[82] Sandini, G. and Tistarelli, M. 1990. 'Active Tracking Strategy for Monocular Depth Inference from Multiple Frames', IEEE Transanctions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 1, pp.13–27.

[83] Spoehr, K.T. and Lehmkuhle, S.W. 1982. *Visual Information Processing* Freeman and Co., San Francisco.

[84] Torre, V. and Poggio, T. A. 'On edge detection', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol PAMI-8, No 2, pp. 147-163.

[85] Ullman, S. *The Interpretation of Visual Motion* MIT Press Cambridge.

[86] Vernon, D. (Rapporteur) 1987, 'Sensoric Feedback for Control and Decision Support Systems', Report of a Workshop in the Framework of ESPRIT Conference 1987, Directorate General XIII (Eds.).

[87] Vernon, D. and Tistarelli, M. 1987. 'Range Estimation of Parts in Bins using Camera Motion', Proceedings of SPIE's 31st. Annual International Symposium on Optical and Optoelectronic Applied Science and Engineering, San Diego, California, USA, 9 pages.

[88] Vernon, D. and Sandini, G. 1988. 'VIS: A Virtual Image System for Image Understanding', Software — Practice and Experience, Vol. 18, No. 5, pp. 395–414.

[89] Vernon, D. 1989. 'Computers See the Light', Technology Ireland, Vol. 21, No. 1, pp. 21–23.

[90] Vernon, D. and Tistarelli, M. 1990. 'Using Camera Motion to Estimate Range for Robotic Parts Manipulation', IEEE Transaction on Robotics and Automation, Vol. 6, No. 5, pp. 509-521.

[91] Vernon, D. 1991, *Machine Vision* Prentice-Hall International, London.

[92] Waxman, A. M. and Ullman, S. 1985. 'Surface structure and three-dimensional motion from image flow kinematics', International Journal of Robotics Research, Vol. 4, No. 3, pp. 72-94.

[93] Wertheimer, M. 1958. 'Principles of Perceptual Organisation', in D.C. Beardslee and M. Wertheimer (Eds.), Readings in Perception, Princeton: Van Nostrand.

[94] Witkin, A. P. 1981. 'Recovering Surface Shape and Orientation from Texture' Artificial Intelligence, Vol. 17, pp. 17–45.

Chapter 2

An Overview of *VIS à VIS* A Virtual Image System for Image Understanding.

David Vernon and Giulio Sandini

The tools of working out salvation
by mere mechanic operation!

Samuel Butler
Hudibras

As we saw in the previous chapter, image understanding is concerned with the development 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 which will allow reasoning processes to operate on this data. In this chapter, we introduce a system called *VIS à VIS* - a Virtual Image System - which provides the computational infra-structure, an environment if you wish, in which the approaches to image understanding described in chapter 1 can be implemented. It should be clear from the overview of computer vision that there is a type of duality between the processes of vision and the representations of visual information on which these processes operate. The original motivation for the development of *VIS à VIS* was to make this 'duality' more explicit by restoring representations to a status which is equal to that of the attendent processes. Specifically, the idea was to provide a

representational framework wherein all of the representations outlined in the previous chapter could be simultaneously instantiated and operated on by the appropriate processes. The emphasis then is on the integration of representations or, at least, the development of a computational framework for the integration of representations. The purpose of this chapter is to describe the *VIS à VIS* data-structures which effect this framework and to outline briefly the manner in which *VIS à VIS* is used. The attendent processes (i.e. vision algorithms) will be described in subsequent chapters together with somewhat more detailed explanations of specific data-structures, where appropriate.

Note that, although the book is specifically concerned with parallel computer vision, we have not yet introduced the concept of parallelism. This will be done toward the end of the book, specfically in chapters 10 and 11, after the complete *VIS à VIS* computer vision system has been described. At that stage, we will discuss the development of the tools for coarse granularity parallelism; tools which turn out to have an elegance which arises from, and depends on, the original design of *VIS à VIS* itself.

Since *VIS à VIS* is part of an on-going research effort, it is essential that it provide simple tools for adding, or prototyping, new analysis and processing facilities and for modifying existing ones. Furthermore, the system should be easy to use and, indeed, it should encourage use. Two mechanisms for user-interaction are provided: an integrated (interpreted) programming language and an interactive menu system. The programming language *VISICL — VIS à VIS* Interpretive Control Language — is described in detail in chapter 10 and summarized later in this chapter, as is the menu-based interface.

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 defines 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.

The acronym *VIS*, for Virtual Image System, in *VIS à VIS* was chosen because of the manner in which the system of images is generated or instantiated. Specifically, the user interactively builds 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 organised hierarchically into pyramids, with image resolution varying from 1024x1024 pixels to 64x64 pixels. Images may be one of several types, for example, framestores, 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, 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-organised 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 and other image characteristics. Hierarchical organisation is achieved by the use of pyramid descriptors which consist of a pyramid label, description, and a linked list of pointers to image descriptors. Significantly, the representation of an image can be one of several types, for example, arrays or augmented BCCs (Boundary Chain Codes). 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, or links, 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 chapter deals in more detail with the virtual image system data-structures, system processing functions, and the menu system and user dialogue, respectively.

2.1 Data Structures.

2.1.1 A top-level overview.

VIS à 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. This interactive configuration facilitates the investigative nature of a research system, where the representational requirement may not be known *a priori*. Of course, any given system of images can be configured as required upon start-up to allow a given sequence of processes to proceed. Since images are organised hierarchically into pyramids, and a system may typically be configured to comprise more than one pyramid, the system can 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 2.1). 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 2-D array of bytes representing that image. Alternatively, an image descriptor of a framestore is linked to a framestore descriptor which, in turn indicates the global structure detailing the operational characteristics of a particular framestore type.

There are several types of image in *VIS à VIS*, for example, framestores, intensity (grey-level), convolution (convolved with a Laplacian of a Gaussian mask) or zero-crossing types. Other types include pseudo-images which make explicit the zero-crossing slope and orientation, boundary (or edge) motion, stereo disparity, depth, and regional relationships (based, again, on the Laplacian of Gaussian filtered image).

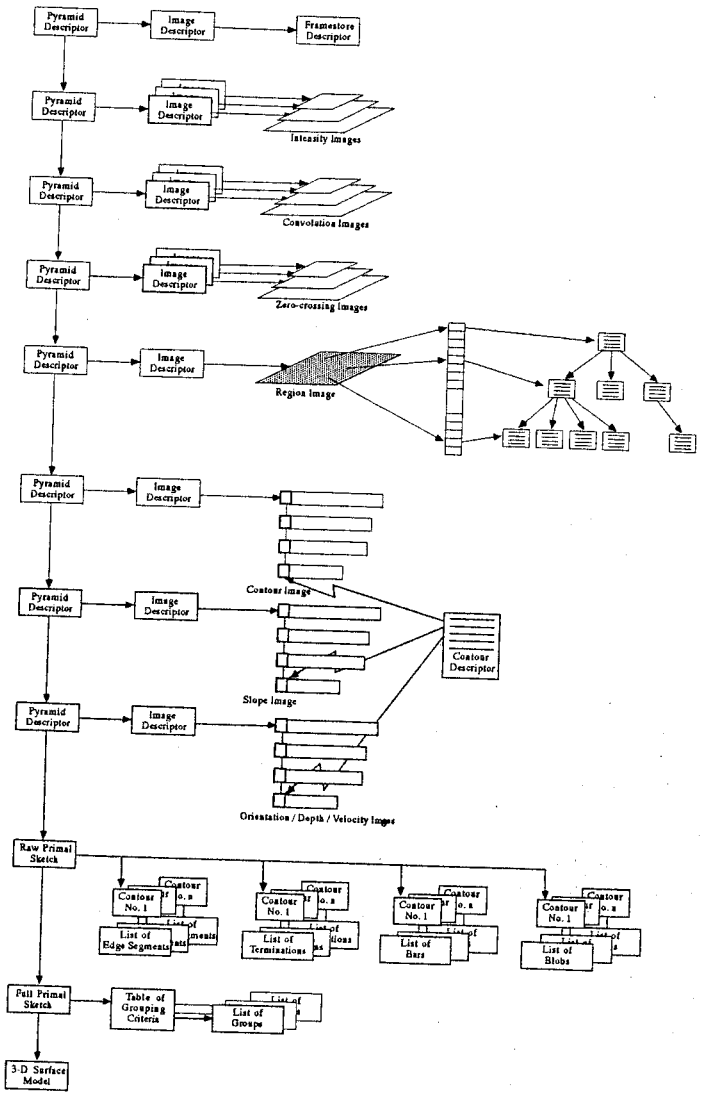


Figure 2.1: Top-level organization of *VIS ā VIS* data-structures

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 2-D arrays, while 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 organisation more commonly associated with the term image.

VIS ā VIS also includes the 'higher-level' data representations which are required for the construction of the 3-D world models, e.g. the Raw Primal Sketch, the Full Primal Sketch, and the 3-D models themselves.

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 organisation of regional descriptors).

The integrated organisation and the component data-structures are depicted, for a typical system configuration, in figure 2.1 and there follows a brief description of each of the main data-structures in the system.

2.1.2 The pyramid descriptor.

A pyramid descriptor, depicted in figure 2.2, 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 organised 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.

2.1.3 The image descriptor.

An image descriptor, depicted in figure 2.3, 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 alpha-numeric 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.

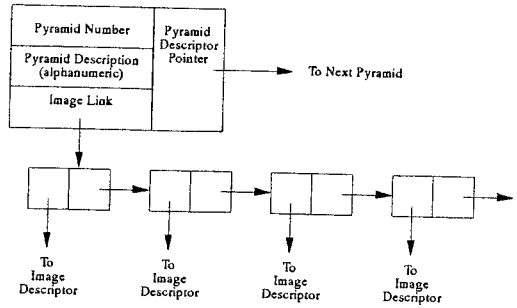


Figure 2.2: Pyramid descriptor

The internal data-type is simply an integer value which identifies the type of image referred to by the descriptor. There are sixteen distinct image types at present. These are the framestore, grey-level (intensity) convolution, zero-crossing, contour, slope, orientation, disparity, velocity, region, depth, range image types, together with the raw primal sketch, full primal sketch, and 3-D model. These image types are described in detail later.

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 512x512 pixels would be a level 2 image since 512 = 1024/2.

The window specification is determined by coordinates 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, logically, 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 framestore 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.

2.1.4 Framestore images.

A framestore image is a mapping to a physical device and is, inherently, device-dependent. In order to achieve the required virtual nature of *VIS ã VIS*, this dependency must be catered for. This is accomplished by the use of a framestore descriptor

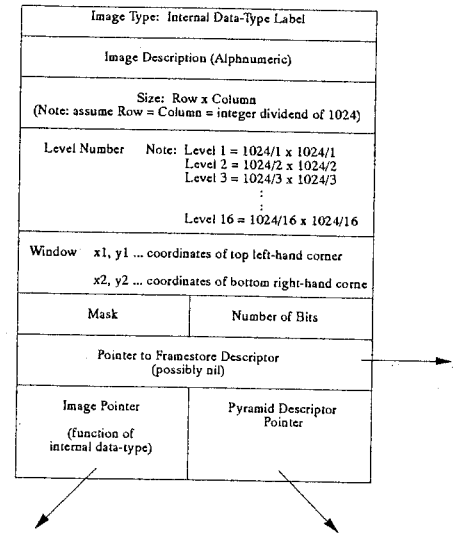


Figure 2.3: Image descriptor

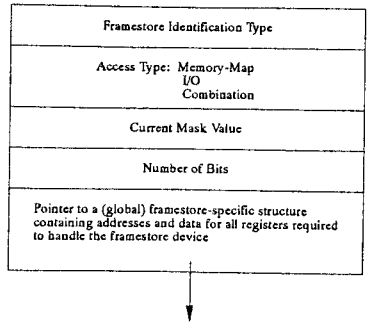


Figure 2.4: Framestore descriptor

(which takes the place of a physical image within the context of *VIS ã VIS*) and by the use of several distinct framestore-specific structures which describe the operational characteristics, i.e. the appropriate addresses and data for all registers, required to handle the framestore device. The framestore descriptor comprises five fields in total, detailing the framestore 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) framestore-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 2.4 details the structure of the framestore descriptor.

To facilitate the exchange of iconic image information between *VIS ã VIS* and other imaging systems, the concept of a framestore has been extended somewhat to incorporate a new framestore type (a *virtual framestore*) which is, in effect, no more than a disk file. Thus, image data can be channelled to and from a file rather than a physical device. The filename currently associated with a virtual framestore can be modified upon selection of the framestore initialisation menu option. This turns out to be an important feature since it allows image data to be transferred to and from *logical* devices or images rather than physical devices or images. The relevance of this feature will become more evident when we discuss the mechanisms required to achieve parallelism.

2.1.5 Intensity images.

An intensity image represents the grey-level, or reflectance, of an imaged scene. This is normally represented by a 2-D array of byte values or pixel values. In the

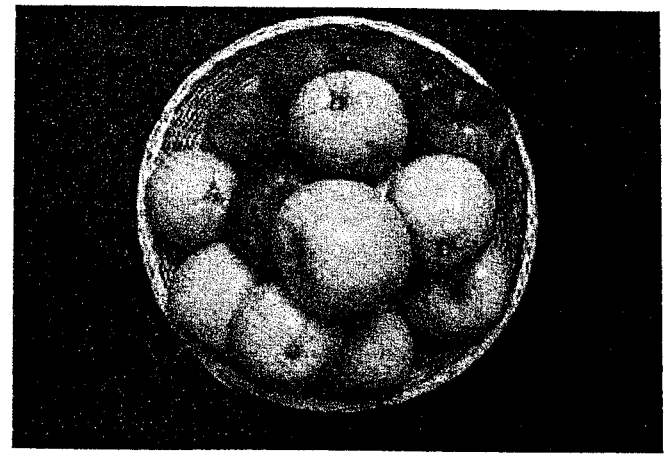


Figure 2.5: An example of an intensity image

Virtual Image System, which is implemented in the C programming language, the representation differs slightly from the norm. An array image, of which the intensity image is a specific example, is defined as a set of 1-D arrays (rows) of bytes; each row being addressed by an element of a 1-D 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 and not using the normal subscript evaluation and is, hence, more efficient. It also allows *VIS ã VIS* to overcome the problems associated with the segmented memory architecture of some microcomputers. Figure 2.5 shows an example of a 256x256 pixel intensity image.

2.1.6 Convolution images.

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. Chapter 3 deals in detail with this type of convolution filtering and figure 2.6 shows an example of a 256x256 convolution image which was generated using a Laplacian of Gaussian filter (standard deviation of Gaussian = 3 pixels).

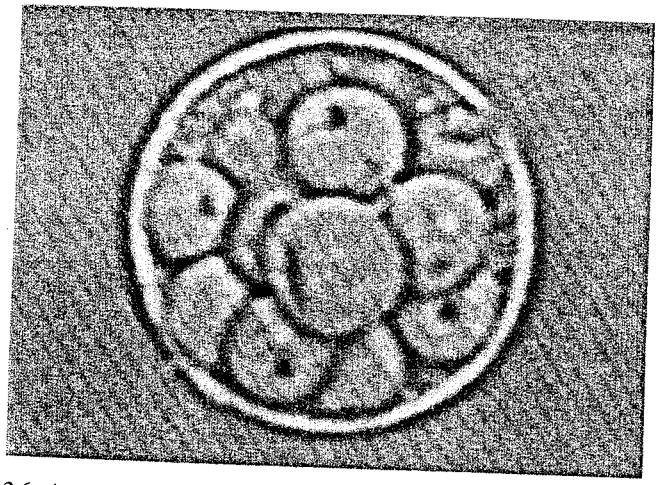


Figure 2.6: An example of a convolution image, i.e. an intensity image after convolution with a Laplacian of Gaussian filter

2.1.7 Zero-crossing images.

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. transverses (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 such zero crossings form connected closed contours. Again, refer to chapter 3 for details of this type of processing and see figure 2.7 for an example of the zero-crossing contours extracted from the convolution image in figure 2.6.

2.1.8 Contour images.

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 2-D array image, but by a series of lists. Each list element contains the boundary 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 organised 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 coordinates of the contour origin and its length. Note that this information is

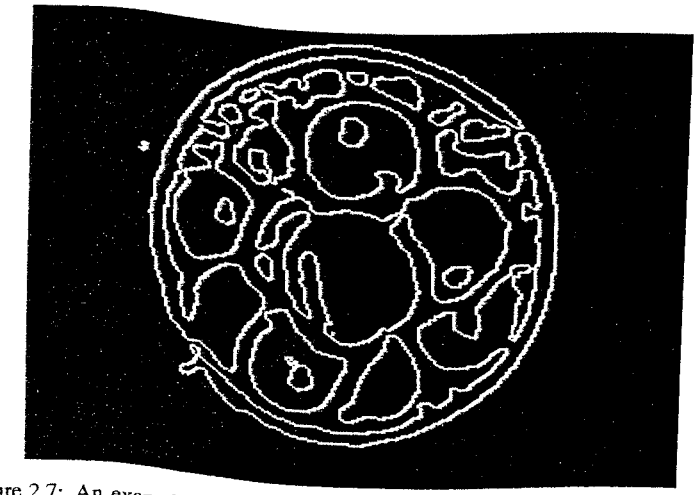


Figure 2.7: An example of a zero-crossing image, i.e. the zero-crossings in a convolution image

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. Refer to figure 2.8 for a schematic representation of the contour image structure.

2.1.9 Slope, orientation, disparity, velocity, and depth images.

Associated with Contour images, and represented in a similar manner, are 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; see figure 2.9. Similarly, the orientation image details the local orientation, or direction, of the contour at every point on each contour in the image; see figure 2.10. 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 (refer to chapter 4).

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. 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

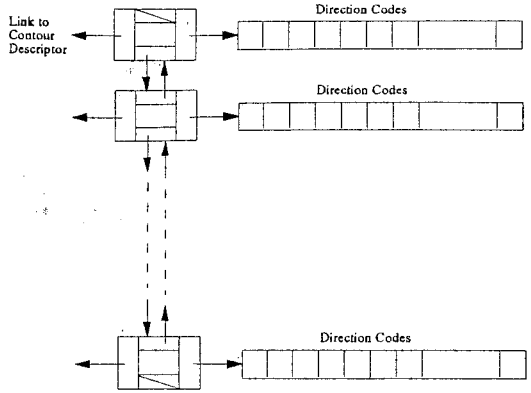


Figure 2.8: Contour image

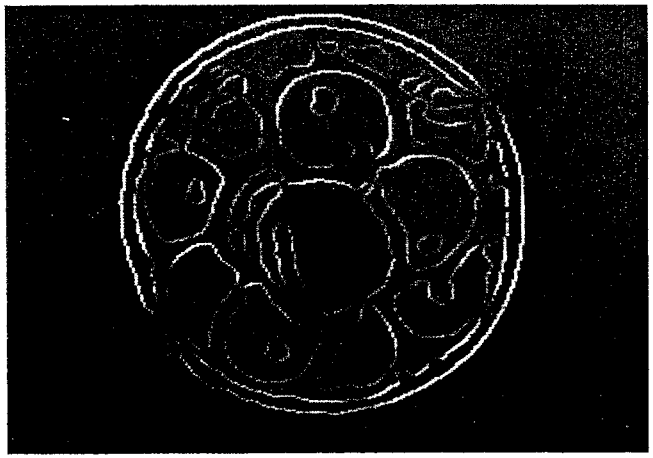


Figure 2.9: An example of a slope image, i.e. the relative strength of zero-crossings in a convolution image

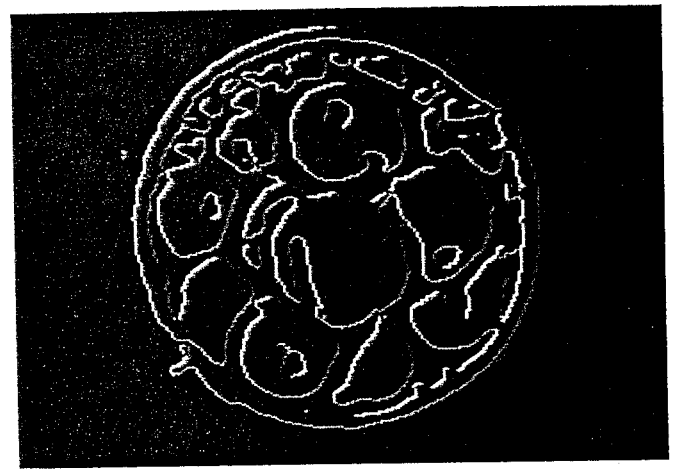


Figure 2.10: An example of an orientation image, i.e. the local orientations of zero-crossings in a convolution image

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. Chapter 5 describes the computation of optical flow in *VIS* $\vec{a}$ *VIS* in detail and figure 2.11 shows an example of a simple flow field.

The depth image represents the distance from a viewer to a point on a zero-crossing contour (see 2.12). It is presently computed either from the optical flow or from the stereo disparity of a pair of images.

Note again that these pseudo-images are represented, not by 2-D 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 2.13 for a schematic representation of these pseudo-images.

2.1.10 The contour descriptor.

Since each 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 set of fields. The first set of fields contains explicit links to the corresponding contour in

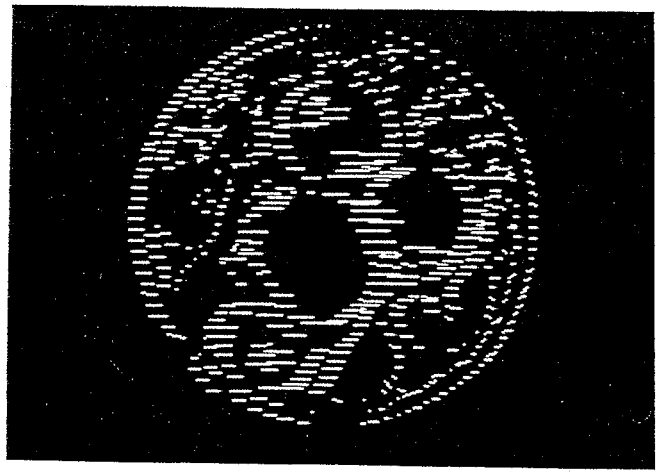


Figure 2.11: An example of a velocity image, i.e. the optical flow of zero-crossings in a sequence of images

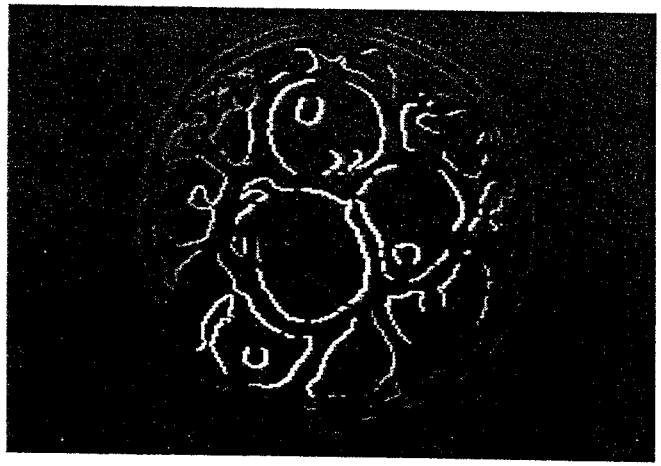


Figure 2.12: An example of a depth image (brightness is inversely proportional to distance from camera).

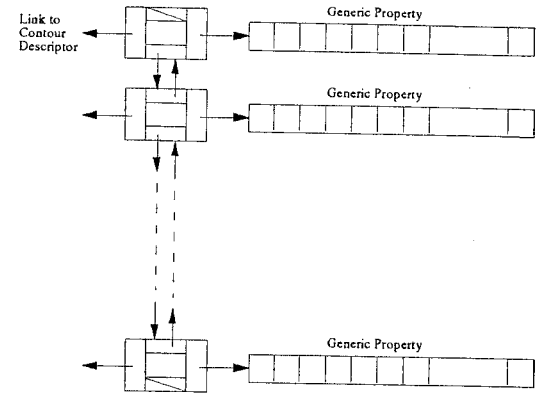


Figure 2.13: Slope, orientation, velocity, and disparity image

each contour-based pseudo-image while the second set contains information regarding gross contour statistics, for example, coordinates 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 organised as a linear list and it is possible to run these lists of descriptors searching for, say, all contours whose gross statistics satisfy some criterion (or set of criteria) and then refer to them explicitly in the appropriate image. Figure 2.14 details the structure of these contour descriptors and one might refer, again, to figure 2.1 to see how the overall linking structure is organised.

2.1.11 Range images.

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 (see figure 2.15). The interpolation required to generate the range image from the depth image is detailed in chapter 8.

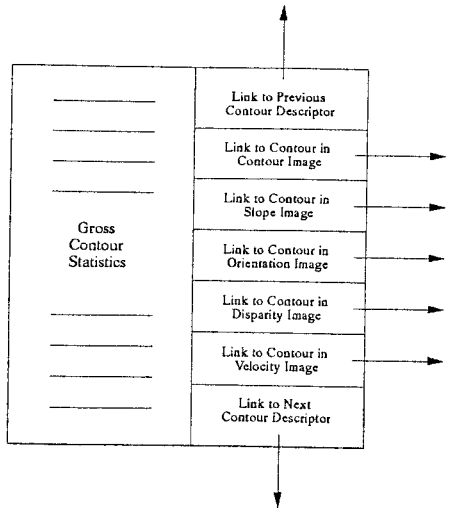


Figure 2.14: Contour descriptor

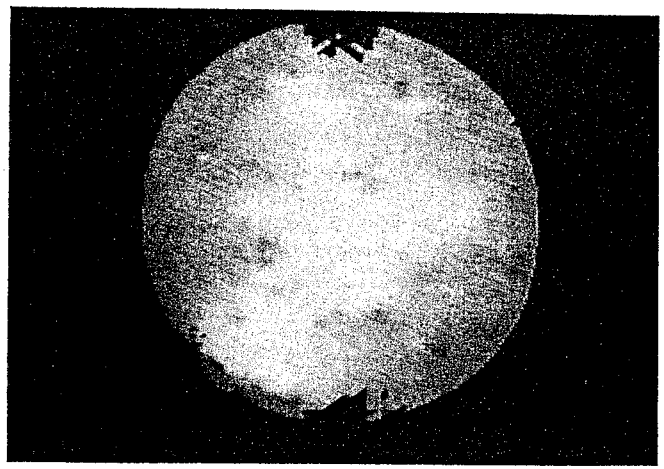


Figure 2.15: An example of a range image (brightness is inversely proportional to distance from camera).

2.1.12 Region images and the region-crossing image and region tree representations.

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, we noted that the extracted zero-crossings formed closed contours. While these contours enclose a distinct region, there are at least two drawbacks with the representation. Firstly, 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 1-D array of true pointers which in turn form links to the corresponding regional descriptor in the so-called region tree, defined below. The logical organisation of this region-crossing image is shown in figure 2.1

Before describing the region tree, it is necessary to first define its components, i.e. the region descriptor. This descriptor, shown schematically in figure 2.16, contains gross regional statistics regarding a single region in the region-crossing image (e.g. centroid, area, moment of inertia, energy) and augments it with information pertinent to the corresponding contour representation of that same region. This contour information is exactly the 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 organisation in a tree-structured (hierarchical) manner (see figure 2.17); 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 organised in a manner which is identical to that of the pyramid descriptors. An example of the regions in a region image is shown in figure 2.18.

In summary, the zero-crossings of images which have been convolved with Laplacian of Gaussian filters form closed contours. The regions bounded by these zero-crossing cntours are represented by the region image. All regions in the region-crossing image are nested within some enclosing region and, thus, each region is part of a nested structure which may be represented naturally in a hierarchical manner. This hierarchical organisation is made explicit in the region tree which represents these regions as a multi-branched tree of region descriptors. *VIS á VIS* also explicitly represents the regions as an associated region-crossing image. This is an iconic image in which each pixel represents the label of the region to which it belongs. Both the region tree and the region-crossing image are logically linked and, furthermore, each node is linked to the corresponding contour in the contour, slope, orientation,

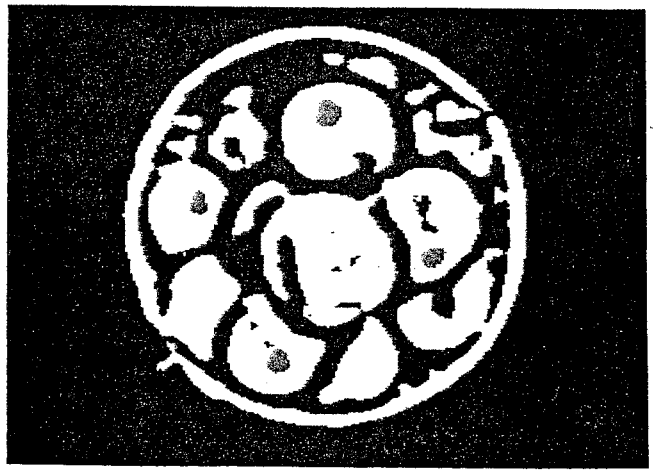


Figure 2.18: An example of a region image, i.e. the regions bounded by zero-crossings

coordinate pair.

The raw primal sketch is represented in *VIS à VIS* by several sequences of RPS (raw primal sketch) descriptors, with a different descriptor for each primitive. The sequences are organized according to their generating zero-crossing contour with one sequence of each particular (primitive) type of RPS descriptor per contour. Thus, there are (potentially at least) $4n$ sequences for each image, where n is the number of zero-crossing contours in the image. Figure 2.19 summarizes the organization of the raw primal sketch in *VIS à VIS* while figure 2.20 shows an example RPS.

2.1.14 Full primal sketches.

Useful though it is, the abstraction provided by the raw primal sketch is still limited. One of the most important shortcomings is that it is spatially restricted and does not convey any explicit information regarding spatially extended entities or objects. The full primal sketch is a representation derived from the raw primal sketch by processes of segmentation or grouping. In *VIS à VIS*, we have chosen to effect the grouping in a context-free manner, without recourse to top-down expectation-driven, or model-driven, analysis. Specifically, the grouping proceeds by forming groups of tokens on the basis of criteria similar to those of Gestalt psychology, i.e. collinearity, curvilinearity, spatial proximity, similarity, and equal spacing. The tokens which are grouped are the raw primal sketch primitives *and the groups themselves*. Thus, the grouping strategy is recursive and it is possible (indeed, it is often necessary) to form groups of groups. Chapter 7 deals with these grouping processes in detail; we

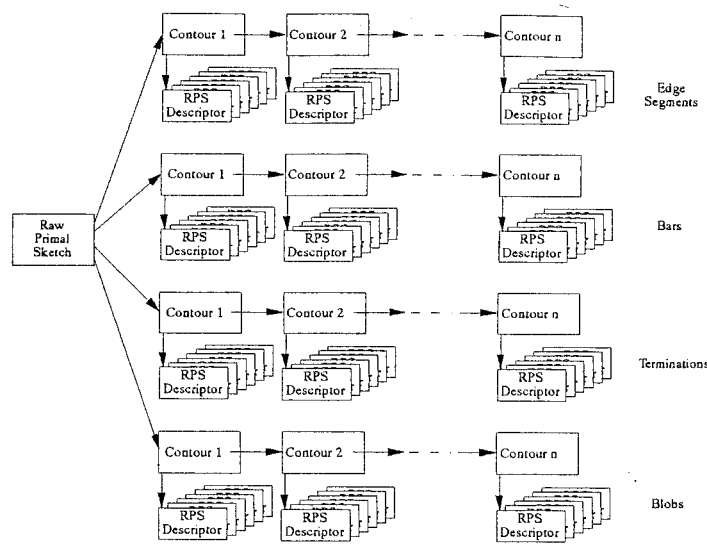


Figure 2.19: Organization of the raw primal sketch.

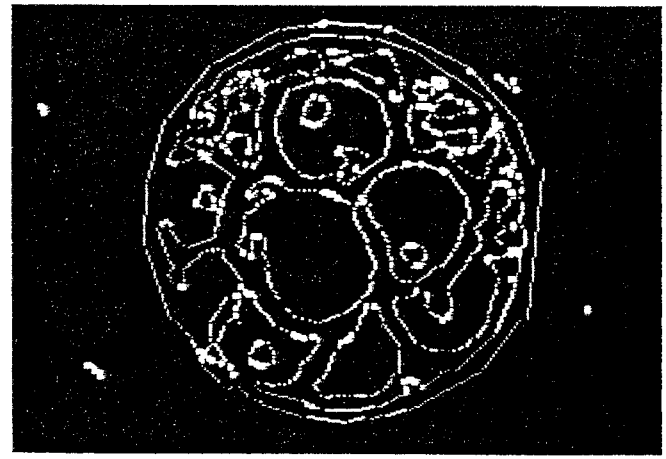


Figure 2.20: An example of a raw primal sketch.

2.2 *VIS à VIS* utilities.

The preceding section described the main data-structures in *VIS à VIS*. We will now proceed to outline the chief utility functions offered by *VIS à VIS* to users.

2.2.1 The System Status.

As *VIS à VIS* in an interactive system for investigating, among other things, 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). Upon invocation of the SAVE command, all the information contained in the system's data-structures is written to file, together with any information necessary to reconstruct the organization of the data-structures. The RESTORE command causes this information to be read and for the system to be reconstructed. This is quite a straightforward procedure if the system into which the status is being restored is empty, i.e. if no images exist prior to restoration. Since this may not always be the case and we may decide to restore a previously-saved system and integrate it with the one we are currently working with, some mechanism must be adopted for resolving the conflict between identically-named pyramids and images. *VIS à VIS* offers three options:

- The existing system is deleted and the new system is restored.
- Conflicts in naming are resolved by allowing thre restored image/pyramid to overwrite the existing system.
- Conflicts in naming are resolved by renaming the restored image/pyramid.

2.2.2 The Menu-based User Interface.

Dialogue between a user and the Virtual Image System (*VIS à 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 used for frequently-used menu options (e.g. Transfer an image, Window modification, display system Status, display Help text, revert to Previous menu, Quit and invoked by hitting T, W, S, H, P, and Q respectively). The mnemonic options are available on every menu.

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 re-invoked 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 à 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.

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.

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 à 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.

2.2.3 *VIS à VIS* Interpretive Control Language - VISICL.

The Virtual Image System Interpretive Control Language (or VISICL for short) was originally conceived and designed as a means to effect an interface between a low-level vision environment (i.e. an early version of *VIS à VIS*) and a high-level rule-based image understanding environment running on a remote workstation. Having realized this goal, VISICL has subsequently been developed into an invaluable tool for the investigation of complex visual processes, ranging from tasks such as the computation of depth from camera motion to 3-D object recognition. This usefulness springs primarily from the convenience of being able to program a complex sequence of image manipulations and to use these programs as procedures in subsequent analysis, once they are stable and robust. Moreover, VISICL supports parallel execution, allowing any computer vision system to control instantiations of *VIS à VIS* (through VISICL). This facility is employed within VISICL itself in order to allow multiple copies of *VIS à VIS* to be executed in parallel on a multi-transputer system, facilitating the execution of complex operations in a structured and efficient fashion. VISICL and the coarse-grained paradigm for parallel processing are dealt with in detail in chapters 10 and 11, respectively. Suffice it for the present to note that VISICL provides all of the most common structured programming constructs of conventional

(simple) programming languages together with primitives for each of the processing and analysis functions of *VIS à VIS*. The full syntax of VISICL is included at the end of chapter 10.

2.3 The processing fuctions of *VIS à VIS*.

The remainder of the book is devoted to the processing that *VIS à VIS* performs on visual data, in general, and on the data-structures just described, in particular. These include, as one would expect, the detection and analysis of intensity discontinuities, the measurement of stereo disparity and optical flow, the computation of depth, the generation of raw and full primal sketches, and the construction and recognition of 3-D objects.

Let us now proceed to look at each visual process in *VIS à VIS* a little more closely.

Bibliography

[1] Faugeras, O. D. 1983. 'Conversion algorithms between 3D Shape Representations' in"Fundamentals in Computer Vision" - edited by O. D. Faugeras. pp. 305-314. Cambridge University Press. range data.

[2] Faugeras, O.D. and Herbert, M. 1986. 'The representation, recognition and location of 3-D objects', International Journal of Robotics Research, Vol. 5, No. 3. pp.27-52.

[3] Flanagan, M. 1991. 'Generation of Grouped Three-Dimensional Raw Primal Sketches'. M.Sc. thesis, Department of Computer Science, Trinity College, Dublin, Ireland, 1991.

[4] Henderson, T. 1983. 'Efficient 3-D Object Representations for Industrial Vision Systems', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-5, No. 6, pp. 609-618

[5] Roberts, L.G. 1965. Machine Perception of Three-Dimensional Solids.', Chapter 9 in Optical and Electro-optical information processing (edited by J.T. Teppett *et al.* , MIT Press, Cambridge, MA. pp.159-197.

[6] Sandini, G. and Vernon. D. 1987. 'Tools for Integration of Perceptual Data', in *ESPRIT '86: Results and Achievements* Directorate General XIII, Commission of the European Communities (Eds.), Elsevier Science Publishers B.V. (North Holland), pp.855–865.

[7] Sandini, G., Tistarelli, M., and Vernon, D. 1988 'A Pyramid Based Environment for the Development of Computer Vision Applications', IEEE International Workshop on Intelligent Robots and Systems, Tokyo.

[8] Vernon, D. and Sandini, G. 1988. 'VIS: A Virtual Image System for Image Understanding', Software — Practice and Experience, Vol. 18, No. 5, pp. 395–414.

[9] Vernon, D. and Tistarelli, M. 1991. 'Using Camera Motion to Estimate Range for Robotic Parts Manipulation', IEEE Transaction on Robotics and Automation, Vol. 6, No. 5, pp. 509–521.

HORWOOD SERIES IN
CIAL INTELLIGENCE
Editor: Professor JOHN CAMPBELL, Department of Computer Science, University
e London

LLEL COMPUTER VISION
IS a VIS System
ERNON, Department of Computer Science, Trinity College Dublin, Ireland, and
or G. SANDINI, DIST, University of Genoa, Italy

ing a computer vision environment which supports coarse granularity parallelism on
ter-based systems, this book discusses a system which treats the organization of the
lata and its representation and interdependencies equally with the visual processes
perate on that data. As an environment, the system – VIS a VIS – provides dynamic
of images and attendant data structures, each of which are linked to provide an
representation of the interdependence of the visual cues. As a vision system, VIS a
rs all the required functionality of 3-D image understanding, from image acquisition,
detection and analysis of intensity discontinuities, computation of depth from stereo
nera motion, segmentation using raw and full primal sketches, through to 3-D model
ction and object recognition. Both geometric and functional parallelism are achieved
iting both the in-built vision control language – with its facility for remote execution
edures – and the integrated system data-structures.

ese issues are dealt with in considerable detail, and the book's emphasis on system
ation integration of information and computer architectures, rather than purely
cal and algorithmic issues, will make it a very welcome addition to the literature.

ip: Computer vision scientists, Industrial research and development engineers,
er scientists, Information technologists.

nors's interests stretch from robot vision to the philosophy of science. His recent research
have been concerned with the development of robot systems which display true autonomy. His
ompasses computational theories of perception, self-organization, and intelligence. Dr Vernon
w of Trinity College Dublin, Ireland.
ndini has been working on research in computer and biological vision for more than twelve
s primary interests at present being the integration of many low-level visual cues in a robust
tional framework. He has also been very active in exploiting the neurobiological principles of
on to postulate and implement artificial visual sensing at the 'retinal' level of computer vision.
Sandini is currently at the University of Genoa in Italy.

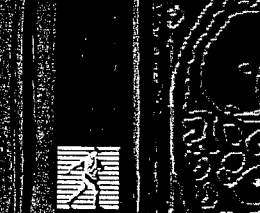
related interest

SPECIFICATION OF ADVANCED AI ARCHITECTURES
CRAIG, Department of Computer Science, University of Warwick

AL INTELLIGENCE AND INTELLIGENT SYSTEMS
ications
NDERSON, Division of Computer Science, Teesside Polytechnic

AL INTELLIGENCE TERMINOLOGY
nce Guide
Editor: COLIN BEARDON, Department of Computer Science, University of Exeter

ISBN 0-13-932716-9



HORWOOD

Vernon and Sandini PARALLEL COMPUTER VISION the VIS a VIS system

ELLIS HORWOOD

ELLIS HORWOOD SERIES IN ARTIFICIAL INTELLIGENCE

PARALLEL COMPUTER VISION
the VIS a VIS system

D. Vernon and G. Sandini