SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 18(5), 395–414 (MAY 1988)

VIS: A Virtual Image System for Image-Understanding Research

DAVID VERNON
Department of Computer Science, Trinity College, Dublin. 2, Ireland

AND

GIULIO SANDINI
Department of Communication, Computer, and Systems Science, University of Genoa, Genoa, Italy

SUMMARY

Image understanding is concerned with the elucidation of a computational base inherent in perceiving a three-dimensional world using vision. This paper describes a low-level (or early) vision software system, developed in the context of current collaborative research activities in image understanding, which goes some way toward fulfilling the goals of portability, ease of use, and general-purpose extensibility. Since visual perception uses several types of disparate, but interrelated, information in some explicit cognitive organization, a central objective of the work is to represent this information in a coherent integrated manner which allows one interactively to investigate the properties of the interdependency between information types.

KEY WORDS Image understanding Vision Cue integration Software tools

INTRODUCTION

Image understanding is concerned with the elucidation of a computational base inherent in perceiving a three-dimensional world using vision. As such, it necessitates the development of tools which will facilitate the integration and interaction of perceptual information, i.e. visual sensory data, and, in addition, the presentation of this information to, and its interaction with, cognitive capabilities. Since computational theories of perception are not well developed, a software research environment should not prejudice the research philosophy nor any investigation of such computational theories. Unfortunately, this is a difficult goal to achieve, and the approach that has been adopted in this instance takes cognizance of this and restricts itself to one particular (and popular) philsophy.[1–6] However, every attempt has been made to extract the most out of the approach by avoiding *a priori* assumptions regarding the intrinsic value of information in both the design and implementation of its representational framework.

This research activity is a collaborative effort and involves five research centres: two based in Ireland, two in Italy, and one in the Netherlands.† As each centre must be

† Department of Computer Science, Trinity College, Dublin 2, Ireland; Captec Ltd., Malahide, Co. Dublin, Ireland; Department of Communication, Computer, and Systems Sciences, University of Genoa, Italy; VDS Ltd., Florence, Italy; Department of Experimental Psychology, University of Nijmegen, The Netherlands.

0038–0644/88/050395–20$10.00

Received 5 January 1987
Revised 4 August 1987

able to use the vision system, a major design criterion was to make the software as portable as possible. In particular, the system was written in the C programming language and currently runs on a VAX 11/780 under VMS, a VAX 11/750 under Unix, and an IBM PC under MS-DOS, and an IBM PC-AT under Xenix. These systems use a variety of image acquisition subsystems including a VICOM image processor, a Digithurst Micro-Eye frame-grabber, Imaging Technology PC-VISION and FG-100 frame-grabbers, and a VDS 7001 Eidobrain image processor.

Since the system described here is itself part of an ongoing research effort, it is mandatory that it provide simple tools for adding new analysis and processing facilities and modifying existing ones. Further, the system should be easy to use and, indeed, it should encourage use. A menu-based user/system dialogue has been adopted and a mechanism provided to allow a user to record a sequence of menu selections which may be saved on file and subsequently replayed. Several such files may be generated, and a directory of these 'command' files is available upon request. All menu-options are supported by on-line help texts. In addition, any processing and analysis session may be suspended and resumed at a later date. A data-flow paradigm was adopted to allow the user to manipulate images merely by specifying source and destination images; the types of these source and destination images define an implicit transformation which is effected upon image transfer. Additionally, the system allows extensive windowing in both source and destination images so that, in addition to being transformed, the destination image may be a scaled and translated version of the source image.

It was decided at the outset of the research project that this imaging system should place particular emphasis on the investigation of low-level, or 'early', visual processes. In particular, the system was to incorporate features to enable integration of distinct low-level visual cues and to allow interactive investigation of the characteristics of such integration. As the approach to low-level vision taken in this project is much in sympathy with that of David Marr and his associates at M.I.T.,[1–6] many of these cues are based on object boundaries, stereopsis, and motion, all of which are derived from the zero-crossings of the Laplacian of an image which has been convolved with a Guassian mask.[7, 8] For example, see Figure 1. Thus, the system was developed with reference to the subject matter of low-level vision while allowing more global design criteria to be finalized, as the project proceeded. This initial restriction in design does not appear to have had any detrimental effects on its general applicability.

This tool for integration of perceptual (initially, visual) information was dubbed a virtual image system (VIS), as it allows a user interactively to build image system structures from scratch, beginning with a null system in which no images exist, and adding images of an appropriate type as required. Images are organized hierarchically into pyramids,[9, 10] with image resolution varying from 1024×1024 pixels to 64×64 pixels; images may be one of several types, for example frame stores, intensity, convolution (convolved with a Laplacian of a Gaussian mask) or zero-crossing images. Other image types are also available (for example, images which make explicit the zero-crossing slope and orientation, boundary motion, stereo disparity and regional relationships). The system can be configured so that it comprises as many pyramids as necessary and images may be added or deleted, and their attributes (for example, an associated mask and window) can be modified dynamically, i.e. at run-time.

The dynamic configuration of the hierarchically organized image structures, which may differ in both content (information type) and representation, is achieved by associating with each image an image descriptor which defines the extent, type, depth

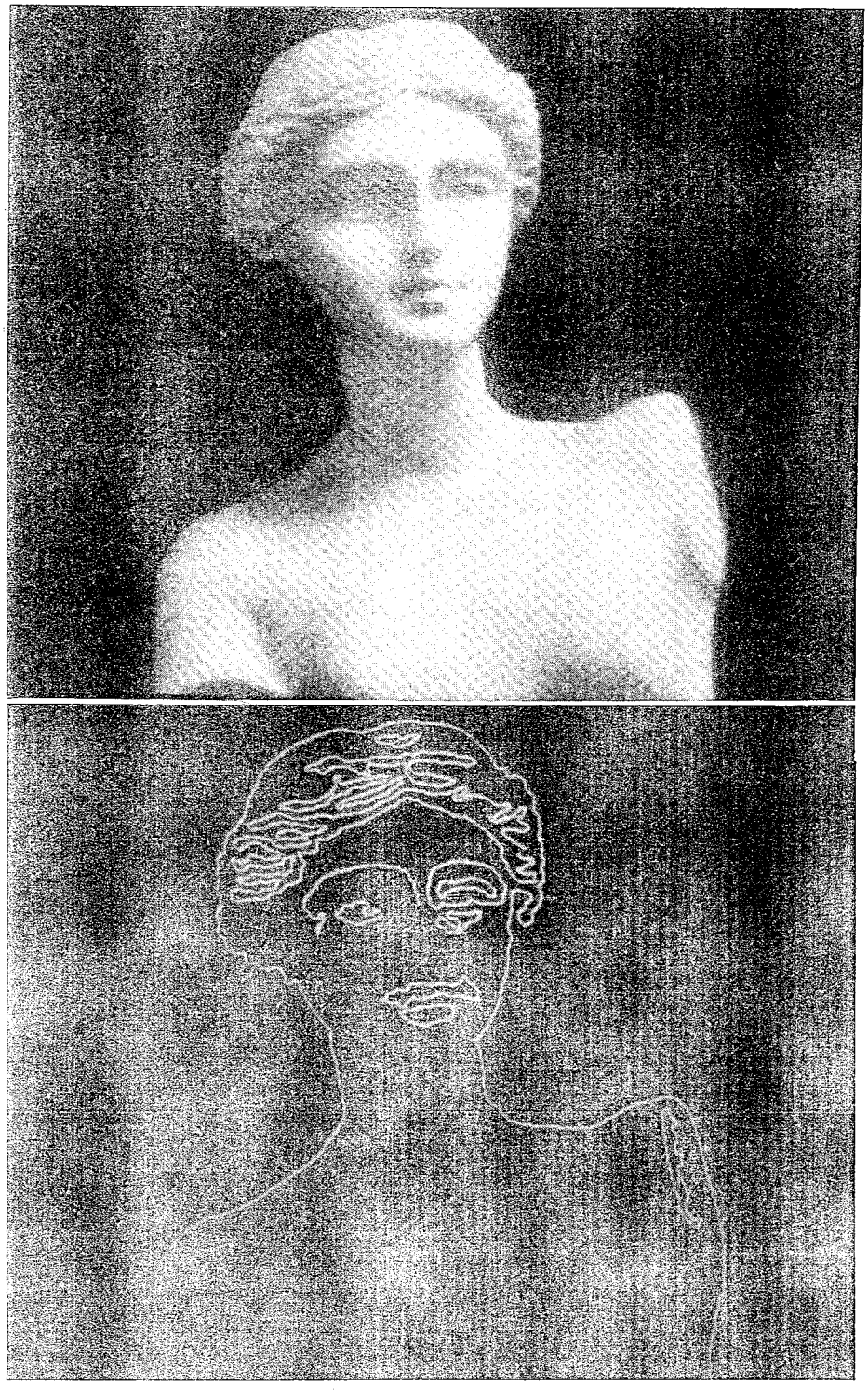

Figure 1. Intensity image and zero-crossings of statuette

and other image characteristics. Hierarchical organization is achieved by the use of pyramid descriptors, which consist of a pyramid label, a description and a linked list of pointers to image descriptors. Significantly, the representation of an image can be of several types, for example arrays or augmented BCCs (boundary chain codes).[11, 12] The type of image is dependent on the type of information to be made explicit. The effectiveness and usefulness of this representational structure is enhanced by the ability to associate logical relationships between different types of data, pertaining to the same visual scene. This is the mechanism by which the low-level visual cues are integrated and investigated.

The remainder of this text deals, in a little more detail, with the menu system and user dialogue, the virtual image system data-structures, system processing functions, and intended extensions which address cognitive reasoning and cursive script analysis, respectively.

A SUMMARY OF THE MENU-BASED USER-INTERFACE

Dialogue between a user and the virtual image system (VIS) is effected, wherever possible, by the use of self-explanatory menus. This ensures that the system is easy to use and that fast and efficient interaction is possible. When menu-based interaction is not appropriate, the system will invoke a question-and-answer session with the user, requesting some alphanumeric reply. In general, the user will have the option of aborting this dialogue.

There are two types of menu option: those invoked by numeric key and those invoked by depressing a (mnemonic) alphabetic key. The latter type is intended for frequently used menu options (e.g. Transfer an image, Window modification, display system Status, display Help text, revert to Previous menu and Quit are invoked by hitting T, W, S, H, P and Q, respectively). The mnemonic options are available on every menu. All screen-handling primitives assume that the terminal device is VT100 compatible; this affords a relatively efficient mechanism for cursor manipulation in a standard, and portable, manner.

Some general-purpose utilities, intended to increase the efficiency of usage of the system, are provided. These include the following.

A *learn* menu option provides the facility to generate a file containing several sequential menu selections, which can then be later reinvoked using the *replay* menu option. Several such command files may be generated and stored on the system: the *directory* menu option enables the user to display a directory of the command files which were generated using the *learn* facility.

Since an integral feature of VIS is the dynamic configuration of image structures during any interactive session, a synopsis of the current status is available upon request using the *system status* option; the status report will step through each level and pyramid sequentially and at all stages the user is afforded the opportunity to skip to the next pyramid, to a specified pyramid, or to terminate the summary altogether.

As VIS is an interactive system for investigating the form and integration of various perceptual cues, it is mandatory that the researcher be able to suspend his/her investigations and experiments and to resume them at a later date. This facility is provided through two menu options, *save* and *restore*, which allow the user to save the current status of the system on a named file and subsequently restore it (or any other saved status).

Finally, on-line assistance is available for each menu option in the form of *help* texts. This information may be obtained by typing 'H' followed by the menu option number. 'HH' will invoke the display of a general help text detailing this procedure for obtaining option-specific assistance.

STRUCTURE AND ORGANIZATION OF THE VIRTUAL IMAGE SYSTEM (VIS)

As mentioned in the introduction, VIS is designed to allow dynamic, i.e. run-time, configuration of the image structures used in any one processing session. Thus, a user can build a system from scratch, beginning with a system in which no images exist, and subsequently adding images (of an appropriate type) as desired and as the situation demands. Since images are organized hierarchically into pyramids, and a system may typically be configured to comprise more than one pyramid, the system may be thought of as a list of 'pyramid descriptors'; each pyramid descriptor detailing the exact nature and structure of its constituent images (refer to Figure 2). In particular, a pyramid descriptor points to several 'image descriptors', each of which details the exact make-up of the image at that level; for example, image size, level number, the number of bits, and an associated bit mask. In addition, each image descriptor points to a structure which is appropriate to the type of that image, for example, an image descriptor of a grey-level image is linked to a two-dimensional array of bytes representing that image. Alternatively, an image descriptor of a frame store is linked to a frame-store descriptor which, in turn, indicates the global structure detailing the operational characteristics of a particular frame-store type.

In general, an image may be one of several types, for example, frame stores, intensity (grey-level), convolution (convolved with a Laplacian of a Gaussian mask),[7, 8] or zero-crossing types.[7, 8] Other types include pseudo-images which make explicit the zero-crossing slope and orientation, boundary motion,[13, 14] stereo disparity,[15, 16] and regional relationships (based, again, on the Laplacian of a Gaussian filtered image).[17] Note that the image representation is of a form which is most appropriate to the information it makes explicit; thus grey-level images, convolution images and zero-crossing images are represented by two-dimensional arrays, whereas contour-based pseudo images (contour direction, slope and orientation, stereo disparity, motion) are represented by series of boundary chain codes (BCCs), where each node of the BCC represents the information appropriate to that image type at a point on the contour. Regional pseudo-images, on the other hand, are represented by a tree structure which makes explicit the inherent nested structure of image regions. The term pseudo-image is used to differentiate these explicit information representations from the iconic two-dimensional organization more commonly associated with the term image.

The effectiveness and usefulness of these varied representations is enhanced by an implicit facility to associate logical relationships between different types of data pertaining to the same visual scene. For example, each contour in an image of one distinct type is explicitly linked to the corresponding contour in all other distinct types. Further, each contour in a contour-based image effectively determines a region, and each region is explicitly linked to (pointed to by) the corresponding regional descriptor in the region image (which would be just a tree-structured organization of regional descriptors).

The integrated organization and the component data structures are depicted, for a typical system configuration, in Figure 2. There follows a brief description of each of the main data structures in the system.

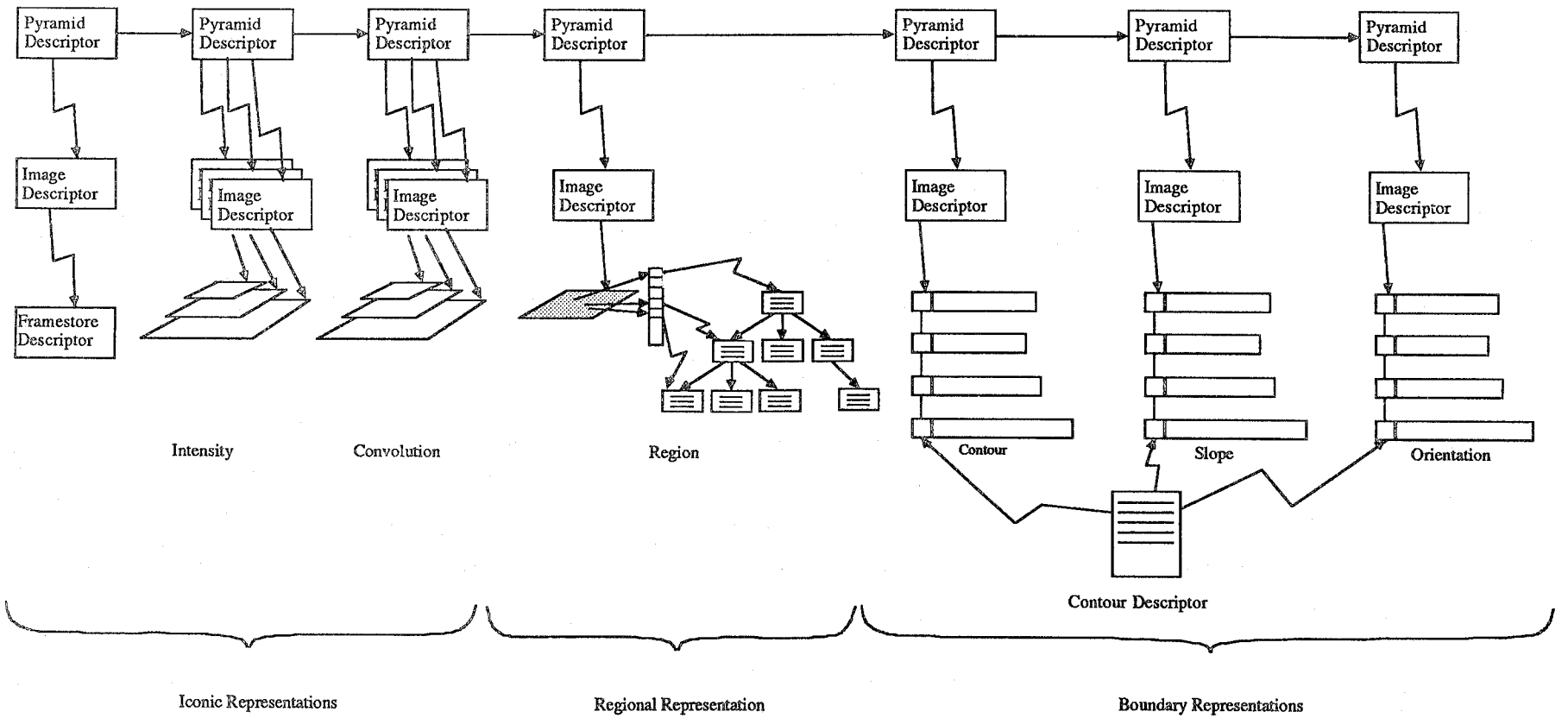

Figure 2. Top-level overview of the virtual image system data-structures

Pyramid descriptor

A pyramid descriptor, depicted in Figure 3, comprises a header node, containing the pyramid identification number and an alphanumeric description, a link to the next pyramid in the system, and a linked list of pointers to image descriptors. This linked list is organized so that the image descriptors are in sorted order, with the first image corresponding to the image at the bottom of the pyramid, i.e. the lowest level or highest resolution. No duplication of images at the same level is allowed. The pyramids themselves are also ordered (i.e. the linked list of pyramid descriptors is ordered), beginning with the pyramid having the lowest identification number.

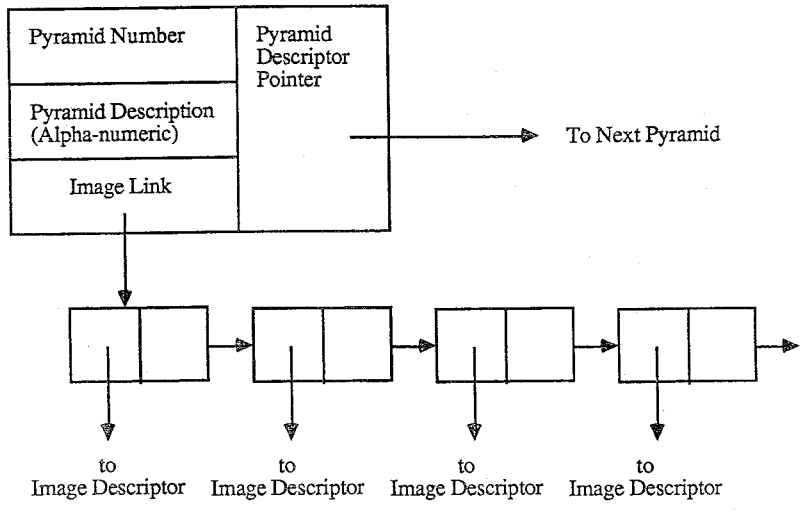

Figure 3. Pyramid descriptor

It may be argued that the use of a linked list of image descriptor pointers is redundant and ineffective. Not to adopt this strategy, however, would force either (a) the organization of the image descriptors themselves in a linear linked list, or (b) the specification, *a priori*, of the maximum number of image descriptors that can be handled by a pyramid descriptor. Significantly, provision would then have to be made for this maximum number in the definition of the descriptor structure (as in the case of a B-tree structure).[18] Since a tree structure was deemed desirable, it was decided not to opt for alternative (a). In the case of alternative (b), it was felt that the provision of sixteen link fields would be inefficient. The solution described above then provides tree-structuring without the space constraints in return for a slightly increased overhead in data-structure handling.

Image descriptor

An image descriptor, depicted in Figure 4, comprises several fields which describe the exact type of the image referred to by the descriptor. The fields include an internal data-type label, an alphanumeric description, the image size, the pyramid level number, a window specification, a bit-plane mask, the number of bits in the image, a pointer to the actual image, and a pointer to the parent pyramid descriptor.

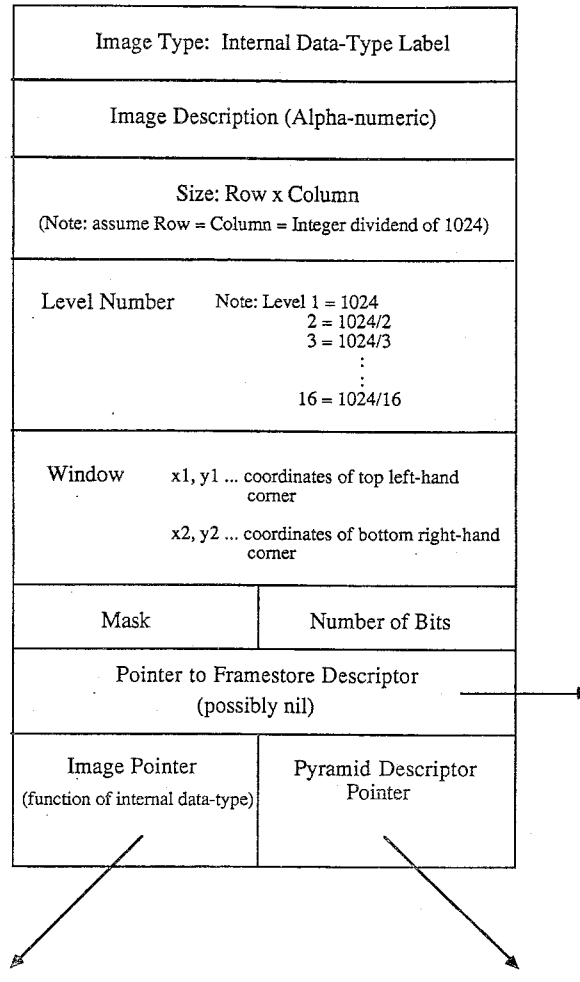

Figure 4. Image descriptor

The internal data-type is simply an integer value which identifies the type of image referred to by the descriptor. There are twelve distinct types at present. These are the frame store, grey-level (intensity) convolution, zero-crossing, contour, slope, orientation, disparity, velocity, region, depth and range image types, respectively. These image types are described in detail later. It is worthy of note that the implementation and integration of newly conceived image types (and data structures, in general) has proved to be a relatively simple task which can be accomplished with the minimum of effort.

The image size is given by the number of rows (or columns) in the image; it is assumed that the number of rows is equal to the number of columns and, further, that this number is an integer dividend of 1024. The level number is, in fact, the corresponding integer divisor: thus, an image of size 512×512 pixels would be a level-2 image, since 512 = 1024/2.

The window specification is determined by the co-ordinates of the top left-hand corner and the bottom right-hand corner of a rectangle; only that section of an image

enclosed by this rectangular window is the subject of system processing and analysis functions.

Any image may be configured, locally, to comprise a distinct number of bits, and any one of these bits may be masked (write-protected) by turning on a particular bit (i.e. setting it to logical 1). At present, this write-protect feature has been implemented for frame-store image types only.

There are two pointer fields in the image descriptor: these are the pyramid descriptor pointer and the image pointer, which provide links to the parent pyramid descriptor and to the actual image representation, respectively.

Frame-store image

A frame store is, inherently, device-dependent, and in order to achieve the required virtual nature of VIS, the dependency must be catered for. This is accomplished by the use of a frame-store descriptor (which takes the place of a physical image within the context of VIS) and by the use of several distinct frame-store-specific structures which describe the operational characteristics, i.e. the appropriate addresses and data for all registers required to handle the frame-store device. The frame-store descriptor comprises five fields in total, detailing the frame-store identification type (an internal integer code), the access type (whether it is configured as a memory-mapped device, an input/output device, or a combination of the two), the current mask value, the number of bits, and the pointer to the (global) frame-store-specific structure containing the device's operational characteristics. Note that the mask and number-of-bits fields are, actually, redundant in that they merely replicate the same information resident in the image descriptor. Figure 5 details the structure of the frame-store descriptor. To facilitate the exchange of iconic image information between VIS and other imaging systems, the concept of a frame-store has been extended somewhat to incorporate a new frame-store type (a virtual frame-store) which is, in effect, no more than a raw binary file. Thus, image data can be channelled to and from a file rather than a physical device. The file name currently associated with a virtual frame-store can be modified upon selection of the frame-store initialization menu option.

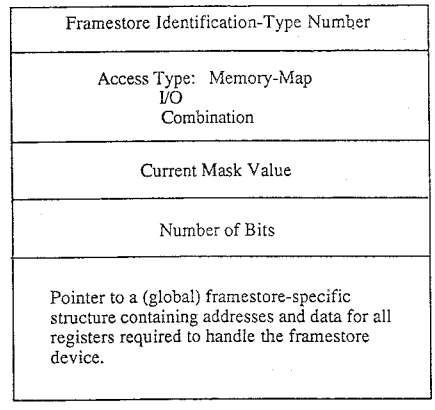

Figure 5. Frame-store descriptor

Grey-level (intensity) image

A grey-level image represents the intensity, or reflectance, of an imaged scene. This is normally represented by a two-dimensional array of byte values or pixel values. In the virtual image system, which is implemented in the C programming language, the representation differs slightly from the norm. An array image, of which the grey-level image is a specific example, is defined as a set of one-dimensional vectors or bytes; each vector (one-dimensional array) is addressed by an explicit pointer which is itself stored in a one-dimensional array of pointers. Thus, an array image (of bytes) is just an array of pointers to arrays of bytes and any variable of this type is just a pointer to a 'char' pointer. Note, however, that one may still reference an element in the image using the familiar double subscript, e.g. image[i][j]. In this case, the image value is accessed by indirection, not by using the normal subscription evaluation, and is, hence, more efficient. It also allows VIS to overcome the problems associated with the segmented memory architecture of some microcomputers.

Convolution image

A convolution image, in the context of this system, is an image which has been convolved with a Laplacian of a Gaussian mask (the first step in extracting the intensity discontinuities in an image) and is represented by an array image of the type described above.

Zero-crossing image

The zero-crossing image, again represented by an array image-type, details explicitly all the points in a convolution-type image where the image function changes sign, i.e. traverses (or crosses) zero. The zero-crossings represent points of intensity discontinuity in the original image and, normally, correspond to perceptual edges or boundaries. A property of the convolution with the Laplacian of a Gaussian mask ensures that, in general, such zero-crossings form connected closed contours.

Contour pseudo-image

A contour image is a redundant image representation, in that it contains exactly the same information as the zero-crossing image, but it represents the information in a different manner. Specifically, the contours are represented, not by a two-dimensional array image, but by a series of lists. Each list element contains the Freeman chain code direction required to generate the next point on the contour; each list represents a single contour in the image. The lists themselves, which are represented by a linear array of bytes, are organized as a linked list (of contours).

Each list has a header comprising the link fields to the next contour, to the previous contour, and to the associated contour descriptor. In addition, the header contains the co-ordinates of the contour origin and its length. Note that this information is redundant as it is also contained in the contour descriptor (yet to be described) but it is convenient for some processing to have it available locally within the 'logical' contour image. Please refer to Figure 6 for a schematic representation of the contour image structure.

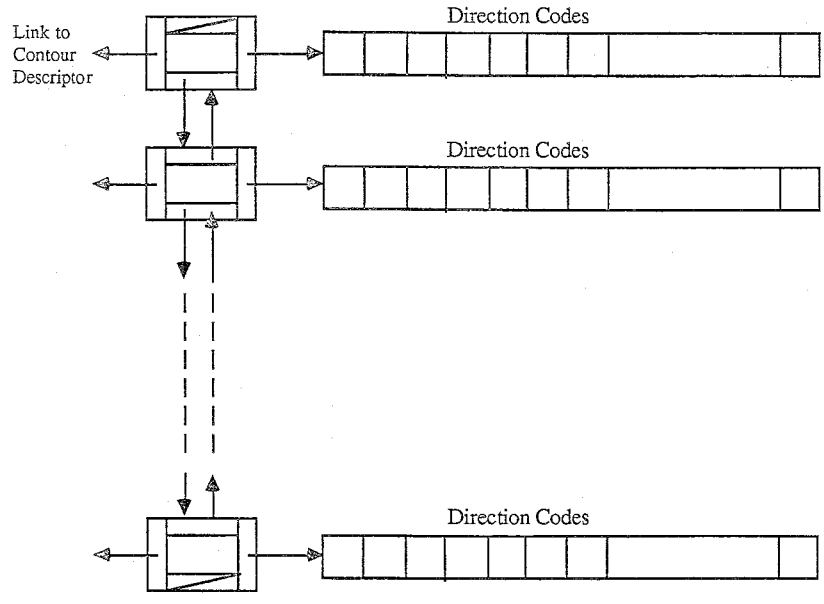

Figure 6. Contour image

Slope, orientation, disparity, velocity and depth pseudo-images

These images are all based on zero-crossing contours and they make explicit some intrinsic property based on analysis of (a series of) these contours. For example, the slope image details the slope, or steepness, of the positive-to-negative transition (a measure of edge strength) at each point of each contour in the image. Similarly, the orientation image details the local orientation, or direction, of the contour at every point on each contour in the image. The disparity image makes explicit the stereo disparity of each point on every contour in an image on the basis of a stereo pair of images of a scene.[15]

The velocity image makes explicit the optical flow or visual motion of each point of every contour in an image on the basis of several images of a particular scene.[13, 14] The optical flow, comprising vector magnitude and phase angle, is derived from the time derivative of a (sequence of) convolution images. This time derivative is, in effect, the orthogonal component of the true flow vectors, which are subsequently computed on the basis of *a priori* assumptions of either object or camera motion and on the basis of vector trajectories constructed by tracking optical flow from contour to contour over an extended series of velocity pseudo-images. To facilitate this tracking, each point of a velocity contour also includes the identification of the corresponding contour in the next frame of the sequence: the corresponding pyramid, image level, contour number, and contour offset.

The depth image represents the distance from a viewer to a point on a zero-crossing contour. It is presently computed from the optical flow using assumptions of either camera or object motion. In the near future, it will also be possible to compute depth from the stereo disparity of a pair of images.

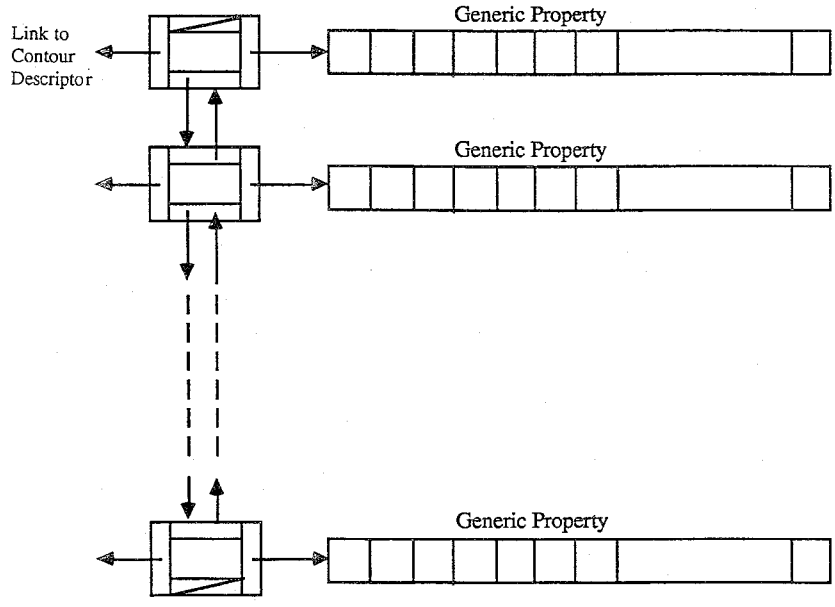

Figure 7. Slope, orientation, velocity and disparity images

Note that these pseudo-images are again represented, not by two-dimensional arrays, but by a representation similar to that of the contour image, i.e. a series of lists. Each element of the list contains the information appropriate to the image type, and each list represents a single contour in the image. Refer to Figure 7 for a schematic representation of these pseudo-images.

Range image

The range image is an array image type, which is derived by interpolation from the depth pseudo-image. It makes explicit the distance from the viewer to all visible points on an object's surface.

Contour descriptor

Since all of the contour-based images above are identical in representation and are based on the same image information, i.e. zero-crossing contours, it is reasonable to attempt to integrate the information in a coherent manner by forming logical links between the constituent lists corresponding to the contours in each image. This is accomplished with the contour descriptor which comprises two sets of fields. The first set of fields contains explicit links to the corresponding contour in each contour-based pseudo-image, whereas the second set contains information regarding gross contour statistics, for example, co-ordinates of the contour origin, contour length, enclosed area, mean slope, standard deviation of slope, mean orientation and standard deviation of local orientation, and a measure of the variation in local orientation. Thus one single contour descriptor links the corresponding contour in each of the contour, slope, orientation, disparity and velocity pseudo-images. Further, there are two additional fields which provide links to the previous contour descriptor and the next contour descriptor. Thus, the descriptors themselves are organized as a linear list and it is possible to run through these lists of descriptors searching for, say, all contours whose gross statistics satisfy some criterion and then refer to them explicitly in the appropriate

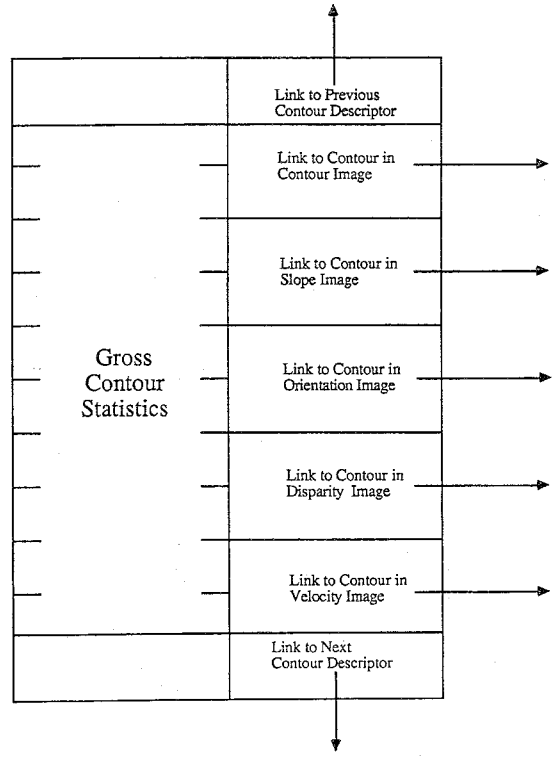

Figure 8. Contour descriptor

image. Figure 8 details the structure of these contour descriptors and one can refer, again, to Figure 2 to see how the overall linking structure is organized.

Region pseudo-image

The region pseudo-image is quite a complex image, comprising two distinct components: a region-crossing pseudo-image and a tree of region descriptors. When discussing the zero-crossing image, it was noted that the extracted zero-crossings formed closed contours.[17] Although these contours enclose a distinct region, there are at least two drawbacks with the representation. First, a zero-crossing is actually an inter-pixel transformation, and when one assigns the label of zero-crossing to a particular pixel, one does so in a nominal sense only. Secondly, the enclosed region is only represented in an implicit manner by the contour definition. A more accurate, and possibly a more flexible, representation involves the explicit identification of all pixels of negative sign and all pixels of positive sign. This is exactly what is meant by the region-crossing image. The representation is enhanced, however, in that each pixel of the region-crossing image is also an (8-bit) pointer or link to a one-dimensional array of true pointers which in turn form links to the corresponding regional descriptor in the so-called region tree, defined below. The local organization of this region-crossing image is shown in Figure 2.

Before describing the region tree, it is necessary first to define its components, i.e. the region descriptor. This descriptor, shown schematically in Figure 9, contains gross regional statistics regarding a single region in the region-crossing image (e.g. centroid, area, moment of inertia) and augments it with information pertinent to the corresponding contour representation of that same region. This contour information is exactly the

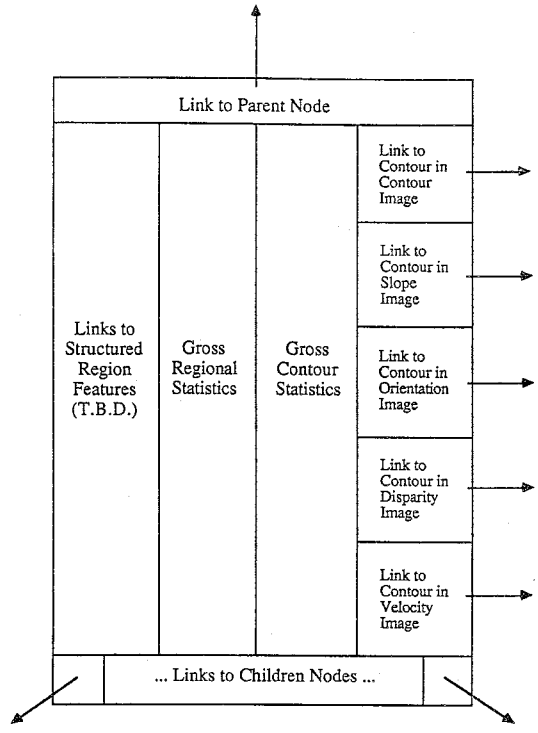

Figure 9. Region descriptor

information contained in the contour descriptors. In addition, the region descriptors have a set of link fields which point to or link to the corresponding contour in the contour, slope, orientation, velocity and disparity images. Finally, the region descriptor includes a further set of link fields to facilitate their organization in a tree-structured (hierarchical) manner; in particular there is a single link to the parent node (i.e. region descriptor) and several links to children nodes. Because a region may have an arbitrary number of regions nested within it, these offspring links are organized in a manner which is identical to that of the pyramid descriptors.

All regions in the region-crossing image are nested within some enclosing region, and they themselves possibly enclose some other regions, again by virtue of the property of convolution with a Laplacian of a Gaussian mask which ensures that zero-crossings form closed contours. Thus, each region is part of a nested structure which may be represented naturally in a hierarchical manner. This hierarchic organization is made explicit in the region tree which represents the region-crossing image a multi-branched tree of region descriptors.

In addition to the region and contour information contained in the node (region descriptor), each node is linked to the corresponding contour in the contour, slope, orientation, velocity and disparity images, and, further, each node is pointed to by the link resident in the region-crossing representation (via the one-dimensional array of region pointers). Figure 10 details the organization of this region image.

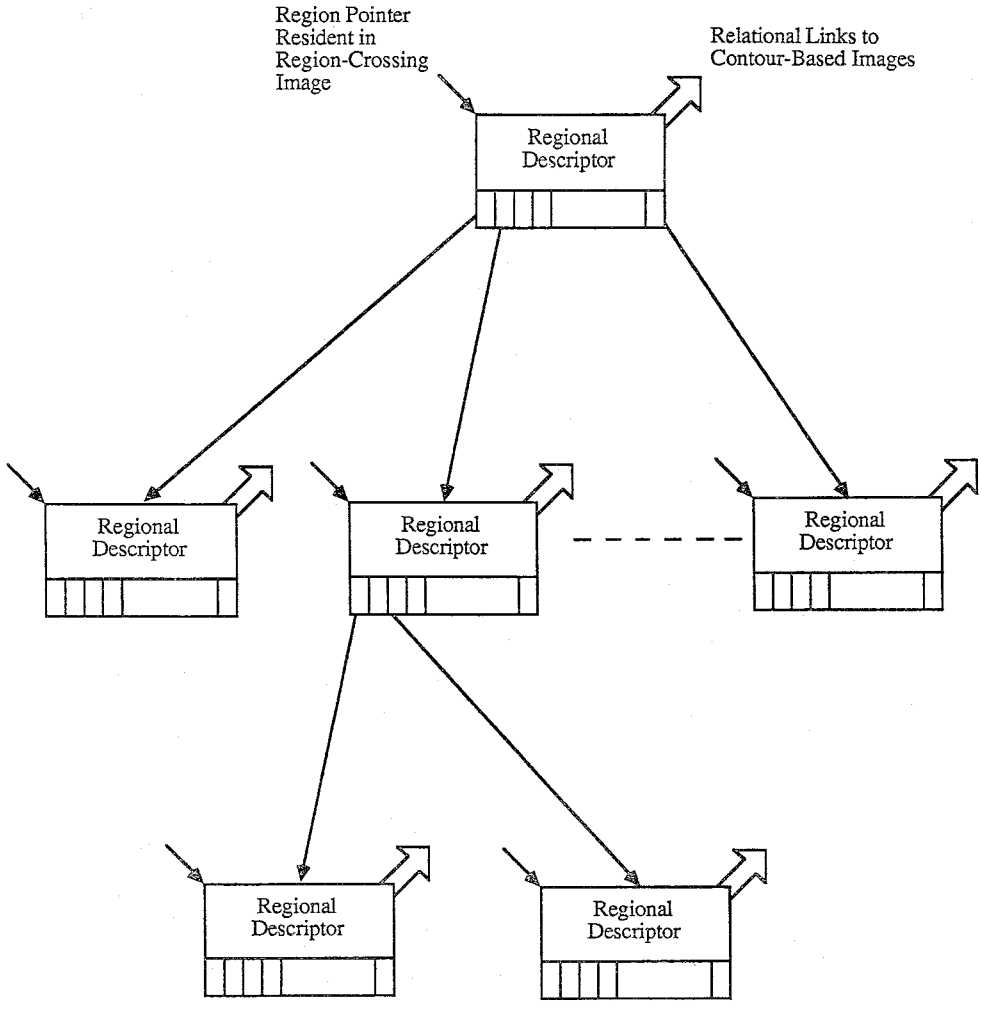

Figure 10. Region image

A SUMMARY OF IMAGE PROCESSING PRIMITIVES AND SYSTEM FUNCTIONS

The mechanism by which image processing transformations and analysis functions are effected by a user is based on a data-flow paradigm; the user manipulates images merely by specifying source and destination images and an implicit transformation, defined by the types of these source and destinations images; is invoked upon transfer. Since VIS facilitates extensive windowing in both source and destination images, the destination image, in addition to being transformed, may be a scaled and translated version of the source window.

The following sections detail the current functional capabilities of VIS, i.e. the processing and analysis functions which are available to the user as menu options.

Frame-store manipulation

There are, at present, three main frame-store manipulation options. These are: frame-store initialization, which causes all frame stores in the system to be (re-)initialized, real-time acquisition and display and video frame-grab. The real-time acquisition and display option provides the user with the ability to cause an incoming video signal (from a camera) to be continuously digitized and displayed, in real-time on a video monitor. When selected, a question and answer dialogue with the user is initiated to ascertain the frame-store which is to be used to accomplish the video capture. The frame-store is identified by its appropriate pyramid number and level number. In general, the user may abort the operation at any point by replying to a prompt with a null entry, i.e. by simply pressing ⟨return⟩. In a manner similar to the real-time acquisition and display option, the video frame-grab option provides the user with the ability to grab (digitize) a single frame of an incoming video signal in a frame-store. Again, a question and answer dialogue, similar to the acquisition option, is initiated. In the case of a virtual frame-store, initialization merely facilitates the modification of the associated file name; real-time acquisition and video frame-grab options are not valid for virtual frame-stores.

Image transfer

This option implements an interactive request to transfer the contents of one level of a pyramid to a (possibly different) level of a (possibly different) pyramid. The source and destination images cannot both be at the same level of the same pyramid. If the image types associated with these levels are different then the implicit transformation will be effected. Not all combinations of source/destination types are allowed; refer to Table I for a summary of the combinations that are currently implemented. The spatial organization of the destination image will depend significantly on the relationship between the window associated with the source image and the window associated with the destination image and also on the hierarchical/pyramidal relationship between the two images. Thus, the destination subimage will be an enlarged or reduced version of the source (in either the x or y directions) and is dependent on the spatial mapping dictated by the window 1 to window 2 spatial mappings and the pyramid level to level mapping. A pixel-filling algorithm is used in all cases and the appropriate co-ordinates in the source image are determined by a two-stage process comprising spatial mapping and bilinear interpolation.[19] The situation where both windows are identical and where both pyramid levels (i.e. the image resolution) are the same is treated explicitly as a special case to increase efficiency.

Note that, as before, the user may abort the request at any point of the interactive dialogue by replying to a prompt with a null entry.

Contour selection

All contours in the contour-based images (e.g. slope, orientation, disparity and velocity) are effectively linked by their associated contour descriptor, which makes explicit several useful statistics about the contour. New contour-type or zero-crossing images can be generated, interactively, on the basis of these global contour properties, and also, on the basis of local properties of the contour. In particular, entire contours can be selected for inclusion in the new image by providing threshold ranges for some subset of gross contour features (i.e. information stored in the contour descriptor). All

Table I. Currently valid image transfers

	Output image										
	F.S.	I	Conv	Z–C	Cont.	S	O	D	V	Dp	R
Frame-store (F.S.)	yes	yes	yes	no	no	no	no	no	no	no	no
Intensity (I)	yes	yes	yes	yes	no	no	no	no	no	no	no
Convolution (Conv.)	yes	yes	no	yes	no	no	no	no	no	no	no
Zero-crossing (Z–C)	yes	yes	no	no	yes	no	no	no	no	no	no
Contour (Cont)	yes	no	no	yes	yes	no	no	no	yes	no	no
Slope (S)	yes	no	no	yes	no	yes	no	no	no	no	no
Orientation (O)	yes	no	no	yes	no	no	yes	no	no	no	no
Disparity (D)	no	no	no	no	no	no	no	no	no	no	no
Velocity (V)	no	no	no	no	no	no	no	no	no	yes	no
Depth (Dp)	no	no	no	yes	no	no	no	no	no	no	yes
Range (R)	yes	yes	no	no	no	no	no	no	no	no	no

contours are selected which satisfy the Boolean condition:

$$(L_1 <= f_1 <= H_1) \text{ AND } (L_i <= f_i <= H_i) \ldots \text{AND } (L_n <= f_n <= H_n)$$

where i is the feature number and L_i, H_i, f_i are the low and high limits on the threshold and the contour feature, respectively.

All contours which satisfy the selection condition are effectively added to the current list of selected contours (in the fashion of a logical OR operation) and, in this way, it is possible to construct arbitrarily complex selection criteria. Further, selection criteria of partial contours based on local contour analysis are also provided. When transferring from a contour pseudo-image to a zero-crossing image, only those contours (and partial contours) which are currently selected are actually referenced. In transferring to contour pseudo images, only contours selected by global features (contour descriptor information) are transferred, although the partial selection information is retained. This mechanism provides a powerful tool for interactively investigating the interrelationship between intrinsic image types based on both global and local properties.

Modify window

This menu option allows the user to modify the window co-ordinates associated with a particular image either by specifying the co-ordinates of two corners of the windows, by propagating a window from another image (with or without scaling), or by translation of an existing window in the image. Window specification using graphic devices (e.g. mouse, digitizing tablet) are at present being implemented.

ANTICIPATED EXTENSIONS

A significant amount of the current research in the area of cognitive modelling is concerned with reasoning about perceptual and physical data. To some extent, it is organized along a hypothesis forming/checking paradigm using structural information theory.[20, 21] It is envisaged that VIS will form the kernel of an on-line perceptual database which is queried by the cognitive processes. Further, the cognitive processes will control the database in terms of its operational characteristics, initiating image transfers and supplying transfer parameters. It is acknowledged, however, that it will be necessary to introduce an extra interface subsystem to facilitate communication between these two modules. This is currently being implemented in the form of a command language interpreter. It is anticipated that VIS and the cognitive modelling system will communicate over an Ethernet, the latter generating command language programs and the former interpreting them and transmitting the requisite information over the network.

The system, as it stands now, is a unified autonomous tool which executes on a single machine. There are three considerations which motivate an extension beyond this organization. First, all of the processes performed in the research of vision are computationally expensive and some processes may be suited to a particular machine architecture more than others. It is evident that a facility to distribute this processing load in a simple manner would be very beneficial. Secondly, as more and more tasks are added to the system the overhead on integration and maintenance becomes quite significant; some method of using VIS without necessitating continuous growth is desirable. The final motivitation for an extension in organization is pragmatic: most researchers have invested many years in developing their own *modus operandi* and are ill-disposed to breaking completely with their well-established development environment. It is not realistic, then, to suppose that all parties would, or even should, integrate all their work into VIS, but rather that they should have the facility to use the tools that the system provides from within their own environment. It proposed to extend VIS in a manner which addresses all of these issues by exploiting the internal structure of the data structures, i.e. the system status. In particular, it is intended to save the system status, not in a single external file, but in several external files on several physically distributed, but networked, computers. Data of a particular type, and of interest to a particular research group, would be confined to a single subdirectory on a single machine. In this way the virtual image status is mapped onto a distributed processing network as a hierarchical system of subdirectories and files. Those wishing to use the data, then, can either invoke their local version of VIS, a subsystem constructed partially from VIS data-structure primitive functions, or by software which is capable of reading and interpreting the file format.

SUMMARY AND CONCLUSIONS

The system described in this paper endeavours to accomplish four things: the provision of a user-friendly and flexible software development environment, the support of several hardware systems, the ability dynamically to configure hierarchically organized image structures of several intrinsic types, and, most importantly, the facility to integrate and form logical links between these intrinsic images in a coherent manner.

The first of these objectives has been achieved by the use of a menu-based user-interface. These menus can be specified by the user/developer in a very simple and

methodical manner. Additionally, two features have been incorporated into the system which significantly increase the efficiency of usage. These are, first, a facility to record and replay a sequence of menu selections (and, in general, responses to system prompts) and, secondly, a facility to save and restore complete system status, allowing temporary suspension of any analysis and processing session.

A data-flow paradigm has been adopted in providing a mechanism by which image transformations (and analysis functions) are effected by the user. Thus, all that is required to process an image is the specification of the source image and the destination image, both of which will have image types associated with them. The transformation implicit in the transfer between images of differing types is effected automatically. Further, the system supports full windowing capabilities so that only specific areas, or windows, within both source and destination images are processed. This also provides an implicit ability to translate and scale subimages. Images are hierarchically organized, forming image pyramids of decreasing resolution. The ability to configure these pyramids dynamically at run-time has proved to be a very useful feature, not in the least because it is difficult to define, *a priori*, the requirements of a typical session spent investigating the repercussions of many transformations and the associated integration of the resultant low-level cues.

The judicious use of duplication (and hence, redundancy) of data within the system was seen to be particularly advantageous. The problems with the generation of this information and of maintaining its integrity were avoided by ensuring that just one or two general routines handle the data structures associated with any one information type. Information could then be retrieved from the structure most appropriate to the processing task at hand.

Since this tool is being used in a collaborative effort, software portability is extremely important. As the system is implemented in the C programming language, this has been achieved by the exclusive use of standard portable library function throughout the implementation and by adhering to standard (VT100) terminal-handling conventions. At present, VIS operates on a VAX 11/780 under VMS, a VAX 11/750 under Unix, and an IBM PC under DOS, and an IBM PC AT under Xenix, using a wide variety of frame-store devices. Finally, it is worthy of note that the increased development time and marginal decrease in run-time efficiency attendant on this commitment to portability has been more than compensated by the ease with which software and algorithms can be exchanged and integrated into the system.

ACKNOWLEDGEMENTS

This research was supported by the European Strategic Program for Research and Development in Information Technology (ESPRIT) under Project P419: Image and Movement Understanding.

REFERENCES

1. D. Marr, 'Early processing of visual information', *Philosophical Transactions of the Royal Society of London*, **B275**, 483–524 (1976).
2. T. Poggio, 'Marr's approach to vision', MIT A.I. Lab., *A.I. Memo No. 645*, 1981.
3. D. Marr, *Vision*, W. H. Freeman and Co., San Francisco, 1982.
4. S. Ullman, *The Interpretation of Visual Motion*, The MIT Press, Cambridge, Massachusetts, 1979.
5. E. C. Hildreth, *The Measurement of Visual Motion*, The MIT Press, Cambridge, Massachusetts, 1984.

6. W. E. L. Grimson, *From Images to Surfaces*, The MIT Press, Cambridge, Massachusetts, 1981.
7. D. Marr, 'Early processing of visual information', *Philosophical Transactions of the Royal Society of London,* **B275**, 483–524 (1976).
8. D. Marr and E. Hildreth, 'Theory of edge detection', *Proceedings of the Royal Society of London,* **B207**, 187–217 (1980).
9. A. Rosenfeld (Ed.) *Multi-Resolution Image Processing and Analysis*, Springer, 1984.
10. S. Tanimoto and A. Klinger (eds), *Structure Computer Vision*, Academic Press, 1980.
11. H. Freeman, 'On the encoding of arbitrary geometric configurations', *IRE Transactions on Electronic Computers,* 260–268 (1961).
12. H. Freeman, 'Computer processing of line-drawing images', *ACM Computing Surveys,* **6**, (1), 57–97 (1974).
13. G. Sandini and M. Tistarelli, 'Analysis of camera motion through camera sequences', Proc. International Conference on Advances in Image Processing and Pattern Recognition, Pisa, 1985.
14. G. Sandini, V. Tagliasco and M. Tistarelli, 'Analysis of object motion and camera motion in real scenes', *Proc. IEEE International Conference on Robotics and Automation*, San Francisco, 1986.
15. D. Marr and T. Poggio, 'A computational theory of human stereo vision', *Proceedings of the Royal Society of London,* **B204**, 301–328 (1979).
16. G. Sandini and M. Tistarelli, 'Recovery of depth information: camera motion as an integration to stereo', *Proceedings of the IEEE Workshop on Motion: Representation and Analysis*, 1986.
17. V. Torre and T. A. Poggio, 'On edge detection', *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **PAMI-8**, (2), 147–163 (1986).
18. E. Horowitz and S. Sahni, *Fundamentals of Data-Structures*, Computer Science Press, Rockville, MD., 1976.
19. K. R. Castleman, *Digital Image Processing*, Prentice-Hall, NY, 1979.
20. H. Buffart and E. Leeuwenberg, 'Structural information theory', *Internal Report*, Department of Experimental Psychology, University of Nijmegen, 1982.
21. E. Leeuwenberg, 'Quantification of certain visual pattern properties: salience, transparency, similarity', in E. L. J. Leeuwenberg and H. Buffart (eds), *Formal Theories of Visual Perception*, Wiley, 1978.