The Image Understanding Environment

Robert M. Haralick
Visvanathan Ramesh
Department of Electrical Engineering, FT-10
University of Washington
Seattle WA 98195

Abstract

We describe the Image Understanding Environment (IUE) designed by the IUE committee consisting of members from General Electric, Stanford University, Columbia University, University of Massachusetts, Amerinex AI Inc, Georgia Tech, SRI International, Advanced Decision Systems, University of Southern California and University of Washington. The primary purpose of the IUE is to facilitate exchange of research results within the Image Understanding community. The IUE will serve as a conceptual standard for IU data models and algorithms and will facilitate code sharing and performance evaluation of new techniques. It will also help in tracking progress in algorithm improvements. Object oriented principles are used in our approach to the design of the IUE. The overall specification of IUE objects consists of the specifications of classes and class hierarchies for various IU concepts such as: images, image features, geometric features, curves, surfaces, 3D objects, sensors, etc. This paper discusses the design details of IUE curve objects, the motivation behind the object choices, and the class hierarchies.

1 Introduction

This paper describes the scope of the Image Understanding Environment (IUE) and outlines specific details of the object design for the curves section in the IUE. The paper is organized into two parts. The first part discusses the general scope, requirements, and design principles of the IUE. The second part discusses specific objects that describe curves that are used in image understanding literature.
2 Scope

The primary purpose of the IUE is to facilitate exchange of research results within the IU community. Besides providing a platform for various demos and tools for DARPA applications, the IUE will also serve as a conceptual standard for image understanding data models and algorithms. The availability of standard implementations for basic IU algorithms will facilitate performance evaluation of new techniques and to track progress in algorithm improvements. The IUE is designed to provide an effective programming environment for rapid prototyping. The IUE is not intended to be a real time system although tools will be provided for the simulation of real time applications such as navigation. The IUE will not support special hardware accelerators but a standard image processing interface will be provided. There is no intention to generate a design suitable for embedding in larger systems, although object class components can certainly be used in the construction of new systems.

An important aspect of the development of the IUE is the support of various application scenarios. Some of the areas that have been selected to guide the design of the IUE include: image registration and fusion, object recognition, active vision, stereo, automated image feature extraction, range image sensor development, range data analysis, image segmentation, automated model learning etc.

2.1 General Requirements and Design Principles

The design of the IUE is intended to support several operational requirements. Some of the main requirements include: interactive viewing and manipulation of large images, support for a wide range of image data types, representations for various image sensors, volume as well as boundary representations of 3D models etc.

The central approach to the design of the IUE is the use of object-oriented design principles. Briefly, an object is a data structure with associated operations, or methods, which are naturally defined for the particular structure. The design is specified in terms of an object class hierarchy which represents abstraction relations between classes. For example a "T" junction is a special type of junction. The internal details of an object are hidden as much as possible so that new implementations can be "plugged in" without significantly affecting the rest of the system.

The IUE will run under the UNIX operating system and will have an user-interface built on X-Windows. Both LISP (CLOS) and C (C++) languages will be supported by the IUE. This is done by providing parallel object hierarchies and a mechanism for communicating between the language environments. Extensive Graphical interaction will be used to provide facilities for model construction, recognition, and examination of features. These tools will be constructed within a uniform user interface methodology and will allow convenient selection and modification of graphic items. Major subsystems of the IUE will be isolated by standard interface protocols to help insulate the system development from rapidly changing component designs. For example, the Programmer’s Interface Kernel (PIK) might serve as an interface to image processing libraries. The IUE will also provide support for comparison and testing.
of image understanding algorithms. This support will include database management for test
suites of images and other data as well as results of standard algorithms on the same data.
Statistical tools will be provided to assist in the determination of classification rates and
algorithm reliability.

3 IUE Objects – Categories

The IUE object heirarchy contains the classes in the following categories:

- Images
- Geometric features
- User Interface
- Curves
- Regions and Surfaces
- Solids
- Coordinate systems and transforms
- Constraints and Sensors

In addition to the above categories, there is a category wherein the classes represent the
general IUE object abstraction that is needed for the other categories. Examples of such
classes include: collections, sets, relations, etc. In this paper, we describe the details of the
classes in the Curves section of the IUE. We assume that all of the classes in the curves
section inherit the interface and methods from a generic class called the "spatial-object".
Hence the discussion on curves will often refer to the term “spatial-object”.

4 Curves

In this section we describe the necessary objects that are used in computer vision literature.
We provide a brief description of the objects and the motivation behind our choices. The
objects described includes curve representations as well as data structures, such as points
and pointsets, from which such curve representations may be computed.

4.1 Points and Pointsets

4.1.1 Points

A point is a spatial-object. We classify points as attributed or non-attributed points. A
non-attributed point contains only the coordinates. The attributes are required since many
algorithms use additional information such as edge direction, edge contrast, graytone surface curvature, etc. An attributed point is a simple point with added attributes. Specifically, an attributed point will have an attribute list that specifies what attributes are stored and a pointer to an area of memory containing attribute values. The various attributes allowed may include (but are not limited to) edge direction, edge contrast, curvature, accuracy/tolerance zone for a point, covariance matrix, transformation matrix etc. Algorithms such as the Hough transform method use edge direction as well as the edge strength. Other recent robust techniques that propagate the uncertainty from the low-level stage need the covariance matrix or the tolerance zone information. The transformation matrix information may be needed to just indicate that the algorithms operating on the data have to use the transformed values while leaving the original coordinate values stored intact.

4.1.2 Pointsets

A pointset is a collection of points that may be ordered or unordered. Points in an ordered-pointset are ordered as they are encountered along an discrete-curve-sequence. Ordered pointsets, in general, are the inputs to algorithms that perform curve-segmentation. The classes “ordered-pointset” and “unordered-pointset” are derived from “pointset”. The only difference between these two classes is that the methods that operate on these two classes are different. A pointset may be attributed. A fitted-pointset is a pointset with analytic fit parameters added. Figure 2 illustrates the partial hierarch for pointsets.

4.2 Curves

A curve is a spatial-object. A curve may be a discrete-curve (an ordered sequence of points) or an analytical-curve (analytic representation of a curve). The class “curve” inherits all attributes of “spatial-object” and in addition stores coordinates of the starting point and the ending point of the curve. A discrete-curve is an ordered-pointset. A discrete-curve-chain is a segmentation of the discrete-curve into its continuous subpieces. A typical curve-segmentation algorithm would take a discrete-curve as input and produce a discrete-curve-chain as output. Some algorithms take a discrete-curve-chain as input and produce a new discrete-curve-chain as output. An example of such an algorithm would be the breakpoint optimization algorithm which operates on a segmented discrete-curve and moves the breakpoints to produce an output segmented curve. A discrete-curve inherits all attributes of an “curve” and, in addition, inherits everything from the class “ordered pointset”. A discrete-curve-chain is derived from the class “discrete-curve” and it contains:

- a slot to store the number of sub-pieces in the chain, and
- pointers to the starting and ending points in the segmented result.

Analytical-curve, an analytic representation of a curve can be non-parametric, or parametric. We use the term “parametric” in the sense that the coordinate values of the points in the curve are functions of a parameter \( t \) that takes on values in the interval \((0, 1)\). We use the
term "non-parametric" to include all other representations of the curve. For example, an analytical-curve may be specified: indirectly as the intersection of two surfaces, or directly by giving the equation of the curve. Analytical-curves can be lines, conics, or splines. An analytic-curve may be specified implicitly or explicitly. For example, a line in its standard form is specified by its starting and ending points and a line in its implicit form is specified as the intersection of two planar surfaces. Similarly, the general representation for a conic is as a polynomial. Often algorithms for curve fitting use alternative representations. For example, a circle is specified by its center and radius, and an ellipse may be specified by the lengths of its major and minor axes, the orientation of the major axis and its center point. A parametric-spline is a sequence of parametric polynomial segments. The polynomial segments are delimited by the breakpoints along the curve. From the general object "parametric-spline" we derive specialized instances of splines depending on the spline type, spline dimension, terminating conditions etc.

A composite-curve is an ordered list of the basic curve types, namely: point, line, circle, conic, or spline. If the composite-curve had \( N \) curves then the added constraint is that the \( i \)th curve's endpoint is the same as the \( i+1 \)th curve's start point. A fitted-discrete-curve-chain is a composite-curve. A parametric-spline is also a composite-curve. Figure 1 illustrates the heirarchy for curves.

4.2.1 Objects

We introduce the following objects to describe curves:

- curve (The base class as described before)
- analytic-curve
- parametric-curve
- general-parametric-curve
- implicit-intersection-of-surfaces-curve
- line-start-endpoint
- line-midpoint-direction-length
- implicit-conic
- parametric-conic
- parametric-circle
- parametric-ellipse
- parametric-parabola
A general-parametric-curve is one where the user specifies the number of basis functions for each dimension, the basis function expressions and the solution is given by the fitted coefficient values. For example, a curve $C$ may be defined parametrically as:

$$r(s) = \sum_{m=1}^{M} \alpha_m \phi_m(s)$$  \hspace{1cm} (1)$$

$$c(s) = \sum_{m=1}^{M} \beta_m \phi_m(s)$$ \hspace{1cm} (2)

where $s$ is the arc length, $\phi_1, \ldots, \phi_M$ are the given basis functions, $\alpha_1, \ldots, \alpha_M$ and $\beta_1, \ldots, \beta_M$ are the unknown coefficients. The parametrization may be done in the interval $(0,1)$. The class "parametric-curve" is derived from "analytic-curve" and the class "general-parametric-curve" is derived from "parametric-curve". All parametric classes such as "parametric-circle", "parametric-parabola" and the "parametric-hyperbola" are derived from the class "parametric-conic". An implicit-conic is one which represents a conic by its implicit form. Subdivisions within this class could be done by setting constraints on the coefficients $A, B, C, D, E, F$ of the polynomial. We can also have attributed versions of the above objects so that we can include attributes such as fitting error, estimates of variances of fitted parameters etc.

### 4.3 General Methods

In addition to the methods that operate on a spatial-object there are several methods that are specific to curves. Methods that may operate on a spatial-object include: rotation, translation, scaling operations, boolean operations etc. The list of additional methods include (but is not limited to):

- get-starting-point-of-curve()
- get-ending-point-of-curve()
- length-of-curve()
- closest-point-on-curve()
- distance-from()
- projection-on-line()
- tangent-direction-at-point()
- curvature-at-point()

The first two functions just return the start and end points of a given curve. The function "length-of-curve" returns the total arc-length. The function "closest-point-on-curve" returns the closest point on the curve from a given point.
5 Conclusion

We have described the scope of the Image Understanding Environment, and the different Image Understanding areas for which the object hierarchies have been laid out. We also described specific details of objects in the Curves section of the IUE. The specifications of these objects are continuously evolving as a result of review by the IUE committee and other IU researchers. The ultimate goal of this project is to provide the basic data structures and algorithms that are required to carry out state of the art research in image understanding.

References

Figure 1: Heirarchy for Curves
Figure 2: Hiearchy for Pointsets