to each of the three color planes. These four objects—RGB, red, green, and blue images—present different views of the same underlying data. In addition to abstract high level interfaces to pixel data provided by the image objects, the IUE will provide a raw-data interface to pixel data. Raw-data objects are one dimensional containers for large amounts of numeric data, and they support direct pointer access to numeric pixel data. The IUE must efficiently support large images (say 10Kx10K) such as those commonly occurring in photo-interpretation tasks. Thus, a direct tile-mapping mechanism must be associated with raw-data objects which allows large images to be block mapped into memory on a demand basis; this mechanism must be efficient enough to support smooth scrolling (roaming) and zooming.

Image-types commonly occurring in image understanding include: A color image with red, green and blue components, where each component is a simple grayscale image. A range image represents a depth image acquired from laser triangulation or a time-of-flight range finder. An image sequence can be viewed as a queue of images indexed by time. For simple-image-sequences queue length is fixed when the sequence is created and all elements must be simple-images of the same size. In contrast, composite image sequences may contain images of any type and may dynamically grow and contract.

The class of composite-images is very broad and the semantics vary considerably between specializations. Composite-images provide the flexibility required to develop objects such as image pyramids and mosaics. An Image-Pyramid is an ordered set of images, each a power of two reduction of the predecessor. This makes several restrictions evident. The first image must be the original, the dimension being a power of $2$, i.e., $2^0$. The depth is of the pyramid, or number of images, is simply $n + 1$. An image of size $2^n \times 2^n$ is said to reside at level $n$ of the pyramid. A Mosaic-Image is a patchwork of images, possibly overlapping, and partially covering an extended 2D area. The proper metaphor is a stack of photographs on a table. Hence, the ge-pixel method returns the pixel value for the first image in the set which is defined at the specified point.

Spatial Object

A key element of the IUE design is the spatial-object which has the mathematical properties of a pointset in $R^n$. The fundamental structures are organized along classical notions of intrinsic dimension, i.e., point, curve, surface and volume. Further distinction is made between implicit and parametric entities. Implicit structures are defined by the vanishing of systems of equations, usually polynomials. Implicit forms are useful for determining incidence or containment. Implicit forms of curves such as the line and conic are used throughout IU research. An example of a commonly used implicit surface is the superquadric.

Parametric structures involve a mapping from a set of parameters to $R^n$. The parametric mapping function is a particular type of relation which defines a mapping between two sets of n-tuples, the Domain and the Range. We further restrict the mapping to be order preserving and one to one. With these properties we can always find a unique point in the domain for a given point in the range and the natural dimension and neighborhood properties are preserved. This is a much stronger condition than is usually associated with the idea of parametric curves or surfaces. The curves here are perhaps more properly called “well-parametrized,” where there is a unique inverse for each point in the range of the curve. A typical example of a parametric curve is the spline. A common parametric surface used in IU is the ribbon.

A coordinate system is associated with the base spatial-object class in order to maintain a consistent definition of coordinates derived from the equational definition of geometric structures. Also associated with all spatial-objects are a bounding-box and centroid point. These ancillary structures enable efficient processing of distance and intersection operations.

There are many other components of the spatial-object hierarchy which are described more fully elsewhere in these proceedings [Ramesh and Committee, 1993].

Coordinates

The geometric relationship between sensors and scenes, among physical objects, and between pixels and the

---

1 Only restricted classes of ribbon surfaces satisfy the unique inverse property.
world has been a core component of the science of image understanding since its inception. Nearly every IU system makes use of coordinate systems and transforms either implicitly or explicitly. The multitude of representations that have been devised, some of which incorporate arbitrary conventions, has been a key obstacle precluding the transfer and sharing of code and results. The following are definitions associated with coordinate spaces.

Coordinate System A coordinate space, in the mathematical sense. It is represented in the IUE by an instance of a coordinate-system class.

Coordinate The coordinate(s) of a point are represented by a vector and implemented as a 1D-array. Coordinates are implicitly associated with a coordinate system.

Coordinate-Transform A specification of a mapping between two coordinate spaces. It is implemented in the IUE by an instance of a coordinate-transform class.

A coordinate transform specifies the mapping between two coordinate systems, and is represented in the IUE by an instance of a coordinate transform class. There is an implied directionality to each transform, and any individual transform may or may not be invertible.

The relationships among coordinate systems and transforms form a graph. Coordinates can be mapped between any two coordinate systems by finding a sequence of transforms that connect them in the graph. There is a potential problem when more than one path exists between two coordinate systems. Ideally, all such paths specify the same transform, but the existence of numerical errors, and the need for high-performance approximations preclude the treatment of all such paths as equivalent. In IUE, the first path established between two coordinate systems is treated specially — all others are considered to be derived transforms. Whenever a transform is requested from the coordinate transform graph, the basal (non-derived) transform is retrieved. Derived transforms can be employed when desired by specifying them explicitly. This policy ensures that coordinate transforms are performed consistently and without introduction of excessive numerical error.

Image Features

Central to any Image Understanding research or application program is the extraction and use of image features. Image features are a part of the general spatial object hierarchy in that they combine both geometric and image signal-theoretic concepts.

Image features provide implementations of methods for extraction, property value computations, display, spatial indexing operations, input, output, grouping, and the various iterators over sets of spatial objects (subsets, all in an area, etc.). Image features are often be used as the basis for the region-of-interest in image processing operations and thus must support geometric and topological operations.

There are several reasons why developing image features as spatial objects is crucial. First, there is a natural correspondence between the sequence of topological constructs, e.g. vertex, edge, and face used for spatial objects, and the descriptions of image features for points and junctions, edges, and regions. Access to this topological representation is especially important for describing composite image features such as linked line segments, adjacencies between regions found in segmentations, and perceptual groups. Second, since image features generally correspond to the projection of three dimensional object models, it is useful to have the same underlying operations and representations used for both of them.

Images features collections can be grouped on a variety of properties such as proximity, alignment, curvature, etc. These grouping operations are supported by various spatial indexing operations such as K-D Trees, quadtrees and the Hough transform.

Image features are discussed in more detail elsewhere in the proceedings [Price and Committee, 1993].

Sensors

Unlike some other aspects of the IUE, sensors are an area of active research where few de facto standards exist.

Two important aspects of sensing in the context of IU are the device and the data produced by the device, represented by the classes, sensor and sensor-model respectively. The class Sensor is the analog of the physical device and is capable of many operations associated with sensing and the production of sensor-models. The output of a sensing operation is stored in an object of type sensor-model. The sensor-model not only contains a pointer to the generated data, it has a copy of relevant sensor-parameters and provides methods to reason about the geometry of the sensor mapping, and uncertainty in data locations and measured values. The sensor may interact with an external device (e.g. a frame grabber) to get real data, or may generate synthetic data either by embellishing a stored image with additional (assumed) properties or by rendering in conjunction with a scene object. In addition to getting the data, it is also the sensor’s task to determine, from the various attributes of the sensor (e.g. lens parameters, digitizer parameters, etc.), the attributes of the sensor-model (related to its geometric mappings and its uncertainty measurements).

While one might think of sensors producing only image-like data, the mapping concept on which sensors are based is not restricted to physical transducers of energy. Hence, it naturally extends to include the production of spatial objects. This allows us to define a geometric-sensor as something that can have the same "lens" slot as a camera and that uses the same sampling pattern as a camera but that maps a scene with many instances of the class spatial-object into a sensor-model which contains a collection of instances of the class spatial-object. For example, a scene full of polygons might be mapped into a collection of vertices and/or line segments.
The IUE User Interface

The IUE will make extensive use of graphical interaction to support the examination of features and to provide convenient tools for model construction, recognition and etc. These tools will be constructed within a uniform user interface methodology and will allow convenient selection and modification of graphic items.

There are three interfaces for dealing with user interactions with the IUE: The user interface (IUEUI), the IUE graphical user interface (IUEGUI), and the programmer interface.

The Image Understanding Environment User Interface (IUEUI) is intended to provide flexible, simple, and powerful tools for exploring data, algorithms, and systems. In addition to general principles of good interface design, there are several important objectives which are specific to image understanding and developing the IUE:

**Use existing interface standards** The interface should be supported by ongoing and future developments in software environments and graphical user interfaces. The interface components should be built on top of existing and emerging interface packages and interface construction toolkits. This is critical for the long term use of our environment because we can depend on continuous advances in these areas that we want to take advantage of in terms of capabilities and cost (other issues involved with this are discussed in section on graphics software). A good example is the evolving Open GL standard.

**A few, powerful interface classes** The same general principles of object oriented design used in the IUE should be applied to the interface: abstraction over common operations to provide a small number of types of interface objects which can be freely combined by a user. The interface should not involve understanding a large numbers of unrelated things.

**Consistent interaction with other IUE objects** The interface should make it straightforward to manipulate and investigate IUE objects. For example, in displaying a spatial object, a user wants to control all aspects of how the domain and the range are displayed. Another aspect of this is providing intelligent default behavior for interacting with IUE objects, such as setting up appropriate types of browsers for different types of objects.

**Control of the display and presentation** While intelligent defaults and context-based behavior is essential, a user should always be able to override them and have complete control and flexibility.

**Support for sophisticated users** Naive users will want support for running tailored applications with several interaction aids such as menus while experienced users who want programmability and significant compression and abbreviation in specifying actions.

To realize these objectives the interface of the IUE is described in terms of three levels. The Graphics Level is the underlying “machine independent” package for basic display and graphic operations which tell the screen what to do. Examples would be X and Postscript. The Interface Kit Level involves packages for the creation and rapid prototyping of user interfaces and related tools which are built on top of graphics level software. Examples are such things as InterViews, DevGuide, and TAE. This level also includes the tools found in the selected software development environment such as editors and debuggers. It is important that these all be thoughtfully integrated. It should not feel like starting up completely different processes when moving from the debugger and editors for code development to the display and browsing operations of the interface. The 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 for very powerful results.

The IUE User Interface is described in these proceedings [Lawton and Committee, 1993].

**IUE Process Control**

Large grain IU operators typically have complex parameter structures. A significant portion of the time spent in developing IU applications is spent in the exploration of the search space defined by the parameters of the operators. For example, a common type of question that needs to be answered by IU researcher is: "What is the most effective Laplacian radius when performing a Zero Crossing segmentation on aerial images of cities?". A great deal of time and effort is spent in determining appropriate values for parameters of a particular operator in a particular domain. Once these parameter values have been determined, it becomes natural to think of the parameterized operator as a new entity that is different from the unspecialized generic operator. This view leads to the concept of an operator as a Task object that can use inheritance and specialization to represent these parameter structures.

IU research also involves a very large amount of processing and data generation; in this type of environment it becomes important to be able to examine the processing history. The Task class readily supports the maintenance of a processing history through the explicit representation of Task parameters. A Task instance can exist in one of three states depending on the specification of the input and output parameters for the Task: it may either be partially specified, fully specified, or completed. In this way, Task instances describe both the complete input and output specification and the processing status of Tasks. The Task objects thus provide a complete description of the large granularity image understanding processing that has occurred in a user environment.

**Associated Classes**

Another aspect of large grain IU processes is that they are used by the IU researcher as a set of tools within an experimental toolbox. Researchers require a flexible mechanism for control and data chaining that al-
allows them to construct experiments that combine individual Tasks into more complex algorithms. The Task class hierarchy provides this mechanism through the CompoundTask and DataflowGraph object classes. With these classes, the user may chain individual Task objects together, either through programs or through the use of an interactive graphical interface.

DataflowGraphs allow the user to specify data pathways between Task objects. The Tasks in a DataflowGraph can have some subset of their input parameters specified dynamically; the input values of the dynamic parameters are specified by the values of output parameters of other Task objects. Whenever values have been specified for all required input parameters, the Task object is executed. Other DataflowGraph objects, such as DataflowConditional nodes and DataGenerator nodes, provide the control constructs that make the DataflowGraph an effective programming tool. A DataflowGraph may be constructed either in C, LISP, or at the interface level, to form these complex processes. In C or Lisp, this complex Task control can be also implemented through a message passing paradigm in which Task instances are parameterized and controlled through messages (generic function calls) from a controlling program.

Applications
The Tasks that will be supported by the IUE will cover a wide range of algorithms and tools. It will be expected that the set of Tasks that are included with the IUE will expand rapidly as the IUE begins to receive wide use and the research groups using the system begin to contribute their own research tools. The following are examples of TaskGroups specified by the IUE design.

ImageProcessing Tools that map image data to image data.

ImageSegmentation Tools that map image data to symbolic data.

PerceptualOrganization Grouping tools to map symbolic data to symbolic data.

GeometricFitting Tools that fit geometric entities to symbolic data.

ObjectMatching Tools mapping object descriptions to symbolic data.

ModelConstruction Tools for creation and manipulation of object descriptions.

The Future of the IUE
The IUE is planned to be developed in three phases: basic, core and version 1.0. The basic version of the IUE accounts for the elementary classes and methods required to support development of core IU classes and algorithms. The core IUE represents a useful intersection of current practice in IU research. The core will provide representations for the major structures and methods used to implement IU algorithms, such as segmentation, grouping, matching and modeling. Finally, version 1.0 will consist of the core plus selected support for curves and curved surfaces as well as demonstration algorithms which illustrate the use of the core class library.

Currently, the IUE committee is developing a data exchange standard which will be immediately useful for exchanging research results. The data exchange format is based on the IUE class hierarchy and provides an ASCII representation for the construction of class instances. The syntax is Lisp-like and can be easily parsed by LEX/YACC. The initial goal of the data exchange specification is to support technology transfer within the RADIUS project. The format is discussed in more detail in these proceedings [Mundy et al., 1993].

At this writing, the solicitation process is underway to select the IUE integrating contractor. It is expected that the selection will be made in the first half of 1993. The implementation of the basic and core versions of the IUE are expected to take about two years from start of contract with version 1.0 being released at the end of the project.

During these project phases there will be continuous review and consultation to ensure that the IUE meets the requirements of the IU community. Any suggestions or comments concerning design or implementation issues are welcome and may be directed to mundy@crd.ge.com.

References

