A PYRAMID BASED ENVIRONMENT FOR THE DEVELOPMENT OF COMPUTER VISION APPLICATIONS

G. Sandini, M. Tistarelli,* & D. Vernon,#*

* University of Genoa
Department of Communication, Computer and Systems Science
Via Opera Pia 11A,
I–16145 Genoa

University of Dublin, Trinity College,
Department of Computer Science,
Dublin, Ireland.

Abstract

One of the major motivations for the implementation of computer vision is the need to develop flexible systems which, through sensory input, can adapt to partially unknown environments or unexpected events. In large scale applications, current approaches require time–consuming tailoring of existing algorithms, exploiting application–dependent constraints, followed eventually by the implementation of specific hardware and software modules. Small and medium scale applications require hardware and software tools, structured as a "development system ", to decrease lead–time to implementation by allowing a user to test existing algorithms, implement new application–specific ones, and to experiment with all facets of the development system in the application environment. The conventional solution to the development system problem is to use "libraries" of sub–routines implementing various algorithms. Such an approach is inherently inadequate as it does not facilitate the integration of visual information derived from several modalities or cues (e.g. shading, stereo, motion and texture) and forces the user to configure the vision experiment by linking various quasi–independent software modules.

This paper describes a computer vision environment which redresses this situation, providing a system which treats the image data (and its organization) equally with the processing capabilities.

Several spatial frequency channels are facilitated using a pyramidal image structure and several levels of image representation are exploited to make the different types of information extracted from the many vision facilities of the system. Furthermore, utility features e.g. such as image system archival and run–time configuration, prototyping, and an interpretive control language, are provided to encourage both interactive experimentation and configuration of target applications.

Introduction

One of the major motivations for the implementation of computer vision applications is the need to develop flexible systems which through sensory input, can adapt to partially unknown environments or unexpected events. The development of such systems is possible, with current technology, only through the exploitation of all the application's constraints. This approach requires a time consuming tailoring of existing algorithms followed, eventually, by the implementation of specific hardware and software modules. Using this methodology, the development of such sensory–based applications can only be amorized by large scale applications.

In the case of small or medium scale applications or even to reduce the development costs of the large scale ones, it is necessary to develop hardware and software tools, structured as a "development system", that can be used to test existing algorithms, to implement special processing steps, and, last but not least, to experiment with the specific application and to evaluate the overall performance.

The usual attitude toward the solution of this problem is
use "libraries" of sub–routines implementing various alg
rithms. This approach, in spite of its wide use, is not satisfa
tory as it does not help the integration of multi–modal visu
data, i.e. the visual information arising from different visu
cues. Furthermore, the execution of a "vision experiment"
left completely to the programmer who has to "link" speci
software modules to test a specific application. However, it
our belief that an application development environment must n
only provide the possibility of *processing data* but, more impo
tantly, the possibility of *integrating this data.*

This paper describes a computer vision environment whi
redresses the situation, providing a system whioh treats t
image data, and its organization, equally with the processir
This is accomplished by means of a *Virtual Image Structu*
which may contain multiple component image structures, ea
one characterized (identified) by photometric information, su
as the time stamp (in, say, a motion sequence), a left/rig
specification (in the case of a stereo pair), the camera paramete
(focal length, attitude of focal axis). Each image structure ma
be organised into different spatial frequency channels using
pyramidal structure [1] with image resolution varying from 10
x 1024 to 64 x 64 pixels (Refer to figure 1). The Virtual Ima
Structure, then, is a hierarchy of these pyramidal image stru
tures.

Coupled with this structure is an interactive environme
which facilitates experimentation with many facets of ear
vision and which allows the user to quickly prototype and te
application–specific algorithms. The complete system, referr
to as *VIS*, for Virtual Image System, is described in the follov
ing section. Subsequent sections proceed to detail the interr
data representations and to explain how they facilitate integratio
of low–level visual information; several examples of the use c
VIS are provided. Bearing in mind that low–level visi
is not an end in itself, the final section concludes the paper l
discussing how the system communicates with other system
e.g. A.I. workstations, and indeed with other instantiations
itself on a multi–processor network.

An Overview of the Virtual Image System

The most important point to make is that *VIS* is a comple
vision software system environment providing key features
allow a user to build systems of image structures, to archiv
them (or part of them), to dearchive them, and to process ar
analysis them. The main emphasis of the processing and analy
ing is "low–level" vision i.e. Marr–Hildreth (Laplacian c
Gaussian) edge detection [2], contour and region analysis [3], th
generation of the "raw primal sketch" [4], and the inference o
3–D structure using stereopsis and motion [5–11]. The uniqu
feature of *VIS* is the manner in which the results of this pro
cessing are represented in a Virtual Image Structure of pyram
dal image systems. We will discuss *VIS* under three heading
each section highlighting a distinctive aspect of the system.

Acknowledgements: The research described in this paper is supported by the ESPRIT project P419: Image and Movement Understanding.

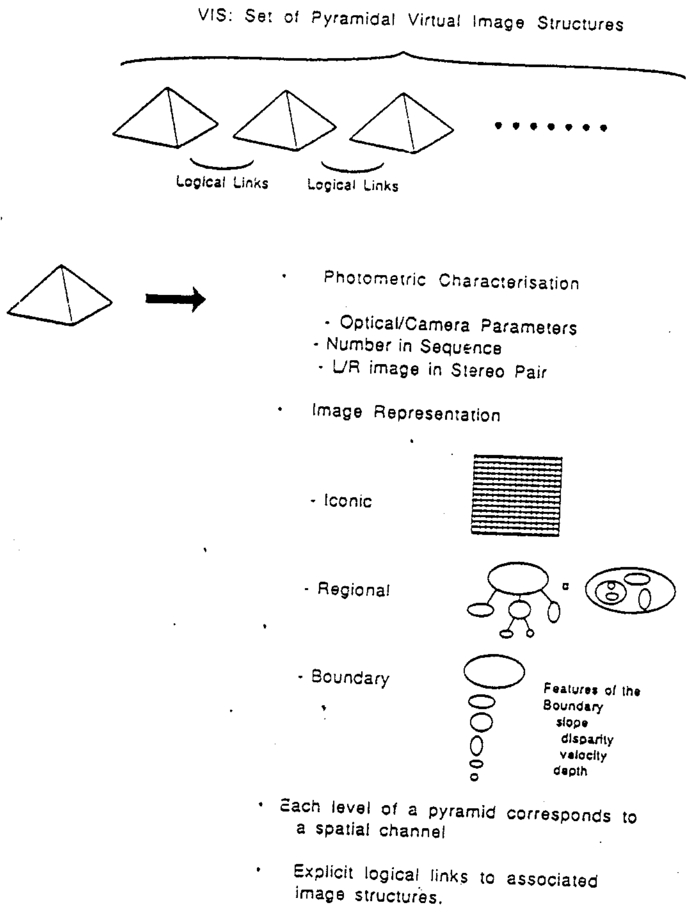
VIS: Set of Pyramidal Virtual Image Structures
Logical Links
Logical Links
Photometric Characterisation
- Optical/Camera Parameters
- Number in Sequence
- L/R image in Stereo Pair
Image Representation
- Iconic
- Regional
- Boundary
Features of the Boundary
slope
disparity
velocity
depth
Each level of a pyramid corresponds to a spatial channel
Explicit logical links to associated image structures.

Figure 1: Schematic representation of Virtual Image Structure

The Software Environment

Vis is NOT simply a library of image processing primitives: it is a complete development environment. The acronym *VIS* derives from the phrase Virtual Image System, so called as it allows user to specify interactively the number, the resolution, the type, and the organisation of the images at run–time; the number of images you can add is only constrained by the availability of memory. Since the atomic unit of data in *VIS* is effectively the pyramid, i.e. a set of hierarchically–related images, images are "added" by inserting them at specified levels in the pyramid. Currently, pyramids can have up to ten levels corresponding to image resolutions of 1024 x 1024 pixels to 64 x 64 pixels respectively. The levels within these pyramids, then, correspond to the different spatial frequency channels utilised during processing.

Each pyramid (or image) is identified with one of three classes of representation: iconic, regional, and boundary. Within each classification there are further sub–categories so that each pyramid (or image) is tagged with an unambiguous and unique data type.

Iconic types include:

1) Framestore: the input from a TV camera or a "virtual framestore" (i.e. a file on disk).
- Intensity: representing a grey–level image (nominally using 8 bits per pixel).
- Stereo–pair: a pair of intensity images acquired by a stereo set–up.
- Motion Sequence: a sequence of images acquired by a single camera.
- Stereo Motion Sequence: a sequence of stereo pairs
- Convolution: the output of a convolution of an intensity image with a Laplacian of Gaussian image (this image can be a "single" image, a "stereo–pair", or a motion sequence or a stereo sequence.
- Zero–crossing: a bit–mapped output of a zero–crossir extraction algorithm

The regional type represents explicitly the hierachical ne ing of regions of uniform sign in the convolution image: this i accomplished via a tree of distinct region descriptions.

The *boundary types* are coded (internally) as chain–code Each boundary representation stores a different measure cor puted at the zero–crossings of Laplacian of Gaussian filtere images. All these representations are spatially registered wi the "zero–crossing boundary representation". Among the bou dary type are:

- Zero–crossing position: the chain of code of the contours.
- Zero–crossing slope: the edge slope for each bounda point in the zero–crossing position.
- Zero–crossing orientation: the local edge orientation.
- Zero–crossing disparity: the disparity of a stereo pair.
- Zero–crossing velocity: the amplitude and direction velocity vectors at each point on the zero–crossing co tours of an image in a motion sequence.

This typing of images means that image processing is accon plished in a particularly simple manner: by *TRANSFERRING* image from one pyramid to another pyramid. All processing effected implicitly. In addition, the system allows extensive wi dowing on both source and destination pyramids so that, in add tion to being transformed, the destination image may be a scale and translated version of the source window.

Once a system has been configured, and, possibly after th user has done some processing and analysis, (s)he may wish suspend the session and resume work later. This can be accon plished by saving the system status and later restoring it. No that the complete system does not have to be saved; at the user discretion, selected parts of it can be saved. Similarly, whe restoring a previously–saved system status, the user can eith overwrite the current configuration or add the new configuratic to the system.

As an example of this organisation of typed image pyram ids, figure 2 depicts a typical configuration used in experimentin with sequences of stereo–pair images.

The User Interface

Users can communicate with *VIS* in two ways: via th menu system or using the dedicated programming languag VISICL (*VIS* Interpretive Control Language). The menu syster facilitates simple interactive use of *VIS*. Each menu comprise up to ten numeric menu options and seven other options whic are invoked using a single alphabetic mnemonic. The numeri options are peculiar to the current menu and will typicall change from menu to menu; the alphabetic options are availabl from every menu. Quite often, a user will need to elect number of options many times, e.g. to configure a system or t accomplish a series of image transfers. *VIS* has a LEAR option and a REPLAY option which logs the menu selections i a file and replays them from that file, respectively.

The second way to use *VIS* is via its own programmin language: VISICL. VISICL is invoked by depressing the menu option from any menu. This brings the user into a simpl BASIC–like editor/interpreter environment from which the use can read VISICL files from disk, save them and run them.

Most of the *VIS* menu options are available using VISICL statements, in addition to the set of *VIS*–related primitives. VISICL also provides the usual structured programming con structs such as IF, WHILE, REPEAT, and procedures.

Finally, it is worth noting that the VISICL commands RUN and CHECK can (optionally) have a range of lines associated with them to delimit the segment of the program which are executed or checked for syntax errors.

In addition, it supports six different types of framestore device: Imaging Technology PCVISION, and FG100 series, Datacube Max Video Series, VDS Eidobrain 7001 systems, VICOM, and X–windows. *VIS* is currently being ported to an Apple MAC II environment supporting a Data Translation frame-grabber. Although this does not necessarily qualify *VIS* as a standard common–tool environment, it does at least facilitate its use on many of the more common architectures.

Internal Representations and Data–Structures

In section 2, we identified one of the major strengths of *VIS*: the ability to configure Virtual Image Structures of pyramid images, in which the internal representation is the one most appropriate to the information being made explicit, and the inter–pyramid links and descriptions which facilitate the expression of image inter–dependencies. From the point of view of the user, these issues are treated at a conceptual level and it is not necessary to be aware of how these representations are implemented in order to use the system effectively. However, if one wishes to utilise the prototyping facility and integrate one's own algorithms one must address these issues of representation and data–structures. This is particularly true for algorithms which will exercise the image interdependencies (thereby gleaning the maximum benefit of the visual integration).

Although this has been dealt with thoroughly elsewhere [12], it is useful to summarize the data–structures here. However, it should be noted well that anybody requiring to manipulate these data–structure can exploit the existing set of data–structure handling primitives.

Recall that there are three key features of *VIS*: first, the disparate data representations for individual images of different generic types (iconic, regional, and boundary); secondly, the pyramidal organisation of images and the collection of pyramids into Virtual Image Structures; and, thirdly, the formation of inter–pyramid links to make the interdependency of images explicit and to represent common statistical information pertaining to properties of each image type. We shall outline the data–structures for each of these three features in turn and in bottom–up fashion.

Images

Each image comprises an image descriptor and a structure which is appropriate to the type of image being represented (iconic, regional, or boundary). The image descriptor comprises:

- The image type
- an alpha–numeric description
- the image size
- the pyramid level number
- the currently defined window in the image
- a link to the image data–structure, per se
- an optional link to a framestore descriptor (if the image type is a
  framestore)
- a link to the pyramid of which it is a component

Iconic images are represented simply as an array of bytes. However, the normal 2–D array is not used for reasons of portability and efficiency. Instead, an iconic image is defined as a set of 1–D vectors of bytes; each vector is "addressed" by an explicit pointer which is itself stored in a 1–D array of bytes. Note, however, that persons developing prototype algorithms may still reference an element in the image using the familiar double subscript, e.g. image[i][j] (in the C programming language). In this case, the image is accessed by indirection, not by sub–script evaluation and is, hence, more efficient.

Regional images are rather more complex, comprising two distinct components: a region–crossing image and a tree of region descriptors. The region– crossing image is, in fact, an iconic representation in which each distinct region of uniform sign in the Laplacian of Gaussian convolution image is labelled with a unique identifier. The topology of the hierarchical nesting of these regions is represented by a tree of descriptors for each individual region. The two components of the regional image are linked in that the region–crossing value is used as an index into a look–up table of links pointing to the corresponding region descriptor in the tree. Thus, one can directly access a region descriptor using the region label stored in the iconic region representation. It is worth noting that each region descriptor (in the tree) stores useful information regarding the morphology of the region, e.g. area, centroid, and energy.

Boundary–type images represent the zero–crossing contours of a convolution image (which are exactly the boundaries of the regions in the region image) as a series of lists. Each list element contains information peculiar to the category of image within the general boundary classification (contour, slope, disparity, etc.). The contour image list elements are the Freeman chain code direction [13, 14] required to generate the next point on the contour; each list represents a single contour in the image. Similarly, the slope, orientation, disparity, velocity, and depth images contain information regarding edge strength, edge direction, stereo disparity, optical velocity, and range, respectively.

The pyramidal organisation is achieved by grouping together the constituent images and linking them to a pyramid descriptor. This descriptor details:

- the pyramid number
- an alpha–numeric description of the pyramid
- links to its component levels (i.e. to the image descriptors)

The complete Virtual Image Structure comprises a list of these pyramid descriptors.

Integration of Images

Finally, the integration of images is achieved through the use of explicit linkages between pyramids on two distinct bases. First, on a snap–shot basis where we link images derived from a single scene, each contour in the resultant processed boundary type images is linked to a contour descriptor. Residing in this descriptor is statistical information on the distribution of all pertinent boundary–based information (mean and variance of slope, orientation, depth, disparity, velocity). Second, on an image sequence basis where we link images derived from a motion sequence, each element of each contour in a velocity image contains, in addition to the information on the amplitude and direction of the velocity vector, the "coordinates" of the corresponding point in the subsequent image in the sequence. These coordinates are expressed in terms of the pyramid number, pyramid level, contour number, and position within the contour. It is this schema of sequence connection that facilitates the tuning of image tracking across an extended motion sequence.

Examples of Image Processing

In *VIS*, the detection of intensity discontinuities will normally be achieved by transferring an intensity image (figure 3) to a convolution image (figure 4), and thence to a region image (figure 5), and from the region image to a contour image (figure 6), simultaneously constructing the slope image (figure 7) and orientation image (figure 8).

As we mentioned previously, a facility is incorporated in *VIS* to analyze the statistics of contour properties, such as slope, disparity, orientation, curvature. We can now select/deselect contours on the basis of these statistics: figure 9 illustrates those contours exhibiting a mean slope greater than the global mean slope evaluated over all contours in the image. We can apply the same approach in dealing with regional images: figure 10 illustrates those regions exhibiting a mean energy greater than the global mean energy of all the regions in the image.

Having extracted the intensity discontinuities, subsequent processing stages include their description using higher–level primitives such as line segments and the computation of depth using, e.g. stereopsis and motion. The first stage is accomplished

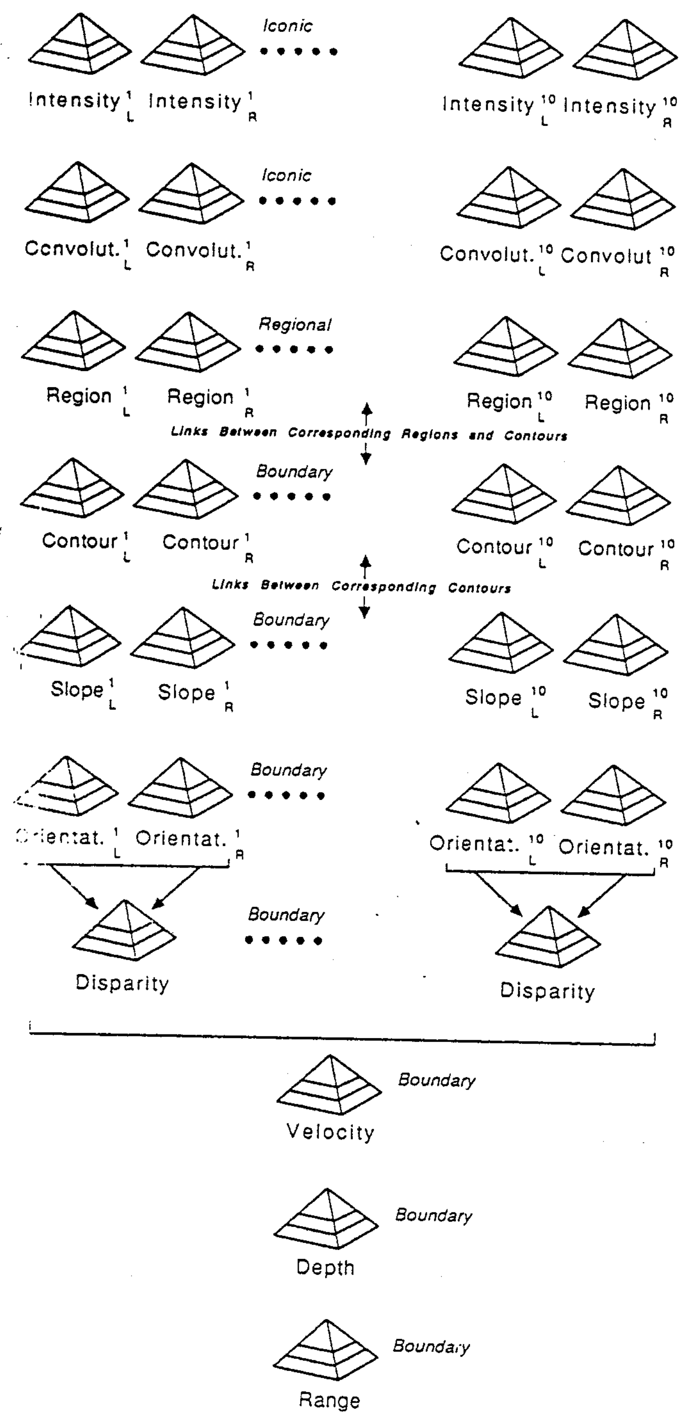

Figure 2: Example VIS for a motion experiment comprising 10 stereo image pairs

The Computer Vision Facilities

Much of low−level computer vision, and in turn high−level vision, depends on one initial key step: the detection of intensity discontinuities in the image. Normally, this step or processing is referred to as edge detection, though it should also be noted that edge detection, per se, is a more embracing term which should also address the physical causes of the intensity discontinuities (e.g. shadows, object boundaries, and surface discontinuities).

VIS uses one of the most popular and advanced techniques for detection of intensity discontinuities: the Marr−Hildreth theory of edge detection which utilizes Laplacian of Gaussian filters. This theory is based on two premises that discontinuities in image intensity give rise to zero−crossings (i.e. points where the image function change sign when differentiated twice) and that the detail of these zero−crossings can be well controlled by first smoothing the image with a Gaussian function. As we have seen, *VIS* provides for a complete set of image types corresponding to different levels and stages of low−level processing. Thus, the identification of the zero−crossings (or edges) using *VIS* requires just two simple actions (i) transfer an intensity image to a convolution image and (ii) transfer a convolution image to a zero−crossing image. Of course, we could transfer an intensity directly to the zero−crossing image but, as we shall see, it is sometimes useful to retain the convolution representation.

Since each of these different image types are inter−related, *VIS* also links them together and, significantly, allows the user to exploit these linkages. In support of these images, *VIS* provides menu options (and VISICL primitives) to compute various statistics of the contour and region properties and to select and deselect contours and regions on the basis of those statistics. Contours can be wholly selected or deselected or the selection process can be applied on a local basis to each individual point on the contour.

The fundamental motivation behind *VIS* is to provide the user with as much information as possible (in a coherent intelligible form) and to allow him/her to manipulate the images on the basis of this information. Although the extraction of intensity discontinuities in an image is fundamentally important, it is just a first step. Subsequent stages include the computation of depth (i.e. the recovery of a 3−D structure) of the scene using, e.g., stereopsis or motion and the grouping of the image contour and their description using higher−level primitives such as line segments.

Finally, it cannot be argued that *VIS* itself represents the end of the road in low−level computer vision; on the contrary, it is just the start but provides a well−founded basis from which to add further low−level modules (e.g. texture, shading) and medium−level modules (e.g. shape). Ultimately, it can be viewed as the low−level front−end sub−section of a complete vision system. To allow users to develop and experiment with their own algorithms, exploiting the rich representations of *VIS* and the VIS−environment, the *VIS* package provides a menu of ten *prototype algorithms*. All that is required of the user is to write his/her algorithm−specific code, compile it, and link it as one of the prototype options.

Host Environments

The computer vision system being described in this paper is intended to be an alternative to conventional library−based development systems. To be accepted as such, it must be available on many common host computers and it must support a representative selection of the more popular framestore devices. The portability that is implicit in these requirements exists largely because *VIS* has grown out of an international European collaborative research program; it was originally designed to service five members of the project consortium each member utilising significantly different imaging equipment. At present *VIS* runs under different types of system based on Unix (and derivatives), VMS, DOS, and transputers.

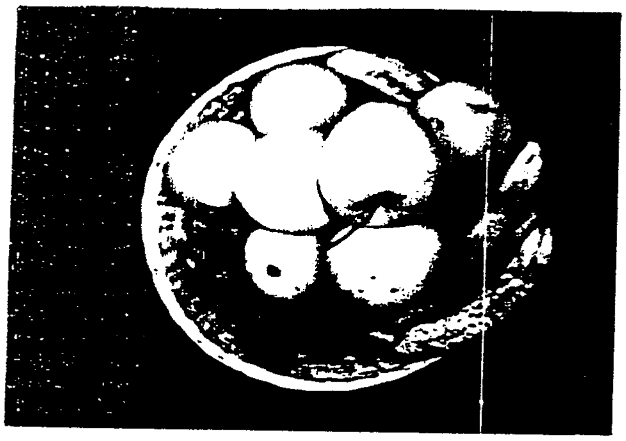
Figure 3: Intensity Image

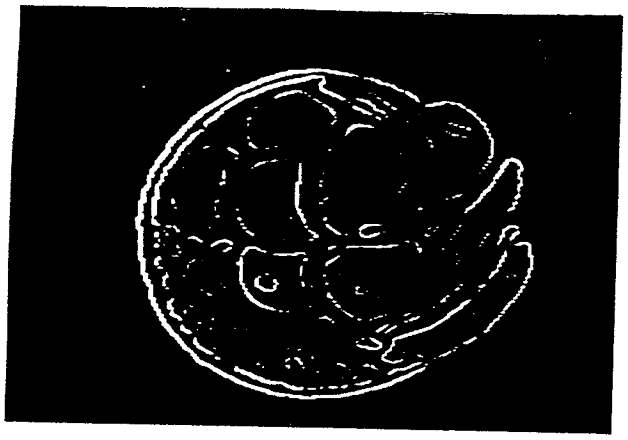
Figure 7: Slope Image

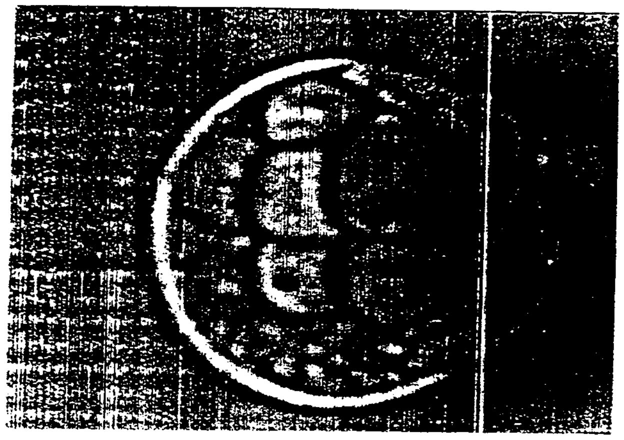
Figure 4: Convolution Image

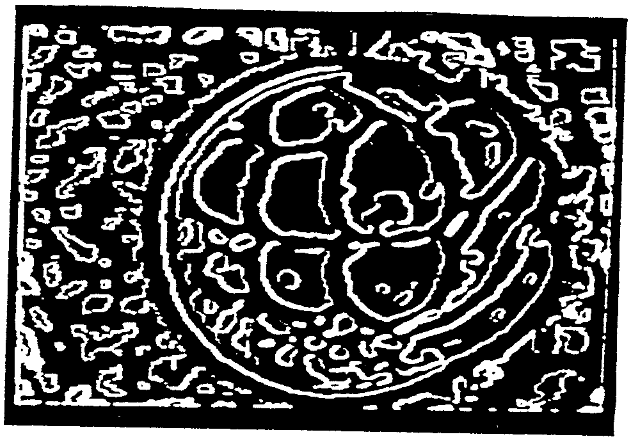
Figure 8: Orientation Image

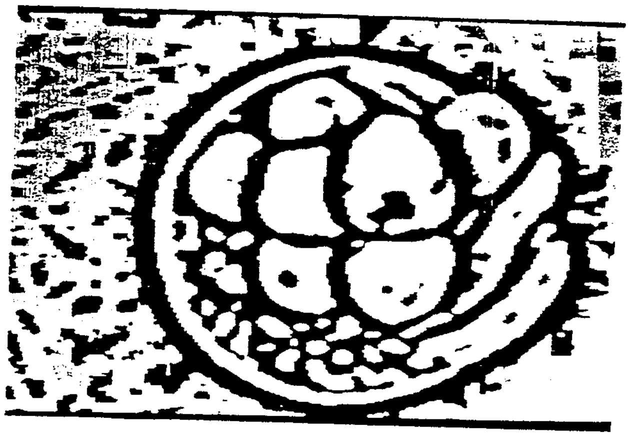
Figure 5: Region Image

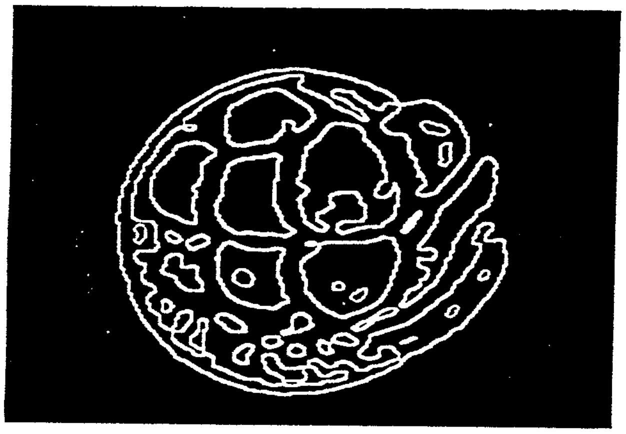
Figure 9: Selected Contours

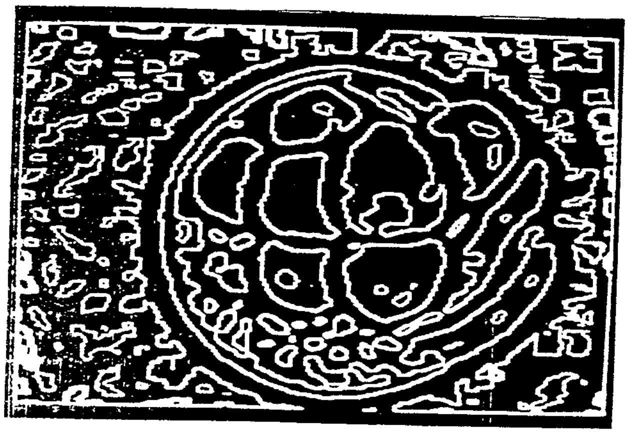
Figure 6: Contour Image

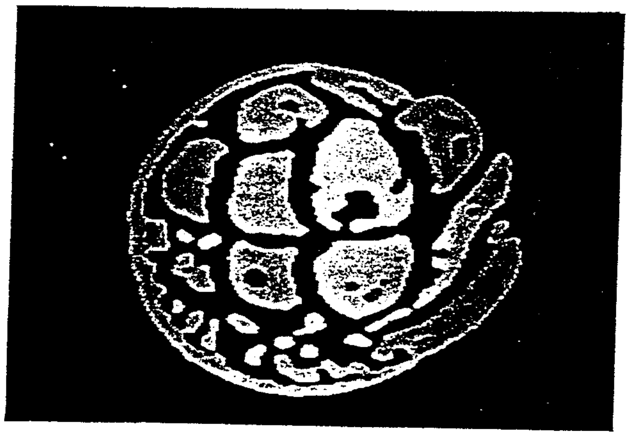
Figure 10: Selected Regions

by constructing Marr's Raw Primal Sketch, a representation which is based on a grouping of zero-crossing points into small line segments called edges (see figure 11). These edges are, in turn, collected together to form lines (which can be either straight or curved). The beginning and end of these lines are marked by terminations (see figure 12). Instances of local parallelism are marked as bars (see figure 13). We mentioned that the second phase of the complete edge detection process was the determination of the physical cause of the intensity discontinuity. While *VIS* does not yet do this, it does allow the user to ascertain whether or not the edge is an artifact or a real physical edge using Marr's "spatial coincidence assumption". The central idea here is that if an edge is caused by a real physical stimulus (e.g. a shadow or an object boundary) it will be present in the zero-crossing images which have been generated by two different Laplacian of Gaussian functions (i.e. two different variances). Furthermore, the zero-crossing will tend to have comparable strength and direction. *VIS* provides an option to identify spatially- coincident contours as a pre-cursor to the generation of the raw primal sketch.

The computation of stereo disparity is illustrated in figure 14 through 17 which depict two stereo images, the disparity computed in local patches, and the final boundary based disparity image, respectively.

In a similar vein, figure 18 through 20 illustrate four snapshots in a camera motion sequence as the camera moves around the object, the resultant velocity vectors, and a "side elevation" of the resultant depth image (in which depth is now proportional to the distance from the right hand side of the image).

For the Future

Low-level vision is not, of course, an end in itself. The information generated at this stage must necessarily feed the higher level cognitive modelling sub-systems and, indeed, these cognitive systems must control the low-level processing.

At the current stage of development, *VIS* facilitates this two-way flow of control and information through the use of its interpretive control language: VISICL. Cognitive modelling systems, typically running on an alien computer generate appropriate VISICL programs, transfer them to the *VIS* host where they are automatically executed, and the resultant information is them transferred back to the alien cognitive modelling system. At present, the information which is transferred in a modified version of Marr's Raw Primal Sketch. Bar and blob primitives are not, as yet, communicated to the alien system. The edge segments in this instance are identified by an initium and a terminus; two points specified, not just in 2-D, but by 3-D coordinates. Since these edge segments are generated from the zero-crossing contours, the third dimension can be directly extracted from the depth map derived from the stereo and motion ranging process.

The system described in this paper was introduced as an advanced vision development tool. We are also attempting to see if it can be adopted as a (near) real-time target environment. Our motivation is that strategic development of industrial vision requires the implementation of emerging 3-D techniques, such as are described in this paper; dedicated hardware, in the guise of custom-designed IC devices or array processors, is not likely to be flexible enough to cater for the varying requirements of these disparate techniques. It is our contention that fast processing can be achieved through the use of loosely-coupled concurrent modules in the form of re-configurable transputer arrays. To that end, we are currently implementing a hierarchical system of transputer nodes, where each node runs *VIS*. Inter-node communication and parallelism is achieved by allowing each node to construct vision programs (in VISICL) and request their execution on a neighbouring node. Results and processed images can be returned to the requesting node either as primal sketch data, as above, or as remotely-generated Virtual Image Structures.

In summary, *VIS* is a tool which places a sophisticated low-level computer vision environment at the disposal of a vision scientist and facilitates integration of low-level visual cues. *VIS* is also a tool which allows the integration of the complete early vision system with high-level cognitive modelling systems. Finally, it is noteworthy that *VIS* is a portable tool and presently supports six host environments with drivers for six different types of framestore devices A commercial version of *VIS*, now available, supports a sub-set of these capabilities and features[1]

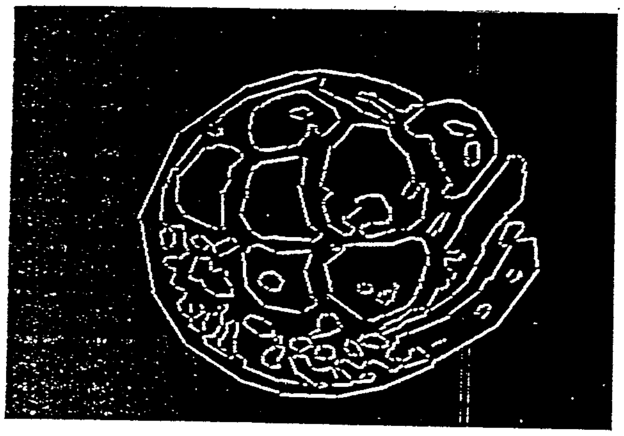
Figure 11: Raw Primal Sketch edges

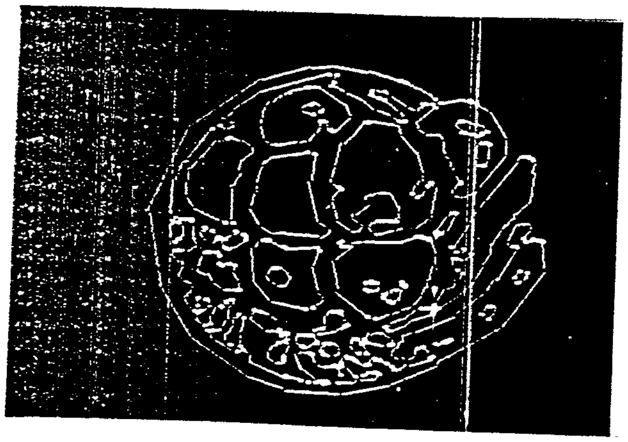
Figure 12: Raw Primal Sketch edges and terminations

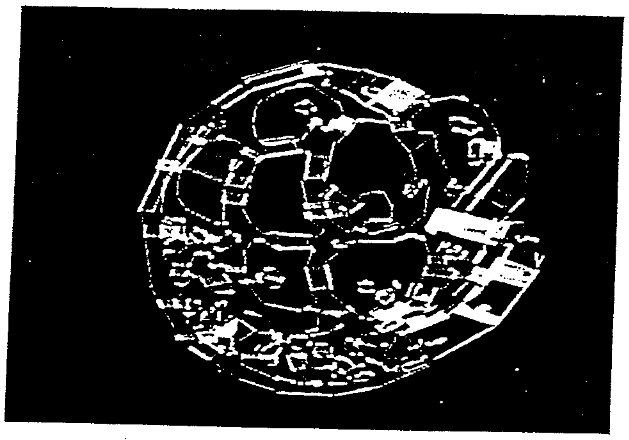
Figure 13: Raw Primal Sketch bars

[1] Aitek Srl - Piazza S. Maria in Via Lata 9/3 - I-16145 Genova Italy. Eidetics Ltd - Kill Abbey House, Blackrock - Dublin Ireland.

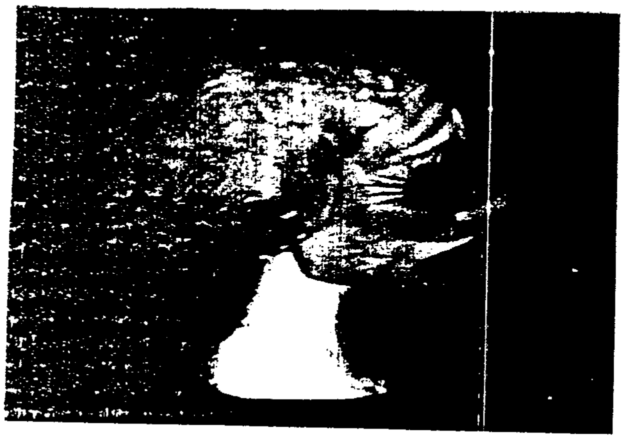

Figure 14: Left Intensity Image in Stereo Pair

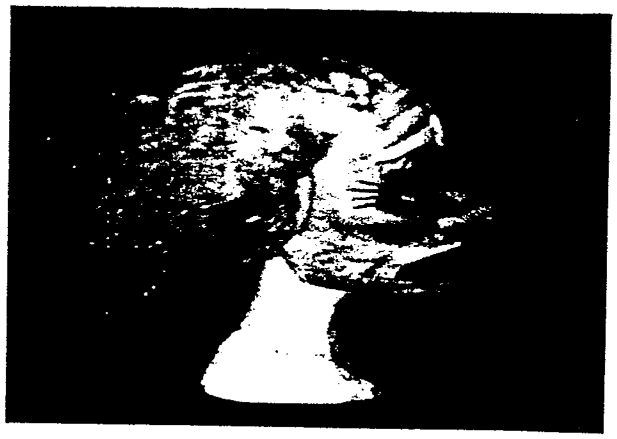

Figure 15: Right Intensity Image in Stereo Pair

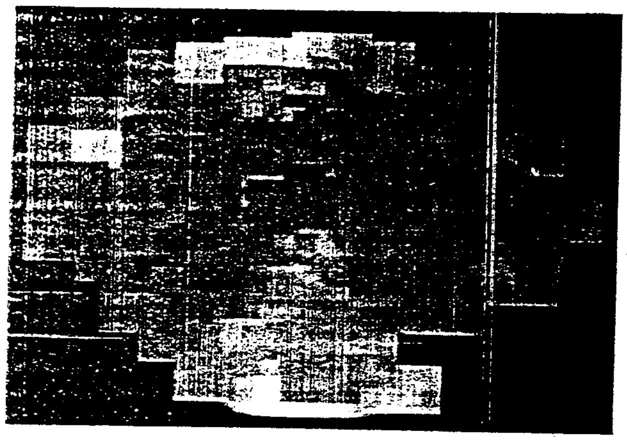

Figure 16: Stereo Disparity (represented as local patches)

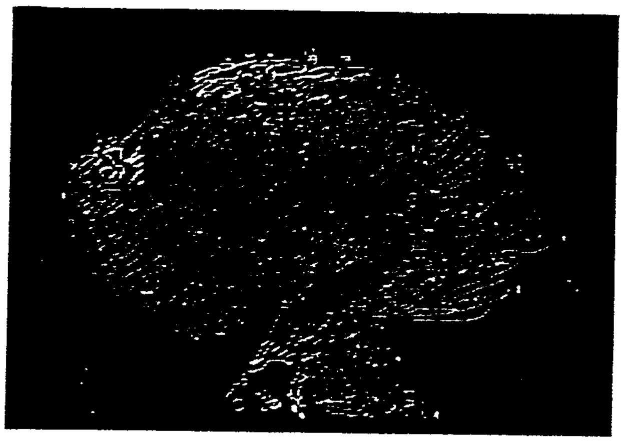

Figure 17: Stereo Disparity (Boundary image)

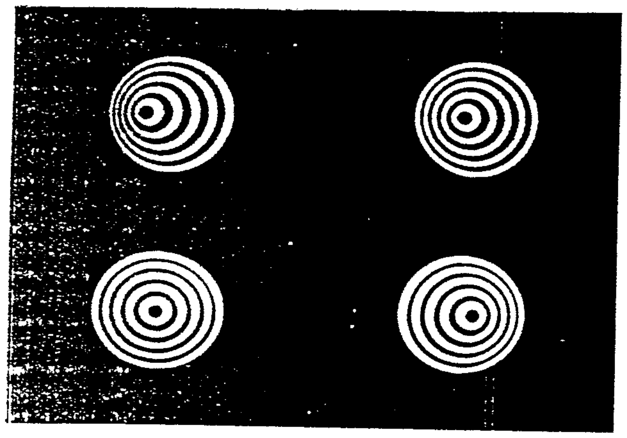

Figure 18: Motion Sequence

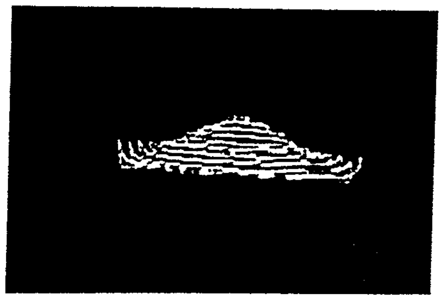

Figure 20: Depth Image (side elevation of contours)

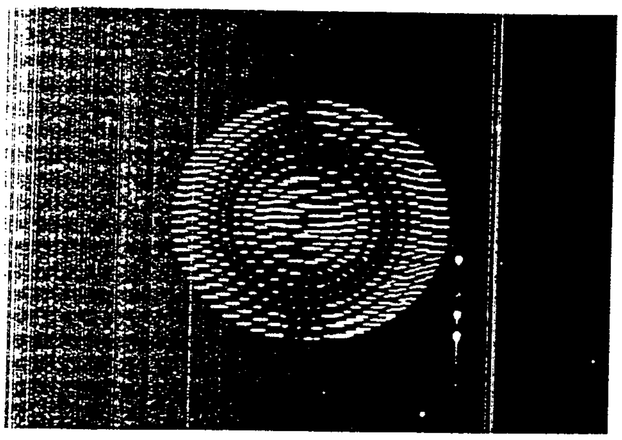

Figure 19: Velocity Vectors

References

1. E.H. Adelson, C.H. Anderson, J.R. Bergen, P.J. Burt, and J.M. Ogden, "Pyramid methods in image processing", RCA Engineer 29-6 pp.33-41 (Nov/Dec 1984).
2. D. Marr and E. Hildreth, "Theory of Edge Detection", Proc. R. Soc. Lond. B 207 pp.187-217 (1980).
3. G. Sandini and D. Vernon, "Tools for Integration of Perceptual Data", ESPRIT '86: Results and Achievements, pp. 855-865 Elsevier Science Publishers B.V. (North Holland), (1986).
4. D. Marr, Vision, Freeman and Co. San Francisco (1982).
5. E.C. Hildreth, The Measurement of Visual Motion, MIT Press, Cambridge, USA (1983).
6. B.K.P. Horn and B.G. Schunck, "Determining Optical Flow", Artificial Intelligence 17, no. 1-3, pp.185-204 (1981).
7. G. Sandini and M. Tistarelli, "Analysis of Image Sequences", Proc. IFAC Symposium on Robot Control, pp.289-292 (November, 6-8, 1985).
8. G. Sandini, V. Tagliasco, and M. Tistarelli, "Analysis of Object Motion and Camera Motion in Real Scenes", Proc. IEEE Int. Conf. on "Robotics & Automation", pp. 627-633, San Francisco, IEEE-CS, (April 7-10, 1986).
9. P. Morasso, G. Sandini, M. Tistarelli, "Active Vision: Integration of Fixed and Mobile Cameras", NATO ARW on Sensors and Sensory Systems for Advanced Robots, Berlin, Heidelberg, Springer-Verlag, (1986).
10. C. Frigato, E. Grosso, and G. Sandini, "Integration of Edge Stereo Information", ESPRIT Project 419 - Technical Report TK1-WP1-DI3 (1987).
11. G. Sandini, P. Morasso, and M. Tistarelli, "Motor and Spatial Aspects in Artificial Vision", Proc. of 4th. Intl. Symposium of Robotics Research, Santa Cruz, SRI International, (August 9-14, 1987).
12. D. Vernon and G. Sandini, "VIS: A Virtual Image System for Image- Understanding Research", Software: Practice and Experience, Vol. 18, No. 5, pp. 395-414, 1988.
13. H. Freeman, "On the Encoding of Arbitrary geometric configurations", IRE Transactions on Electronic Computers, pp. 260-268, (1961).
14. H. Freeman, "Computer processing of line-drawing images", ACM Computing Surveys, Vol. 6, No. 1, pp. 57-97 (1974).