INVESTIGATING INVARIANT PATTERN RECOGNITION USING A FLEXIBLE DEVELOPMENT ENVIRONMENT

J. Austin B.N. Brooke

Advanced Computer Architecture Group,
Department of Computer Science,
University of York,
Heslington,
York. YO1 5DD

This paper consists of a short note describing a software development environment used in research involving scene analysis and image processing. It presents an overview of a project using the environment which is investigating a novel form of invariant pattern recognition based on neural network techniques. The paper highlights the need for flexible software tools which permit large scale research in vision in an integrated and uniform manner.

The York Advanced Computer Architecture Group is involved in a number of integrated image processing projects which involve scene analysis, image databases, hierarchical neural network studies. All of these projects have been based around the Scene software environment which has been developed as a central core to coordinate the use of common data formats and function definitions.

The major aim in producing the environment was to:

1) support various levels of user interface

2) provide a common data format

3) provide the flexibility to be applicable to a wide range of projects

4) be open and extensible

As an example of a major project which uses the scene development environment the implementation details of recent work are given on a scene analysis system which incorporates grey-scale N-tuple processing, fovial sampling and a novel form of invariant pattern recognition. Briefly the scene development environment has the following characteristics.

Scene

Scene is a flexible system in which users can produce monolithic systems or "unix style" piped systems consisting of a number of components each providing one operation. They can also use either of the two standalone systems provided. The software has been developed on a Sun 3/50 system running SunOS 3.5. It is written in C and, apart from the display and device specific functions, would be easily portable to other BSD Unix systems.

A common object format is used for all data passed between functions and between the scene system and the operating system. The structure of this object is shown in figure 1.

This structure greatly simplifies parameter passing, error checking, management of objects in memory, loading and storing of objects. The structure contains two basic items: a name of the object for identification and the type of the object (obj_type). The remainder of the header is specific to a particular object type and will contain a pointer to the actual object, e.g. in the case of an image it will contain a structure consisting of image dimensions and a pointer to the actual image pixels.

The system consists of up to 3 layers (shown in figure 2): just the library functions, the library functions plus object management layer, or as a standalone system incorporating a user interface.

AVC 1989 doi:10.5244/C.3.52

```
typedef     struct              GenObj {
      char name[NAMELEN];
      enum                      objtype     obj_type;
      union                     {
            Image               img;
            /* convolve list */
            ConvList            conv_list;
            /* result list from convolution */
            ResList             res_list;
            /* Gaussian filter masks */
            DogMasks            dog_masks;
            /* Map that defines tuple points */
            TupMap              tup_map;
            /* States for tuple rankings */
            TupRank             tup_rank;
            /* State memory for testing */
            TupMem              tup_mem;
            /* points to centre tuple maps */
            TupList             tup_list;
            /* N-tuple testing results */
            TupRes              tup_res;
            /* rotation/scaling invariance templates */
            InvTemp             inv_temp;
            /* transformation codes */
            InvCode             inv_code;
            char                dummy[100];
      } obj;
} GenObj;
```

Figure 1. Object Structure

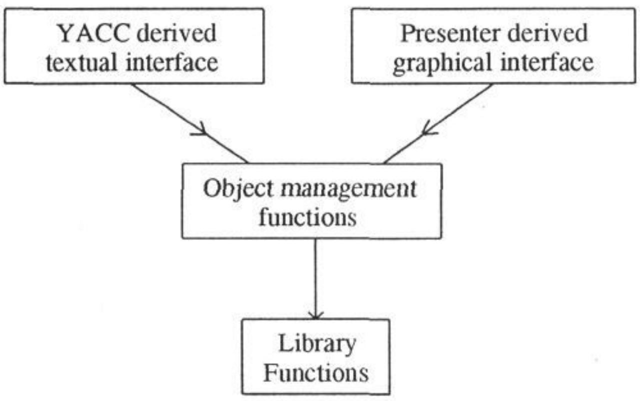
YACC derived textual interface
Presenter derived graphical interface
Object management functions
Library Functions

Figure 2. Scene System Architecture

A scene library exists from which users can import functions into their own programs. Many standard image processing functions and support functions are supplied, e.g. loading and storing of images, rotation, histograming, thresholding, extracting parts of images, pixel access, blitting images onto other images, various halftoning methods, convolutions using built in masks such as Sobel or Roberts operators or using user supplied masks.

Users can load object management functions as well as image processing functions into their application programs. This provides a method of storing and managing objects in memory. The underlying method is transparent to the user. At present objects are stored in a singly linked list but this method may be changed, if necessary, transparently to the user. Objects are retrieved by name because this is the most suitable for using with the standalone command interpreter system, but additions can easily be made to find objects by some other method such as numerical identifier.

Two versions of the standalone system are available. One is a command line based environment which uses a textual command interpreter. The advantage of using this method is that it is not tied to any particular hardware or availability of devices. This means that if speed is important it can be run on a file server which may be faster than a workstation. The other is a graphical interface based on Presenter[1] . This is a tool for constructing graphical interfaces which contains only one primitive: the region. An illustration of the interface as it is presented to the user is shown in figure 3. From this objects, such as buttons or sliders which may be primitives in other graphical interface systems, are constructed. An interface has been constructed with it which, like the command interpreter based system, incorporates the object management layer. Each operation has it own region which when mouse activated brings a box onto the screen with spaces for the required parameters for the operation. These can be filled in by hand or taken from the list of objects in memory. When an operation box has been fully composed the operation is performed.

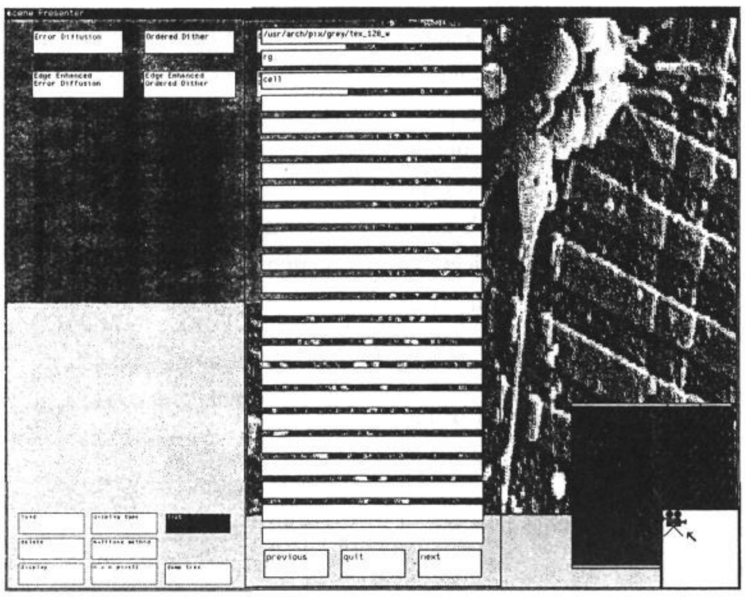
Error Diffusion
Ordered Dither
Edge Enhanced Error Diffusion
Edge Enhanced Ordered Dither
/usr/arch/pix/grey/tex_128_w
rg
cell
delete
display
previous
quit
next

Figure 3.
Presenter graphical interface for the scene environment

Of particular interest was the ability to support distributed tools. At York an image server was available as a central resource which permitted local storage and capture of images and access to dedicated image processing hardware. To permit access to this resource, network protocols were incorporated into the scene system. This allows remote execution of image capture and processing operations.

Details of the full scene analysis system are described in[2] but the implementation aspects of the following three sub-systems of the complete system are given in the context of the scene system:

- Distributed invariant classification system
- Fovial sampling input subwindow system
- Grey-scale N-tuple processing

Of particular interest is how the development environment has been used to produce highly efficient implementations of these subsystems. The following describes briefly the problems tackled.

Distributed invariant classification system

The distributed invariant classification system allows the production of a rotation and scale invariant representation of an object for high speed matching with a number of single view templates. The system incorporates basic functions of the ADAM associative memory[3] , which is based upon neural network techniques. The scene system has been used to develop a flexible and high speed implementation of this method which is allowing a thorough analysis of the method's performance. The method uses a number of tables which are indexed in relation to the output of the fovial input subsystem. This produces the invariant representation that is matched against the stored template. The scene system object management layer is again used to manage these tables and templates allowing storage retrieval and display of the items in a simple and effective manner.

Fovial sampling input subwindow system

The scene analysis system incorporates a polar sampling input windowing system which exploits fovial sampling. To support such a scheme a front end processor has been developed that uses small adaptable N tuple recognisers as edge detectors / circular resolution reducers.

Close to the focus point the circular sampling windows are small, further from the point of focus the circles are larger. To support such a system many tables are needed to specify: the samples that make up each window, the position of these windows and the mapping of tuples within each window. Each window is processed by the grey scale N tuple process (below) to produce an output which must also be stored. All of these tables are stored as objects in the object management layer of the scene system. The whole subsystem is highly adaptable and may be easily reconfigured to deal with different sampling structures. The scene system has allowed the effective development of the subsystem and has permitted flexible storage and retrieval of results for assessment purposes.

Grey-scale N tuple processing

N tuple processing takes a tuple of pixel values from an image and assigns a state to it. For a binary image the binary code itself would be suffices as a state number, but for an 8-bit grey scale image this method would allow too many possible states, e.g. for a tuple size of 4 there would be 256^4 possible states. A method is described by Austin[4] which reduces the number of states by ordering the pixel values into a user chosen number of ranks. This produces a more manageable number of states. e.g. a 4-tuple divided into 3 ranks will have 51 states. The fundamental problem with this method is speed. Each tuple must be sorted, ranked and its state calculated. We will describe a considerably faster method in which each tuple is reduced to an integer code which is used to index a pre-compiled table to derive a state. No explicit sorting, or calculation of state is needed. It will be shown that there are 2 stages to preparing the state tables. Firstly all valid ordering codes are generated and reduced to rankings. A state is assigned to each ranking and hence each ordering code can be assigned a state. A table is then produced which directly correlates ordering codes and states. This is a sparse table because most ordering codes are invalid. The table is a scene object and can be used with the standalone systems and with customised scene systems.

Summary

It has been shown how current work at York is aimed at providing an integrated support environment for research in computer vision. The major structure of the system has been developed which integrate a graphics environment (presenter) with a flexible command sub-system (scene). This, coupled with the common data formats and distributed processing ability, is providing an open and extensible system into which other image processing libraries may be incorporated. It has been illustrated how the system is being developed along side a project, providing a test bed for the implementation of the environment, and also an essential research tool for the project.

1. R. K. Took, ''The Presenter - A Formal Design for an Autonomous Display Manager'', pp. 151-169 in *Software Engineering Environments*, ed. Ian Sommerville, Peter Peregrinus (1986).
2. J Austin, ''High Speed Invariant Pattern Recognition using Adaptive Neural Networks'', in *Proc. of Conference on Image processing and its applications, Warwick, UK.* (July 1989). To be presented
3. J.Austin, ''ADAM: A Distributed Associative Memory For Scene Analysis'', pp. IV-285 in *Proceedings of First International Conference on Neural Networks*, ed. M.Caudill, C.Butler, IEEE, San Diego (June, 1987).
4. J. Austin, ''Grey Scale N tuple Processing'', pp. 110-120 in *Pattern Recognition : 4th International Conference, Cambridge, UK*, ed. Josef Kittler, Springer-Verlag, Berlin (1988).