AI-2 VISION NOTES 1987/88

J A M Howe

D.A.I. TEACHING PAPER NO. 6

C O N T E N T S

These notes supersede Occasional Papers Nos. 20, 31, 35 & 50.

Revised Edition February, 1988.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

Introduction to Machine Vision

Definition: Relationships to Image Processing, Pattern Classification and Scene Analysis.

The Machine Vision Association of the Society of Manufacturing Engineers defines machine vision as "the use of devices for optical non-contact sensing to automatically receive and interpret an image of a real scene, in order to obtain information and/or control machines or processes". The references to 'automatic interpretation' and to 'real scenes' are critical parts of the definition since they help to distinguish machine vision from three closely allied fields

* Image Processing
* Pattern Classification
* Scene Analysis

each of which has contributed useful techniques.

Image Processing (often referred to as image enhancement) is concerned with the generation of new images from existing images. The techniques used, mainly derived from linear systems theory, are concerned with noise reduction, deblurring and edge enhancement. The end result is still an image, for interpretation by a human observer.

Pattern Classification is concerned with the identification of objects by means of their features, such as total object image area, perimeter length, ratio of perimeter squared to area. The value obtained for a particular feature becomes the co-ordinate of a feature point in a feature space, a multi-dimensional space in which there is one dimension for each feature measured. An unknown object is identified by comparing the distances between its feature points and those of various object types. The most likely identification is given by the smallest distance. A simple example is given as Figure 1.

The approach as explained assumes that sample feature combinations from images of real objects form uniform, multi-dimensional Gaussian clusters around the ideal points in feature space. Unfortunately, analysis is not straightforward; for example, features may interact producing elliptical distributions, clusters may not look Gaussian, so much more mathematically complex classification methods have been developed (see Pattern Classification and Scene Analysis : Duda and Hart).

Finally, the starting point for **Scene Analysis** is a low level symbolic description, such as a line representation of plane faced objects. The objective here is to transform the low level description into a higher level description that is in a more useful form for the task in hand. This might contain information about spatial relationships between bodies, their shapes and their identities.

Application Areas

There are three broad areas of application for machine vision systems in industry:

* **Inspection.** This is the most important area. In particular, automated inspection is frequently the only practical way of checking products to the standards imposed by product liability law which, in some countries, imposes penalties on manufacturers whose products are found to be defective, without any need for proof of negligence on the user's part. Tasks include not only visual examination for defects, but also measurement of dimensions, counting items, checking of orientation, and so on.

* **Guidance.** This is probably the most rapidly growing application area. It includes control of manipulators and unmanned vehicles, and so on, in parts handling, sorting and transport. The vision requirements are quite different from those of inspection.

* **Recognition.** In the industrial environment where parts delivery is highly organised there is relatively little need for systems which recognise parts from their shapes. However, with increasing emphasis on automated assembly of small batches of objects, vision systems which can quickly recognise unsorted parts will become more and more important.

In 1983, the size of the North American market for machine vision systems was $30m. Today, it is around $150m, and is estimated to rise to approximately $850m in 1990. This rapid rise in practical applications is almost wholly attributable to the steeply declining cost of computing power. First, machine vision is greedy for computer memory. Typically, an image is 512 x 512 picture elements, with each element representing one of 256 grey levels (i.e. each pixel requires one byte of memory), leading to a need for a quarter of a Megabyte. Second, machine vision techniques are greedy for processing cycles. For example, an edge detector may require a thousand computer operations per picture element, implying more than 250 million operations on a single image. A moderately powerful machine, capable of executing a million operations per second, would take about five minutes to apply such a filter to an image. In an effort to produce machine vision systems capable of processing images in real time, much time and effort has been devoted to the construction of special purpose parallel processors. For those based on cellular arrays, as Figure 2 shows, there has been an increase in processing speed of a thousand fold every ten years. However, these devices are still expensive. By and large, practical applications have been restricted to the simplest possible tasks. The paradigmatic situation is that of a single object, presented against a high-contrast background, with lighting controlled to eliminate shadows, highlights or other features that would make analysis difficult. An object is recognised by extracting features from a 2-D image, and by matching these features against 2-D object descriptions stored in memory. This limits the system's recognition to known objects observed from standard viewpoints. Many tasks naturally fit these constraints or can be readily engineered to do so, using special lighting arrangements and other artifices. Some examples include picking parts off a moving conveyor belt, bonding leads to semi-conductor chips and inspecting bottles for misaligned labels. Case studies of other applications are given in Automated Visual Inspection: Batchelor et al., pp 479-534; Machine Vision Sourcebook: Braggins et al., pp 287-343.

So, in summary, the limitations of current industrial vision systems can be stated as a set of requirements:

High contrast between object and background
No shadows
Single objects (no overlaps, i.e. no occlusion)
2-D object descriptions
Rigid objects
Standard viewpoint

Some examples of tasks beyond current limitations include:

Bin picking
Recognition of parts suspended from an overhead conveyor
Recognition of non-rigid objects
Fault detection
Robot vehicles

Bin picking, for example, is hard because parts of objects in a jumble have low contrast, and occlude each other, making it difficult to isolate separate parts in an image. Inspecting parts on a finished assembly is difficult for the same reason, and is less amenable to engineering simplification, such as dumping the contents of a bin on a surface to separate the individual parts. Swinging parts on an overhead conveyor are not constrained to maintain a standard viewpoint. Non-rigid objects can assume a continuum of configurations and thus do not lend themselves to characterisation in terms of fixed 2-D prototypes. Similarly, it is impractical to model in detail the appearance of all conceivable flaws (dents, scratches, blemishes, and so forth) in a fault inspection task. An archetypical example of a class of tasks that are inherently difficult to sustain are those involving robot vehicles in outdoor environments, such as construction site clearing, forestry and underwater maintenance of oil well equipment.

Content of course

We will begin by considering techniques for image capture, including lighting and viewing techniques. Next, we will consider the problem of characterising binary images of two-dimensional objects to achieve identification, using both connectivity and boundary tracking techniques. Thereafter, we will consider techniques for extracting edge points from the grey level representation and for combining them to produce an edge representation of the object(s) in a scene. This leads to the problem of segmentation: splitting up the edge representation into separate bodies, i.e. scene analysis. Our final task will be to consider the problem of generating and using 3-D models to identify instances of objects, beginning with Roberts' classical approach.

REFERENCES

Batchelor, B.G., Hill, D.A. and Hodgson, D.C., 1985. Automated Visual Inspection (Eds). Bedford, UK : IFS Publications.

Braggins, D. and Hollingum, J., 1986. Machine Vision Sourcebook. Bedford UK : IFS Publications.

Duda, R. and Hart, P.E., 1973. Pattern Classification and Scene Analysis. New York : Wiley.

Nevatia, R., 1982. Machine Perception. New Jersey : Prentice Hall.

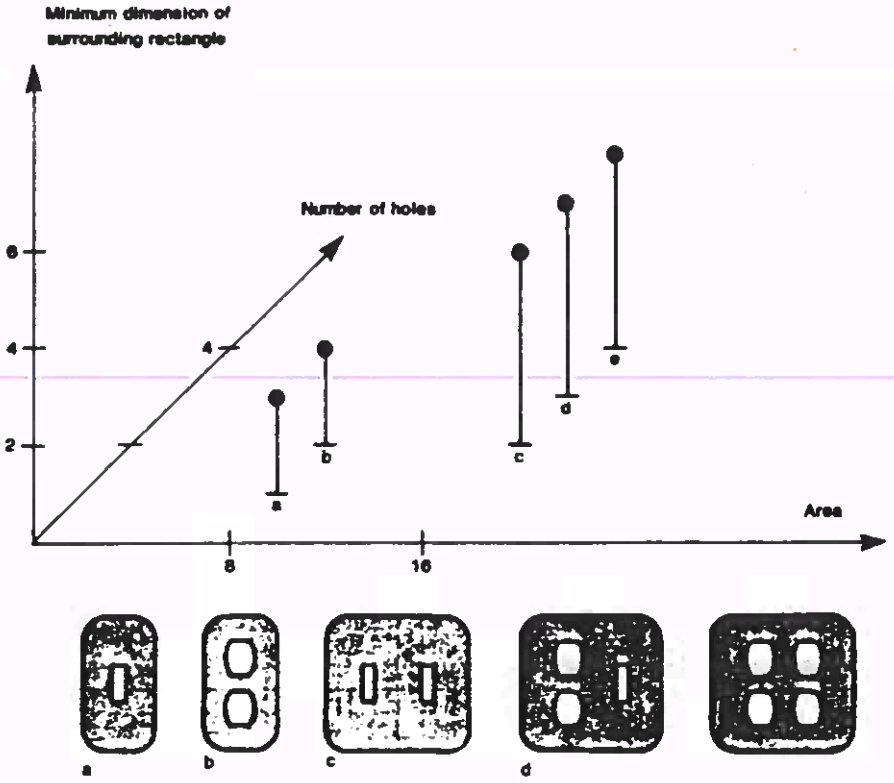


Figure 1 A feature space. An unknown object is identified according to the distances between the unknown and the models.

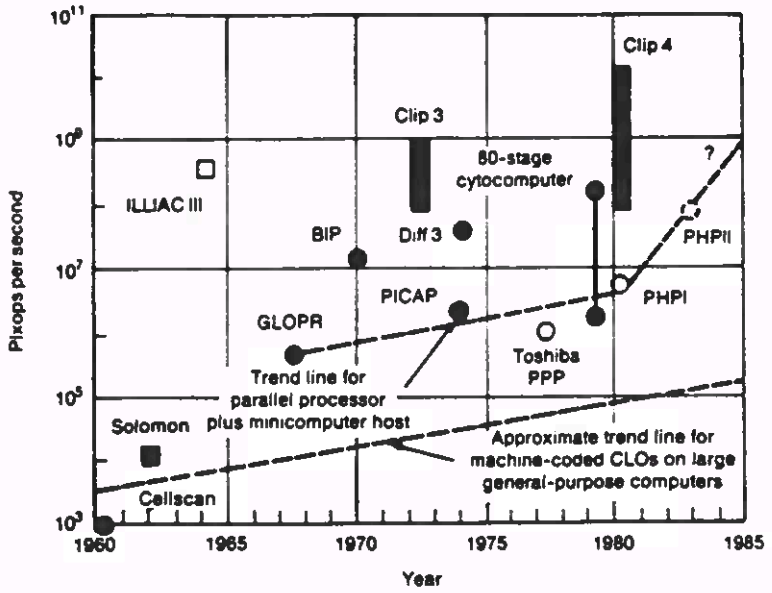


Figure 2 Operating rate of cellular logic machines versus time (from Preston [39]).

Source: K. Preston, "Cellular Logic Computers for Pattern Recognition," IEEE Computer, Vol. 16, No. 1, January 1983, pp. 36–50. © 1983 IEEE. Used by permission.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

Lighting and Viewing

In any machine vision problem the quality of the lighting is extremely important. Frequently, through poor choice of methods, key features are obscured by glare or the light intensity at the detector is too low.

Illumination equipment

This can be considered in two broad categories,

(i) lamps
(ii) equipment for regulating, manipulating and directing light from these lamps.

Types of lamps, their characteristics and uses are summarised in Table 1 (from Batchelor et al., 1985). Equipment for matching a lamp to the system's requirements are listed in Table 2 (also from Batchelor et al., 1985).

In choosing a lamp, one has to take account of the:

* wavelength required
* intensity required
* area to be illuminated
* reflectivity of the object
* transmission efficiency of image acquisition system
* space available.

Two of the most important factors are spectral distribution and shadowing. Besides visible light, most of the lamps in Table 1 radiate considerable amounts of infra red. Each has its own characteristic continuous spectral distribution: some of these have one or more spikes while others have near Gaussian distributions. Also, the tungsten filament lamps tend to project shadows of their filaments on to the viewing surfaces. Finally, while fluorescent lamps do not generate shadows, their light is not uniform along their length.

Factors which have to be taken into account in choosing devices for manipulating the light include

* Defects
* Transmission efficiency
* Filter characteristics
* Diffuser characteristics
* Condenser characteristics
* Reflector characteristics
* Intensity requirements

Finally, some typical illumination problems and possible solutions are given in Table 3 (from Batchelor et al., 1985).

Optics

In most machine vision applications, an object's image will be formed on the sensor by a lens, where the sensor is typically a television camera. The sensor's size, its resolving power and the system length determine the initial optics design. Since the majority of sensors have a resolution limit expressible in terms of a number of picture elements per picture height (or width), normally a magnification value is selected that matches the detail in the image to the sensor's resolution. Having decided upon the magnification required and the system's length, the focal length of the lens is calculated by means of the lens formula

$$\frac{1}{f} = \frac{1}{u} + \frac{1}{v}$$

$$m = \frac{v}{u}$$

where **f** is the focal length, **u** is the object-to-lens distance, **v** is the image-to-image lens distance, and **m** is the magnification.

Some examples of the use of the simple lens are given in Figure 1 (from Batchelor et al., 1985). Particularly when working with images of 3-D objects, the depth of focus is imparant. The lens should be chosen such that the objects are within the distance defined by the depth of focus divided by the square of the magnification of the system. This is known as the depth of field. Since u and v have a non-linear relationship, the depth of field is not symmetrically disposed about the object plane (as shown in Figure 1b).

Relationship between resolution and contrast

The image quality of a lens is specified as the relationship between resolution and contrast, where contrast is defined as the difference between the intensities in light and dark areas of the image detail normalised by the difference between white and solid black areas. By using an increasingly fine pattern of black and white stripes (a grating) as test object, the image of the stripes will gradually become grey making them difficult to distinguish. This is due to lens defects called aberrations which blur the image. Under monochromatic light, typical types of aberration include:

* **Spherical aberration.** Different focus points for light rays passing through the lens centrally and off-centre.

* **Coma.** Comet-shaped images of point objects produced by oblique, off-centre light rays.

* **Astigmatism.** Different parts of image in focus at different distances from lens.

* **Field of curvature.** Variation in focus across image when a flat sensor is used since the surface of best focus is domed.

If white light is used, chromatic aberrations also occur:

* **Longitudinal chromatic aberration.** Colour fringes appear on each image plane, since light of different colour comes to a focus at different distances from the lens.

* **Transverse chromatic aberration.** Oblique rays of different colours strike the image plane at different points, causing differences of magnification.

The above problems can be minimised by using a narrow colour band of light and stopping down the lens so that only its central portion is used (provided sufficient light is available).

Examples of lighting and viewing techniques

Sixty three methods of lighting and viewing scenes are described by Batchelor (Batchelor et al., 1985). Some of the most useful are:

Method 1 - to provide uniform, omni-directional illumination (see Figure 2);

Method 2 - to view silhouettes (see Figure 3);

Method 3 - to acquire 3-D shape information (see Figure 4).

Relationship between scene and image

Since we will be dealing either with 2-D silhouette images or with 2-D images of polyhedral objects, the relationship between the scene and the image is relatively straightforward. To further simplify the analysis, we will start by considering scenes containing single objects.

The purpose of the lighting and viewing techniques described above is to cause light reflected from the scene to be focussed on to the sensitive surface of a photo transducer where it forms patches of light and dark. It is the transitions in luminance that carry information about the features of the objects in the scene, i.e. the bordering edge in the case of a 2-D silhouette image, and the external and internal edges in the case of 2-D images of polyhedral objects.

In the case of incoherent illumination of laminar objects, the intensity profile of a light-to-dark transition in an image formed by a perfect lens is shown in Figure 5 (from Batchelor et al., 1985). Notice that the true position of the edge is at the 50% intensity point, and that this is independent of the degree of blur. However, the actual position may vary from this point due to the effects of electronic noise, lens aberrations, and so on.

The intensity profile across an edge of a laminar object is somewhat different when the light is coherent, as shown in Figure 6 (from Batchelor et al., 1985) where the edge is located at 25% full intensity. In this case, the edge is sharper than an edge generated by incoherent light.

Note that thick edges cause further degradation of the ideal edge intensity case, making it impossible to analyse except in simple cases. Figure 7 (also from Batchelor et al., 1985) illustrates the intensity profile for a perfect thick edge, and shows how focusing can remove spurious reflections. However, the majority of thick edges are unlikely to be perfect, so the intensity profile will be significantly degraded.

Ideally, one would like to examine each transition in image luminance and classify the type of edge producing that transition. What the above analysis shows is the difficulty of doing this due to degradation introduced by factors outside the user's control.

Also, the luminance transitions might not necessarily specify the edges of surfaces of objects in the scene at all. Instead, they might be produced by highlights, shadows, texture gradients across surfaces, dirty surfaces, surface markings such as scratches or printed letters on a box. In other words, the information in the luminance distribution is not specific to its source, i.e. we cannot determine unambiguously which property of the physical world is represented by a given luminance distribution. Most of the time, much of this information will act as extrinsic noise. In other words, the information will not be relevant to the visual system's immediate purpose: it will merely mislead and distract it, and lengthen processing time. For example, printed letters on the surface of a tin are "noise" when the perceiving system's goal is to extract contour information; but when it is trying to identify the contents of the tin, the noise-versus-information relationship is reversed.

In conclusion, no matter now much care is taken in setting up lighting and viewing conditions, and matching them to the characteristics of the sensor (see next note), noise in images is a fact of life in the real world. It is usually not possible to remove all of it, though some can be got rid of by judicious pre-processing techniques which will be encountered in the next note.

REFERENCES

Batchelor, B.G., Hill, D.A. and Hodgson, D.C., 1985. Automated Visual Inspection (Eds). Bedford, UK : IFS Publications.

Table 1 Types of lamps – characteristics and uses

Types of lamp	Wattage and dimensions	Characteristics	Envelope shapes	Other applications	Variations	Inspection uses
Tungsten filament*	0.25–1000 W 2.5–180 mm dia Beam angle 10–70°	Non-uniform point source Flux shape primarily depends on filament shape Moderate efficiency Narrow beam with integral lens	Cylindrical – subminiature and miniature Bulbous – medium and large wattages	Instrument panels Fibre-optic sensors Vehicles Domestic lighting Interior display spotlights	Lens end – miniature types Clear, opal and coloured envelopes Bowl reflector Rough service	Miniatures surrounding a camea lens Borescopes Microscopes Occasionally light boxes Low intensity illumination
					Short-wave infrared	Penetration of infrared translucent materials
Quartz halogen*	10–1500 W 44–300 mm long	Non-uniform point and linear sources Envelope temperature 350°C + Moderate efficiency	Cylindrical – short and long lengths	Vehicle headlights Interior display Cine projectors Photography Photocopiers Floodlighting Theatre stages	Smooth and multifacet dichroic reflectors Integral lens Transverse and longitudinal filaments	Borescopes Microscopes Fibre-optic light sources Profile projectors High-intensity illumination Penetration of semi-translucent materials
Fluorescent* (low pressure discharge)	4–125 W 300–2400 mm long	Even linear distribution Low intensity Low temperature High efficiency Long life	Cylindrical – long, straight and circular tubes	Office Shop Hotel Interior display Domestic	Circular with diameters from 151–340 mm	Light boxes Large long object illumination Peripheral illumination Circular objects
	6–40 W 225–1200 mm long 125 W bulbous	Long-wave ultraviolet	Cylindrical – medium Bulbous – small and large wattages	Forgeries Food contamination Mineralogy Gemmology	Integral wood glass filter	Crack detection with dye penetrant – low intensity

Table 1 continued

Types of lamp	Wattage and dimensions	Characteristics	Envelope shapes	Other applications	Variations	Inspection uses
Arc (discharge)						
High pressure mercury	100 W ultraviolet 20 mm dia. 100 mm long	Short bright arc of 'black light' peaking at 366 nm	Cylindrical – medium length	Laboratories	Other wattages not applicable	Fibre-optic light source with light diameter 2 mm max.
	250 W	Intense visible arc. Both need ballasts	Cylindrical – medium length			Special microscopes
Xenon	150 W	High intensity Instant response 3 mm long arc Needs control gear Emits uv and short-wave infrared Resembles daylight	Cylindrical – medium length	Flashtubes Strobes Beacons 2000 W in lighthouses	Other wattages not applicable	Special microscopes
LED † (light-emitting diode)	20–40 mA 2–15 mm dia. 2–3 × 6 mm rectangle	Fairly even beam of low intensity Long life	Button Rectangle	Instrument illumination Opto-electronic and fibre-optic sensors	Infrared and visible red, green, yellow	Small, thin transparent objects with sensitive cameras

* May be dimmed

† Although they are not lamps, they are used for illumination

Table 2 Operations which may be performed to make a lamp compatible with other parts of an inspection system

Operation	Description	Component/attachment
Absorbing	Retaining light of certain wavelengths so that only desired wavelengths reach the object	Filters (Note, differential absorption effects image acquisition)
Bifurcation	Dividing light from a single source into two branches which may be aligned at the user's discretion	Twin arm light guides Swan (goose) necks with or without focusing lenses, filters and polarising caps
Branching	The multiple version of bifurcation	Multibranch light guides
Chopping	Strobing or breaking the beam into short bursts	Shutters Rotating discs
Collimating	Producing a cylindrical or rectangular 'column' of light	Plano convex lens
Collecting	Gathering light from a lamp and focusing it on a smaller area	Spherical or cylindrical lens – usually biconvex, aspheric or plano convex
Condensing	Collecting the radiation from a source and removing unwanted elements, e.g. shadow	Aspheric or biconvex lens
Diffusing	Spreading out light from a source evenly over an area	Diffusers (see page 50)
Interfering	Preventing the passage of light other than that of selected wavelengths	Filters
Polarising	Eliminating all light waves except those in one plane	Polarising filter
Realigning	Rerouting light from a source to an object with the facility to negotiate obstacles and align at the user's discretion	Light guides (see Table 4.5)
Redirecting	Rerouting light from a source in single steps at predetermined angles dependent on the component used	Mirrors Reflectors Lenses Plastic, glass quartz rod
Smoothing	Removing the intensity ripple effect associated with an electricity supply	dc supply High frequency ac
Splitting	Dividing a beam of light into two paths, e.g. transmission and reflection at predetermined angles of bifurcation	Beam splitters
Spreading	See Diffusing	
Stabilising	Compensating for electrical, chemical and physical changes which cause the intensity of a lamp to vary	In-circuit photoelectric cell
Stimulating	Activating a chemical so that it emits wavelengths of higher order than those received, e.g. by a crack detection dye penetrant	Ultraviolet lamps with or without quartz fibres Filters
Transforming	Changing the shape of a source, e.g. circular to linear	Cross-section converters

Table 3 Illumination problems and solutions

Cause	Affected component/object	Problem	Possible solution
Chemical			
ageing	*Ultraviolet filters* *Plastic optical fibre*	*Ultraviolet degradation*	*Planned replacement* *Glass or uv degradation resistant fibre*
	Lamp	*Filament decomposition*	*Planned replacement* *Compensation by stabilisation*
radiation	*All components*	*Nuclear radiation causes browning of glass*	*Non-browning glass* *Consumable components/regular replacement*
Electrical			
fracture	*High-pressure lamps*	*Disruptive fracture of envelopes*	*Suitable enclosure*
fluctuations	*Lamps*	*Variations in intensity due to voltage fluctuations and lamp ageing*	*Stabilisation with in-circuit photoelectric cell*
life	*Lamps*	*Failure during inspection cycle*	*Planned replacement* *Automatic changeover to standby lamp*
Environmental			
dirt	*All components*	*Reduction in intensity* *Changes in optical characteristics, e.g. glare spots*	*Continuous purge by good quality air* *Regular cleaning*
	Object	*Superimposition of false image, e.g. stains and micro shadows*	*Cleaning* *Homogenous light*
gases (explosive)	*Electrical components*	*Hydrocarbons in atmosphere could be ignited by electrical spark*	*Intrinsically safe light sources* *Remote illumination with fibre optics*
(poisonous)	*Certain lamps*	*Ozone*	*Suitable enclosure*
inaccessibility	*Object*	*Insufficient space for lamp to illuminate object adequately*	*Fibre optics, mirrors, prisms, lenses*
'missiles'	*All components* *Object*	*Parts being manufactured escaping from handling systems*	*Protective materials* *High impact resistance materials, e.g. polycarbonate*
Mechanical			
bending	*Light guides* *Transparent rods*	*Overbending increases attenuation and may cause fracture*	*Observation of design limit, i.e. bending radius = 10× diameter approximately* *Suitable outer jacket*
flexing	*Light guides*	*Repeated bending causes eventual fracture*	*Avoid flexing or planned replacement within design limit, i.e. 500,000 flexes*

Table 3 continued

Cause	Affected component/object	Problem	Possible solution
tension	*Light guides*	*Sustaining tensile stresses beyond capacity of assembled optical fibres*	*Suitable jacket*
Optical			
aberration (chromatic)	*Object and lenses*	*Coloured rings on periphery of illumination especially if incorrectly focused*	*Correct focusing* *Lens large in size to be compatible with system* *Achromats*
contrast	*Object*	*Features merge into background, i.e. lack of resolution*	*Realignment of illumination system* *More intensity* *Coloured and polarised filters*
distribution	*Lamp and object*	*Shadow superimposed on object*	*Diffusers* *Condensers* *Change lamp burning position*
homogeneity	*Lamps* *Fibre-optic attachments*	*Uneven distribution (similar to distribution problem above)*	
light scatter	*Object*	*Transparent objects may collect light so that features become indistinguishable*	*Realignment of illumination system* *Diffusers* *Matt black background*
reflectivity – general (excess)	*Processing equipment and object*	*Saturation makes image indistinguishable due to brightness*	*Rheostats* *Diaphragms* *ND filters* *Realignment of illumination system*
(inadequacy)	*Processing equipment and object*	*Features indistinguishable due to darkness*	*Increase illumination* *Use more sensitive camera*
reflectivity – local (excess)	*Processing equipment and object*	*Glinting*	*Focused and redirected light* *Realignment of illumination system* *Filters* *Diffusers* *Removal of dust*
translucency (excess)	*Object*	*(See light scatter)*	
(inadequacy)	*Object*	*Internal features and defects near surface invisible*	*Increased visible intensity* *Try short wave infrared*

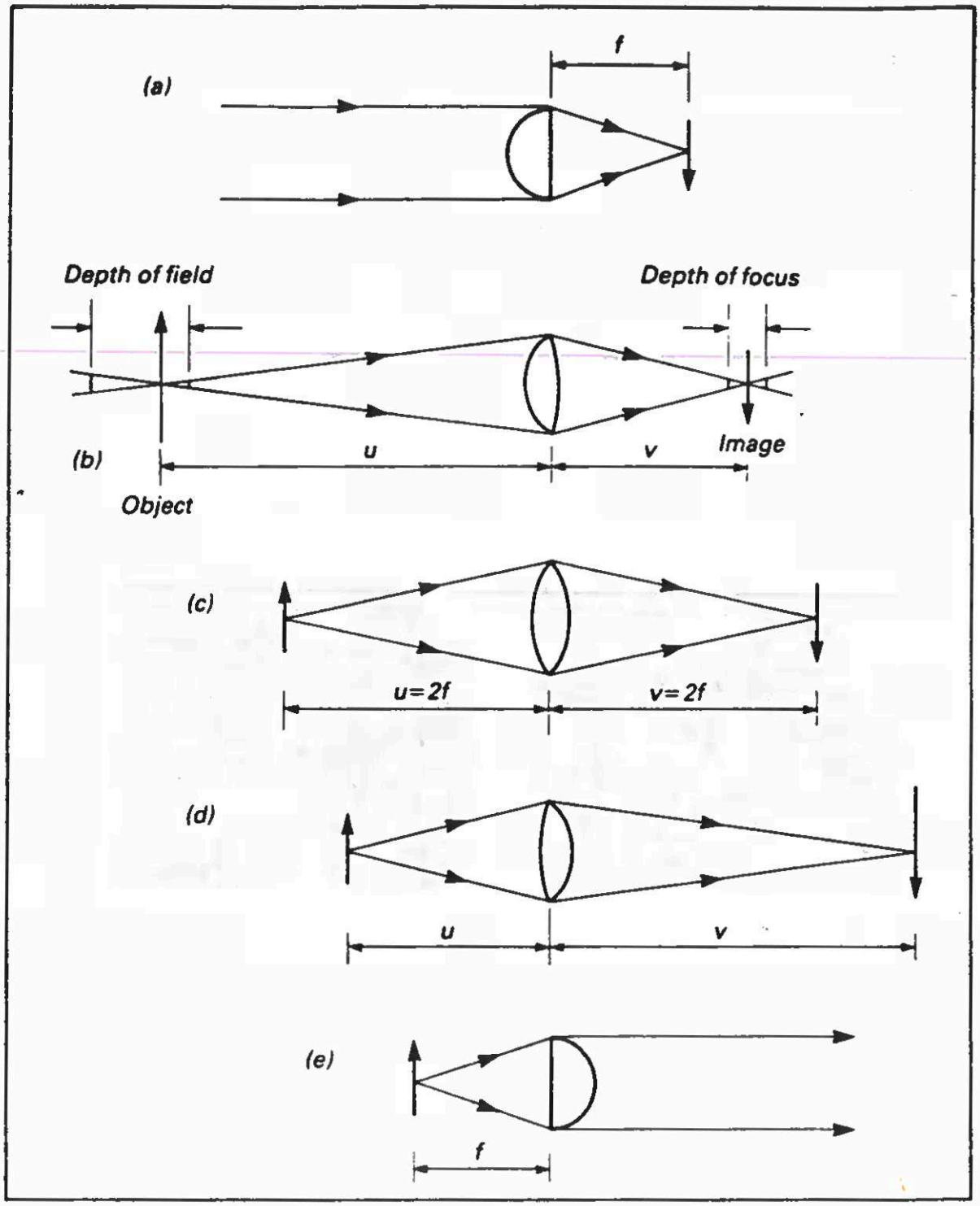


Fig. 1 Image forming basics: (a) image of distant object, (b) demagnification, (c) unit magnification, (d) magnification, and (e) collimation of light source or distant image forming

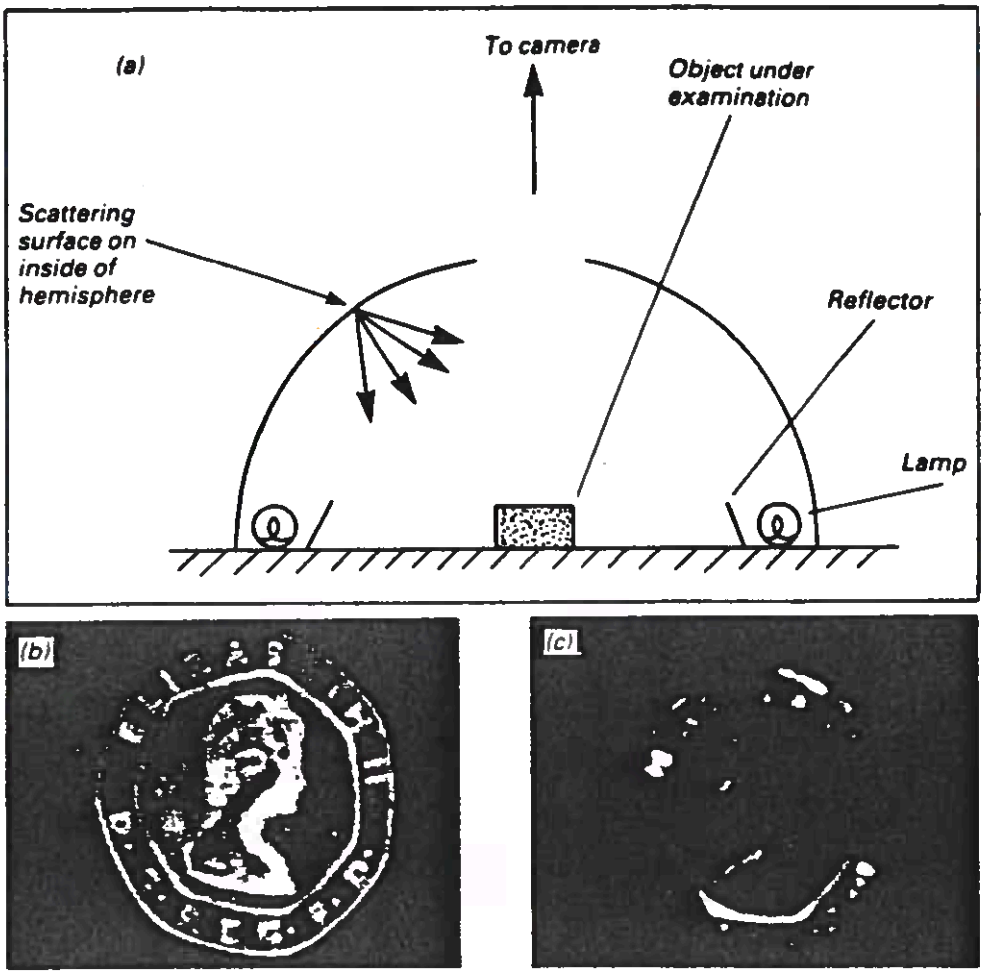


Fig. 1. Omni-directional lighting: (a) cross-section through the hemispherical diffuser, (b) view of a bright coin in omni-directional light, and (c) the same coin in ordinary light

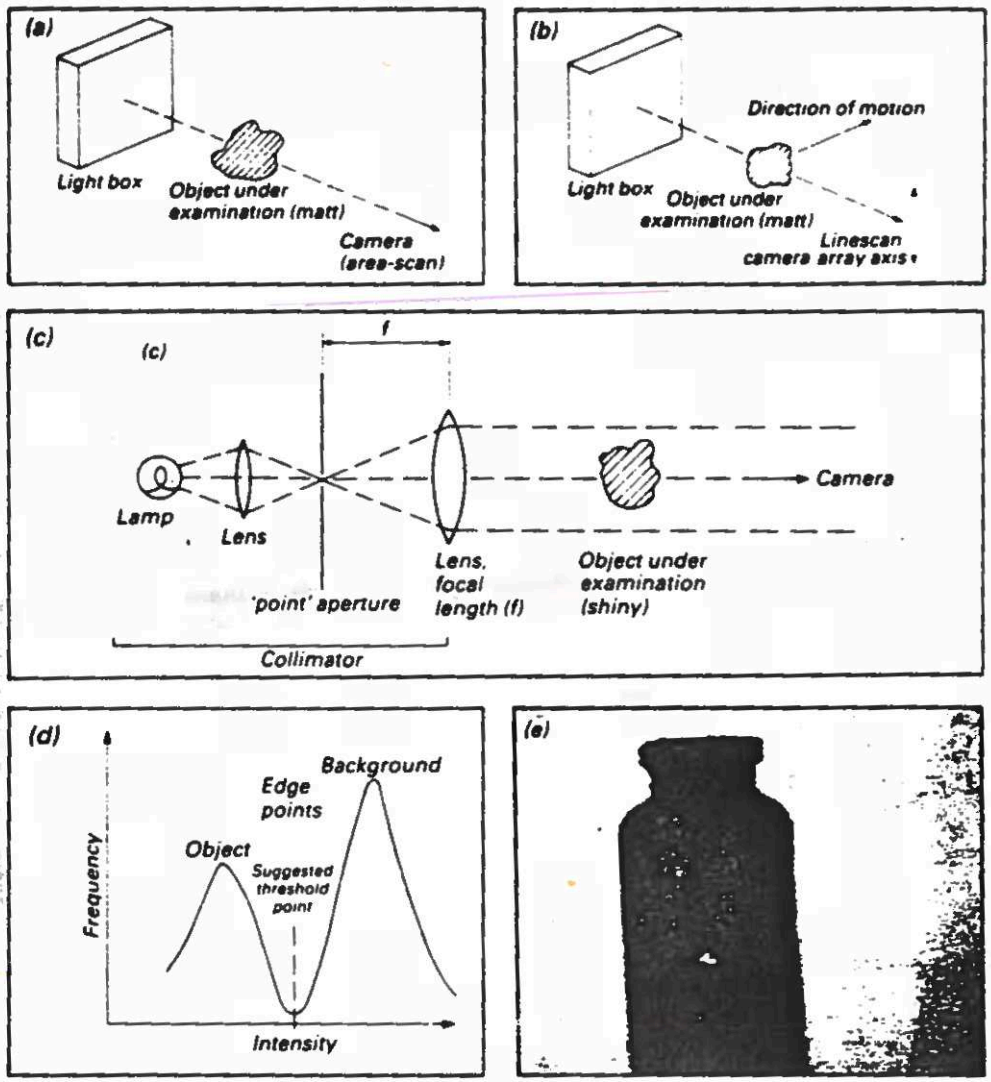


Fig. 3 Viewing silhouettes: (a) simple technique suitable only for matt objects and using an area-scan camera, (b) viewing matt objects using a linescan camera, (c) viewing shiny objects such as polished metal, (d) typical histogram of a back-illuminated opaque object (thresholding can be achieved successfully by using a parameter derived from the position of the 'valley' in the histogram), and (e) silhouette of a glass vial

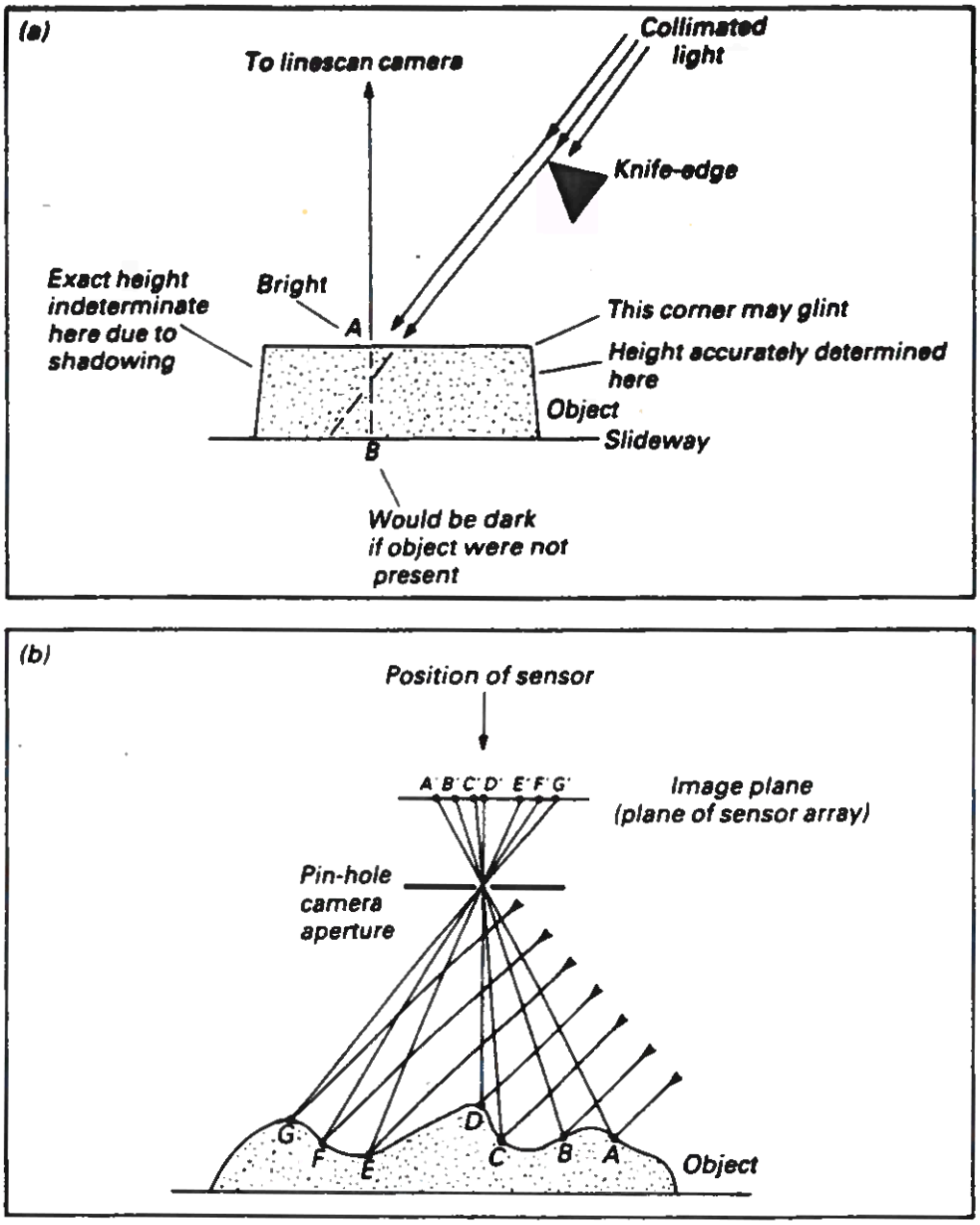
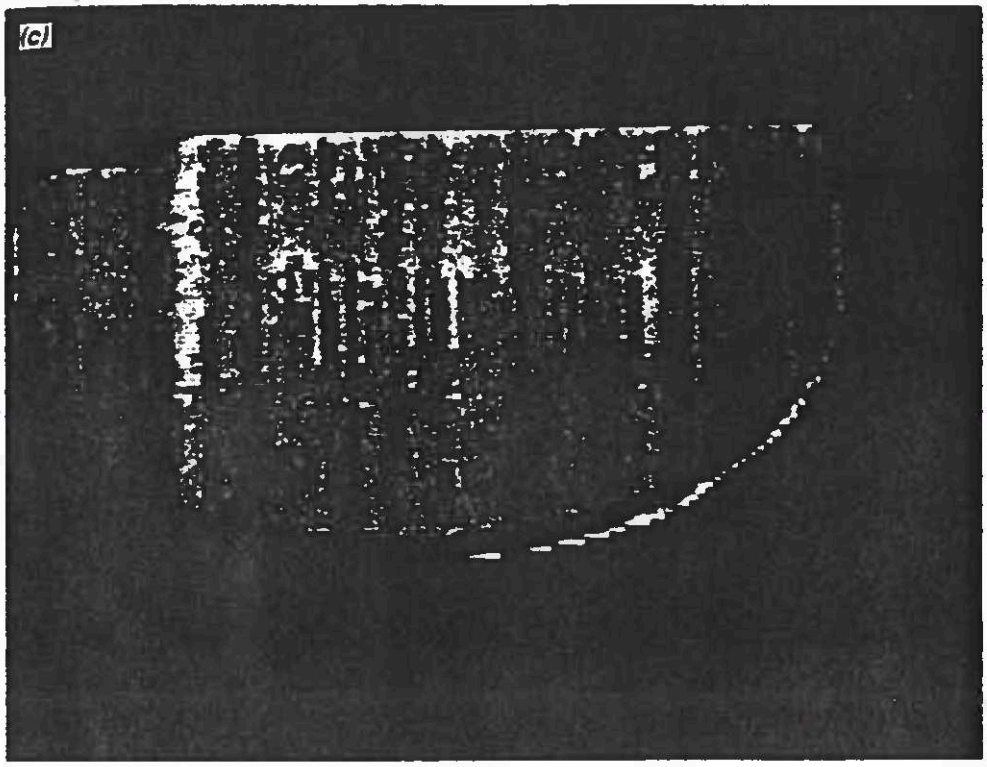

(a)
To linescan camera
Collimated light
Knife-edge
Exact height indeterminate here due to shadowing
Bright
A
This corner may glint
Height accurately determined here
Object
Slideway
B
Would be dark if object were not present
(b)
Position of sensor
A' B' C' D' E' F' G'
Image plane (plane of sensor array)
Pin-hole camera aperture
G
F
E
D
C
B
A
Object

Fig. 4



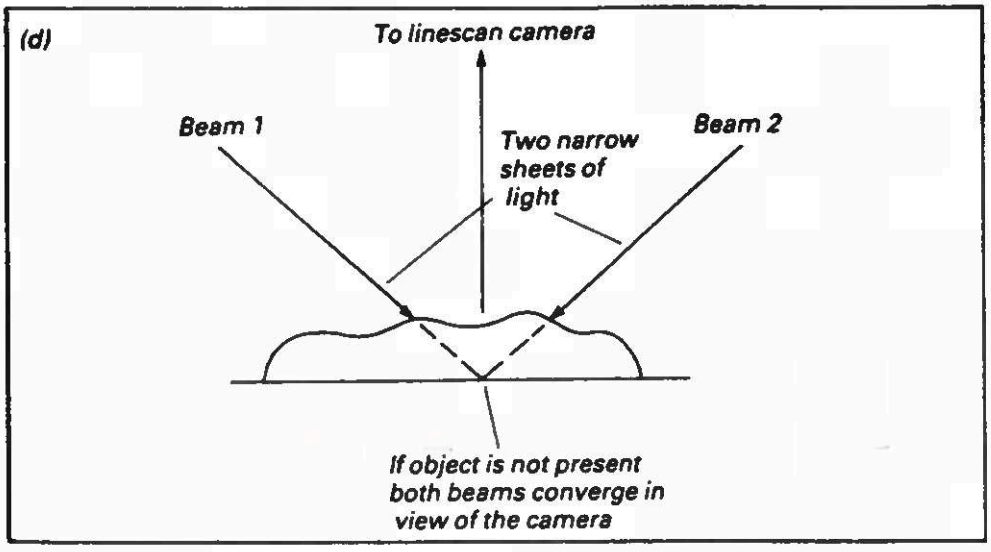


Fig. 4 Structured light, binary method: (a) optical arrangement (single light–dark edge), (b) ray geometry (cross-section through the linescan camera and knife edge), (c) height map of the component in (b) – it is a simple matter to threshold such an image, and (d) twin-beam method avoiding the shadowing problem (cross-sectional view)

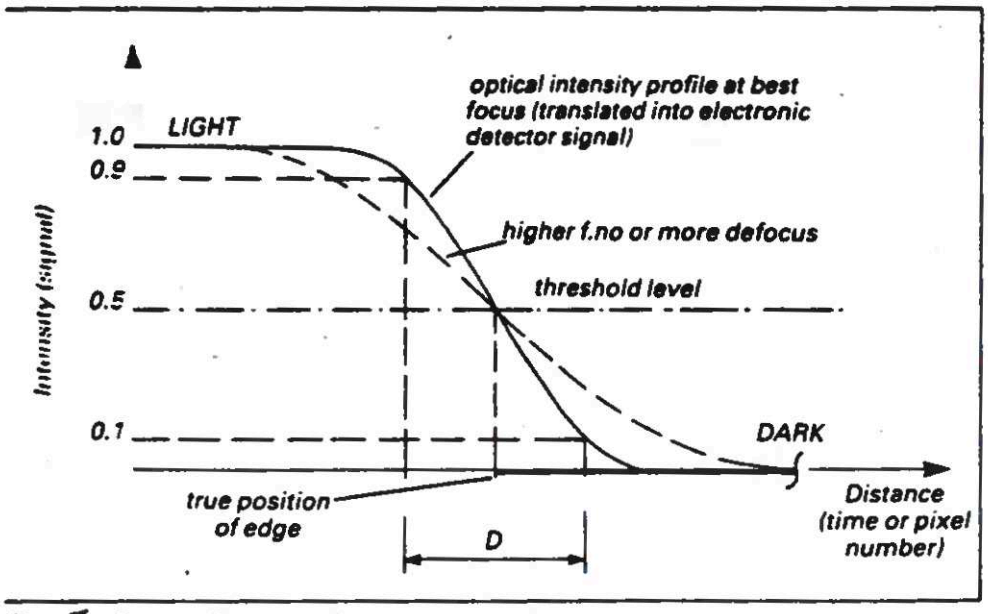


Fig 5 Edge definition and measurement – incoherent illumination

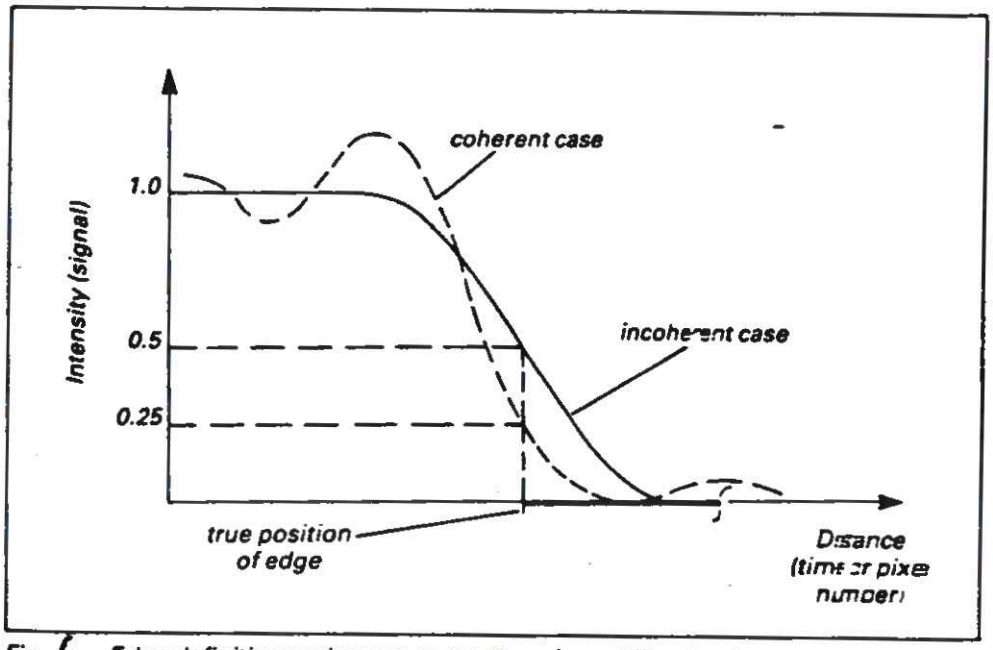


Fig 6 Edge definition and measurement – coherent illumination

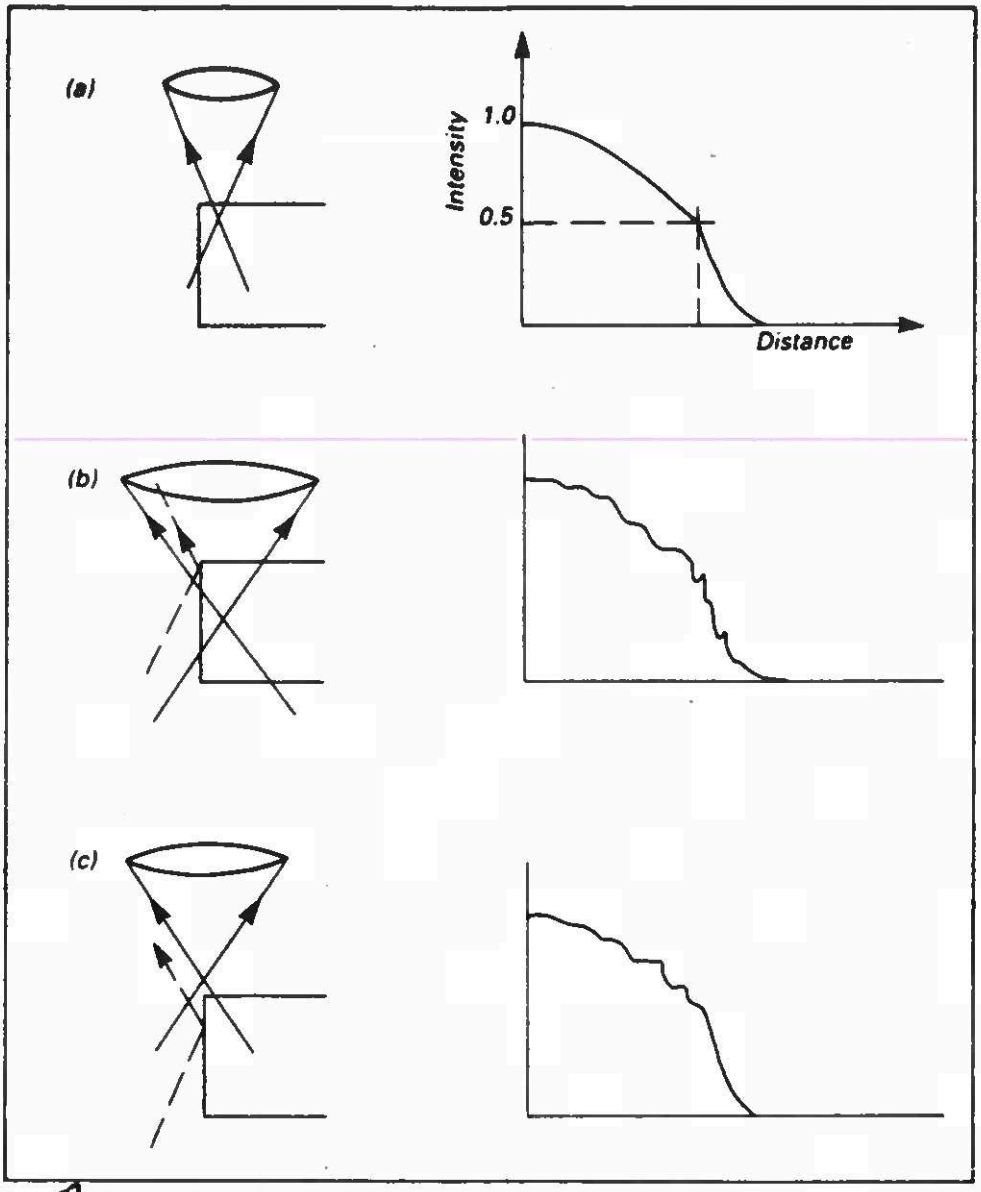


Fig. 7 Intensity profiles for a perfect thick edge: (a) profile for a perfect thick edge, (b) the effect of spurious reflections, and (c) elimination of spurious reflections from the dark half of the image

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

Capturing the Image : Building the Gray Level Description

The first step in gathering information is to convert the light from the scene into useful electrical signals. This is done by scanning the image, and digiting the resulting data to produce a data structure called the gray level description which contains numerical values, each of which represents the luminance at a defined position in the scene (or its image) (see Figure 1).

Two operations must be performed to obtain this data:-

1. The positions of the locations must be defined, and a method for visiting these locations must be devised. This is called scanning.

2. The image intensity must be measured at each location, and expressed as a numerical value. This is called quantization

Scanning

There are two general approaches to scanning. First, a focussed beam of light, such as a laser beam, is moved across the scene in a controlled manner, and the light reflected from the scene is recorded. This is known as the flying spot scanner approach. It assumes that there is no other light source competing with the moving beam (see example of welding system below). The second approach is to control which spot is looked at through a moving peephole or aperture, and to record what is seen through the aperture at each instant. This is the flying aperture technique.

Two methods which fall into the former category are the drum scanner and the laser scanner. In the case of the drum scanner, a photographic print of a scene is fastened to the surface of the drum which is rotated discontinuously. While the drum is stationary, a moving carriage containing

a light source and a photo-diode detector traverses across it. The X-Y position values are obtained from shaft encoders linked to the drum and carriage respectively (see Figure 2).

At the heart of the laser scanner is a rotating mirror which deflects the beam in one or two dimensions. Reflected light is picked up by a sensitive detector. Again, X and Y position values are obtained from encoders attached to the mirror drive mechanism (see Figure 3).

Turning now to flying aperture techniques, these encompass television systems and the more recent charge transfer devices (CTDs). While there are a variety of detector devices for generating television images, the vidicon is the one most often used in machine vision applications. In a vidicon tube, the image is focussed on a photosensitive screen and scanned with an electron beam (see Figure 4). Typically, its spectral range is from 400 to 750 nanometers, with peak sensitivity around 550 nanometers. Unfortunately, electron-beam deflection varies in a non-linear way across the screen, so the position information is inaccurate and non-repeatable. Also, the image drifts around the screen surface due to temperature and voltage fluctuations. Finally, the vidicon tube suffers from image persistence.

CTD technology offers a number of advantages over conventional tube-type cameras, including light weight, small size, low power consumption, high sensitivity, wide spectral range (from 450 to 1,000 nanometers) and lack of persistence. There are two main classes of CTD's - charge coupled devices (CCDs) and charge injection devices (CIDs). For imaging purposes, the CCD is an array of closely packed MOS capacitors forming a shift register (see Figure 5). Charges on the array are transferred to an output register either one line at a time or one frame at a time. An example of the use of a CCD array is given in Figure 6 (from Clocksin, 1985). This shows a MIG welding torch packaged with a sensor system, part of a visually guided arc welding device. In this application, infra red light is the best choice for illuminating the work site since the most significant spectral components of scattered light from the nearby arc are in the visible and ultra violet bands. With this in mind the sensor components used include a pair of rectangular CCD cameras (488 x 380 elements) fitted with narrow band optical interference filters having a spectral bandpass of 10 nanometers centred at 830 nanometers, and four infra red laser diodes emitting at 830 nanometers.

The use of infra red light also gives improved transmission through smoke and fumes produced in the course of welding. In the CID device, charges are not transferred at sensing. Instead, they are read using an X-Y addressing technique similar to that used in computer memories (see Figure 7).

Quantization

Regardless of which method is used to convert from the analogue image to final digital form, each cell (pixel) in the sampled image will assume a value between some minimum luminance value representing black (usually 1) and some maximum value representing white (minimally 16 but up to 256). This is the quantization process which is usually performed by the hardware interfacing the sensor to the computer.

A simple, image-processing system is shown in Figure 8. Like the drum scanner, it uses a photocell mounted on a moving carriage which scans the cell across the image from left-to-right and from top-to-bottom. The cell's measurements of light intensity are expressed as an electrical signal whose magnitude varies according to the value of the light intensity. This analogical signal is transformed into numbers by an analogue-to-digital convertor, the size of each number representing the light intensity at a particular 2-D position in the image, e.g. 0 for black, 15 for white. These numbers are stored in a 2-D array in the computer's memory. We can look at the contents of this array, either by printing out the numerical values or by feeding these values to a full-tone printer to produce a gray-level image in which each square represents one pixel of the stored gray-level description, and each square's gray level directly shows the intensity of the associated pixels.

Data Acquisition Problems

There are two characteristics of this image data acquisition process that give cause for concern. The first is the precision of the process. This refers to the spatial resolution of a given system (Figure 9). Both sampling and quantization determine whether or not the digital representation captures the significant features of the original scene. Although a very course digital representation contains enough information so that a person can recognize a face (see Figure 10, from Harmon, 1973), machine vision programs usually work with digital images at higher resolution.

The second characteristic of the process is its accuracy. This refers to gray scale resolution. What we are interested in here is the data degradation that might have occurred in the data acquisition process, and some measures for compensating for that degradation.

One form of degradation is intrinsic noise, in the form of isolated pixels whose gray-level values are radically different from those of their neighbours. It can be caused by defective phosphors in photosensors, or round off errors in the analog-to-digital conversion process, and so on. Spurious values could pose problems for later processing methods. High values are likely to be interpreted as feature points which a system will try to account for at some later stage in the processing (in vain!); low values are likely to be interpreted as belonging to the scene background, producing gaps in the feature data. To eliminate these spurious values, a local transformation operation, called "smoothing", can be applied. It is a local operation since it is applied at each point in an image.

Basically, smoothing operations rest on the assumption that the actual scene consists of areas that are much larger than the area represented by a single pixel. Accordingly, pixels that differ markedly from their immediate neighbours are errors that ought to be removed.

The following is a description of a simple smoothing operator:

> If any point in the picture is brighter than all of its eight immediate neighbours, its luminance value is reduced to make it the same as the brightest of its neighbours; if any point in the picture is dimmer than any of its eight immediate neighbours, its luminance value is increased to make it the same as the dimmest of its neighbours.

Notice that this operation is conservative in the sense that it removes some of the noise without reducing the amount of information in the representation. In particular, it eliminates isolated noise points, but has no effect upon noise that occupies two or more adjacent image points.

A simpler, more liberal smoothing operator that would reduce the significance of larger regions of noise is:

Replace the luminance value of each point by the average of the luminance values of its eight immediate neighbours.

Unfortunately, the application of this operator to every point in an image will have the effect that every edge will be blurred. Indeed, several successive applications of the operator would wash away the entire picture. Clearly, therefore, smoothing operators are useful, but must be carefully chosen to try to eliminate whatever kind of intrinsic noise is present in a set of digitized pictures, without also removing significant features of the pictures themselves.

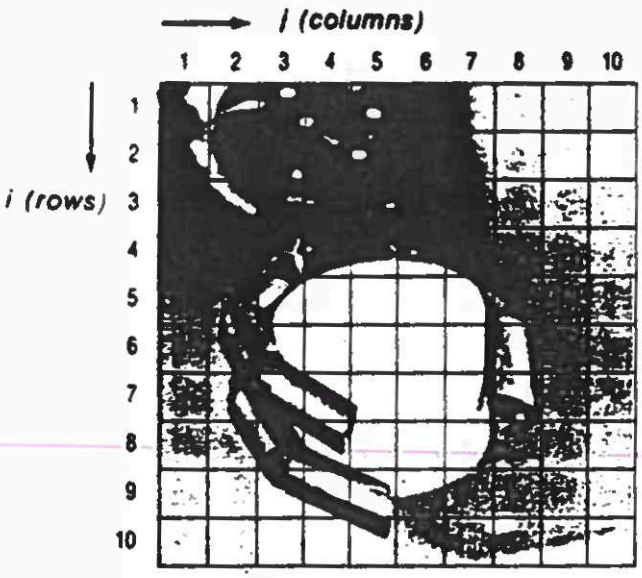
A global transformation that might improve accuracy is histogram equalization, where the histogram is a bar chart of an image, with the X-axis representing pixel intensity levels and the Y-axis the number of pixels at each level. From information theory, it can be shown that a uniform distribution of pixel values results in maximum information content in the description. In other words, a histogram with roughly equal numbers of pixels at each level is the optimum situation. Histogram equalization defines a mapping that stretches contrast (i.e. expands the range of gray levels) for gray levels near histogram maxima and compresses contrast in areas with gray levels near histogram minima (see Figure 11, from Hall, 1979). Further details of the technique can be found in Hall: Computer Image Processing and Recognition, 1979, pp 166-173.

REFERENCES

Clocksin, W.F., Bramley, J.S.E., Davey, P.G., Vidler, A.R. and Morgan, C.G. 1985. An Implementation of Model-Based Visual Feedback for Robot Arc Welding of Thin Sheet Metal. Int. J. Robotics Research, 4, 13-26.

Hall, E.G., 1979. Computer Image Processing and Recognition. New York : Academic Press.

Harman, L.D., 1973. The Recognition of Faces. Scientific American, 229, 70-83.



j (columns)

i (rows)	1	2	3	4	5	6	7	8	9	10
1	45	78	35	34	41	40	74	121	126	128
2	42	67	12	27	31	44	66	115	129	124
3	40	63	28	25	20	27	59	109	117	126
4	62	74	49	110	136	145	138	102	112	115
5	124	103	77	226	238	243	240	104	109	191
6	127	117	69	109	221	230	225	109	101	107
7	130	127	66	78	118	210	205	93	85	103
8	134	134	85	73	129	175	190	125	99	100
9	138	135	133	199	80	94	98	104	105	108
10	139	136	133	129	126	122	112	119	139	171

Figure 1. The first step of digitization is to partition the image into cells (pixels) addressed by row and column, shown (exaggerated) on the left. Within each pixel, the digitizer measures and assigns a number corresponding to the image brightness (right).

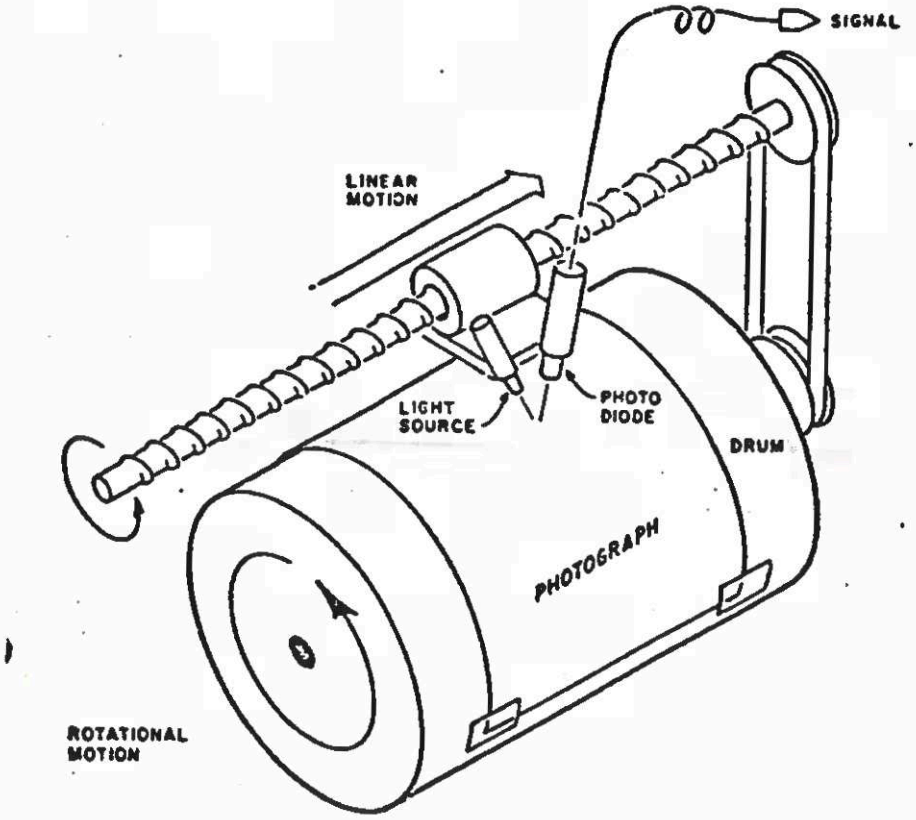
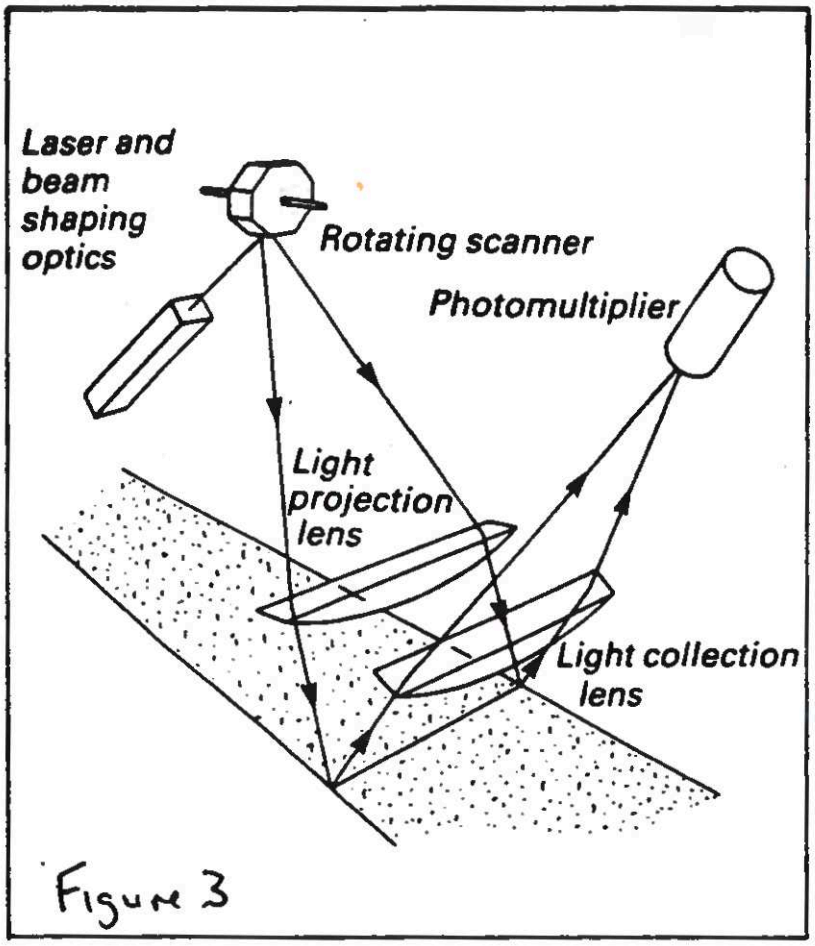


Figure 2: *A drum scanner produces high-quality results by moving the photograph relative to the sensor. Its drawbacks are that it requires precision mechanical construction, works very slowly, and the signal it produces is not video-compatible.*



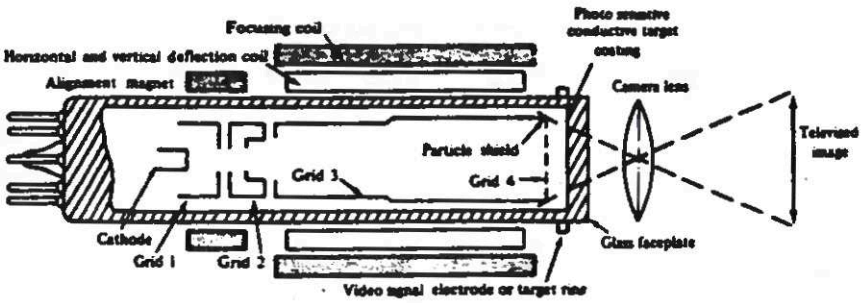


Fig. 4 The vidicon.

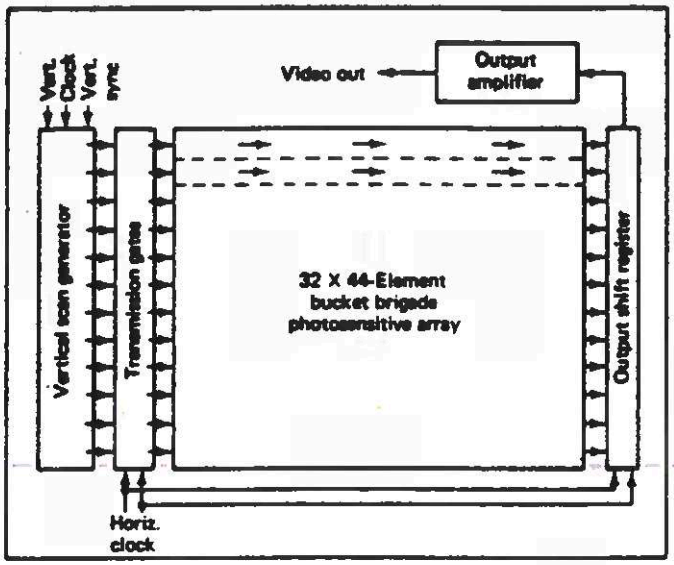
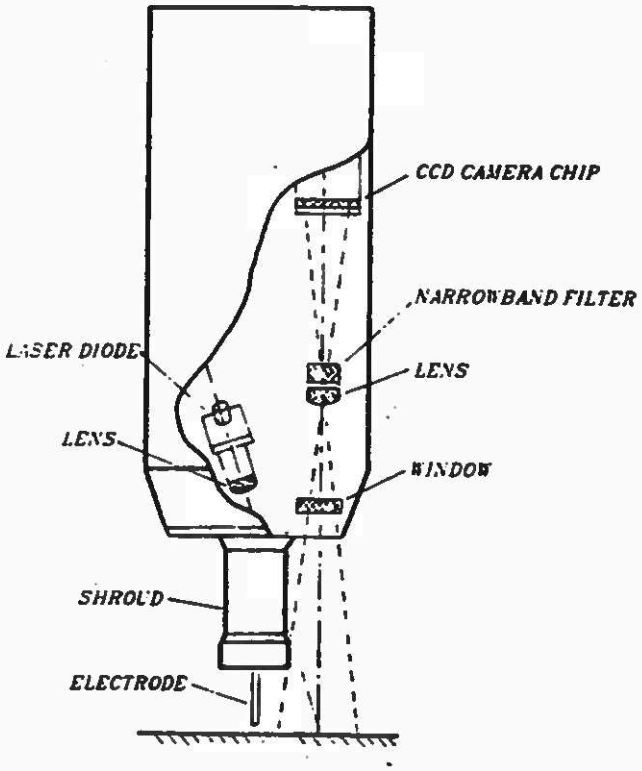


Fig. 5 A CCD array (line transfer).

Fig. 6 Cutaway sketch of the torch/sensor package showing the optoelectronic components for one camera/laser combination. Not shown is the duplicate set of components on the opposite side of the torch axis.



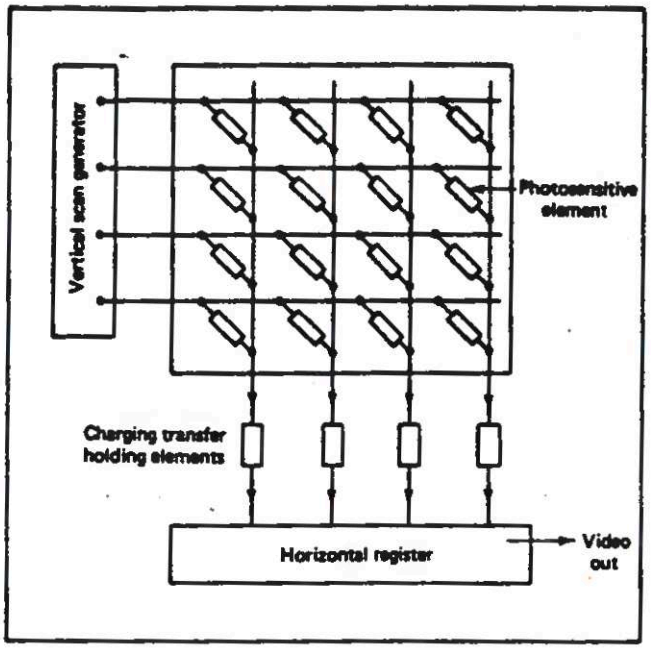
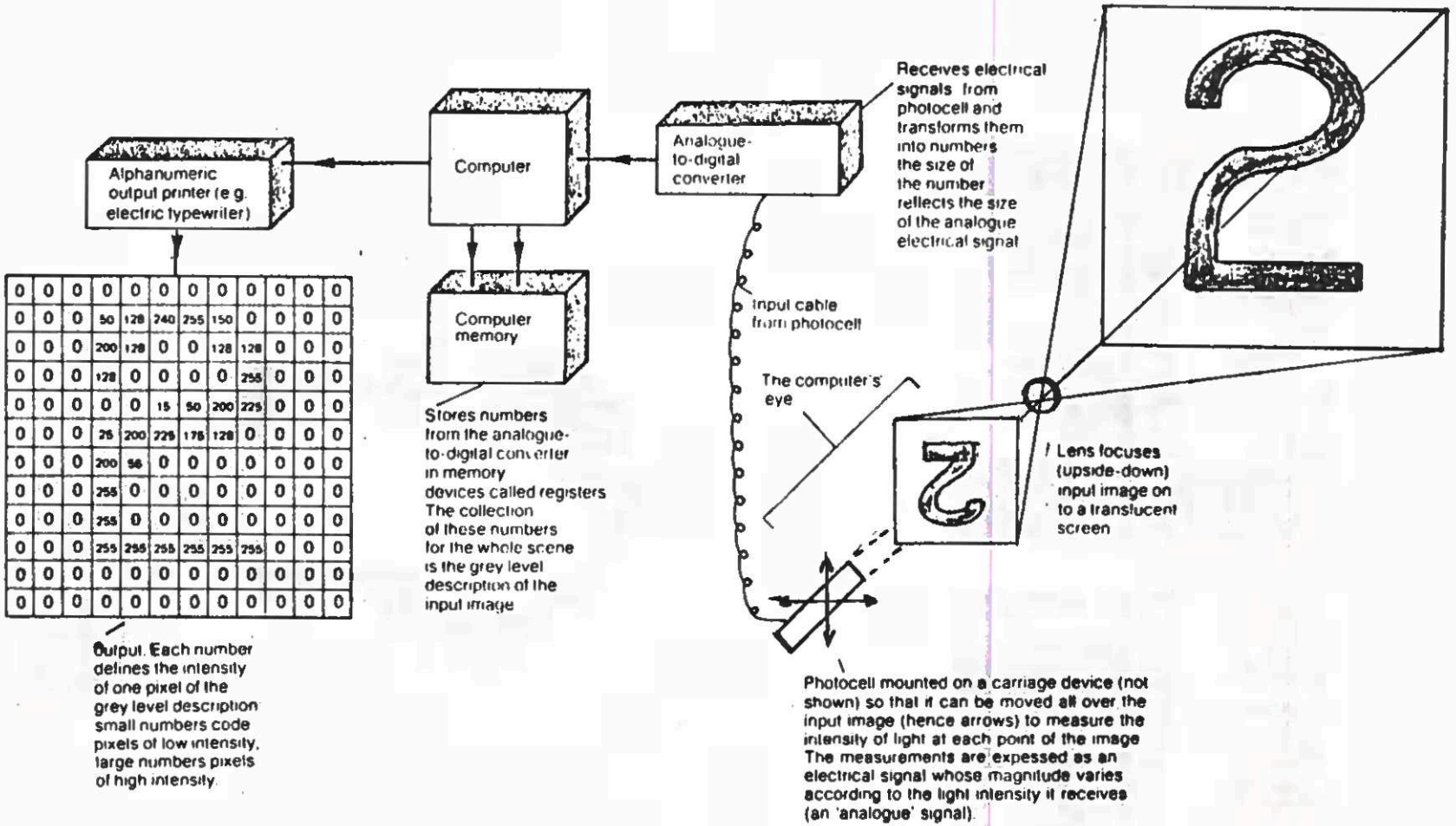
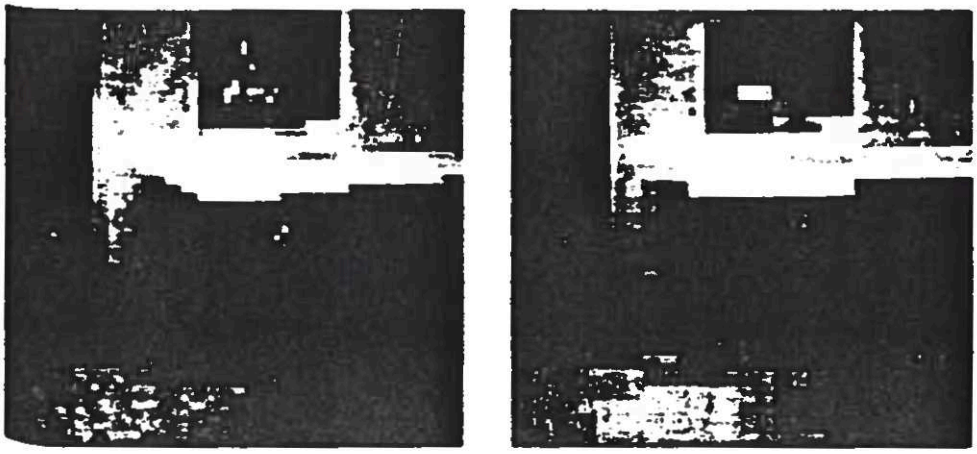


Fig. 7 A CID array.



0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	50	128	240	255	150	0	0	0	0
0	0	0	200	128	0	0	128	128	0	0	0
0	0	0	128	0	0	0	0	255	0	0	0
0	0	0	0	0	15	50	200	225	0	0	0
0	0	0	25	200	225	175	128	0	0	0	0
0	0	0	200	56	0	0	0	0	0	0	0
0	0	0	255	0	0	0	0	0	0	0	0
0	0	0	255	0	0	0	0	0	0	0	0
0	0	0	255	255	255	255	255	255	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure . 8



a *b*

Figure 9 Effect of reducing precision. (*a*) 60 by 60 resolution. (*b*) 30 by 30 resolution.

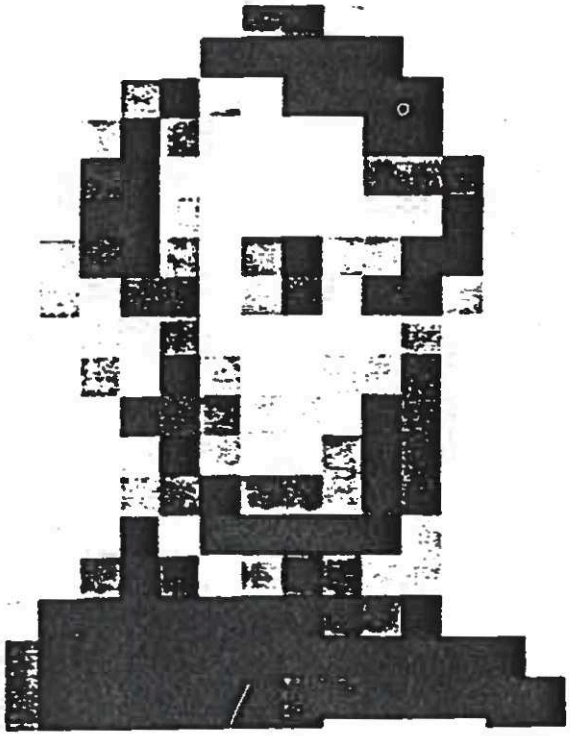


Figure 10 Low-precision picture of a familiar face. (*Leon Harmon, Bell Telephone Laboratories.*)

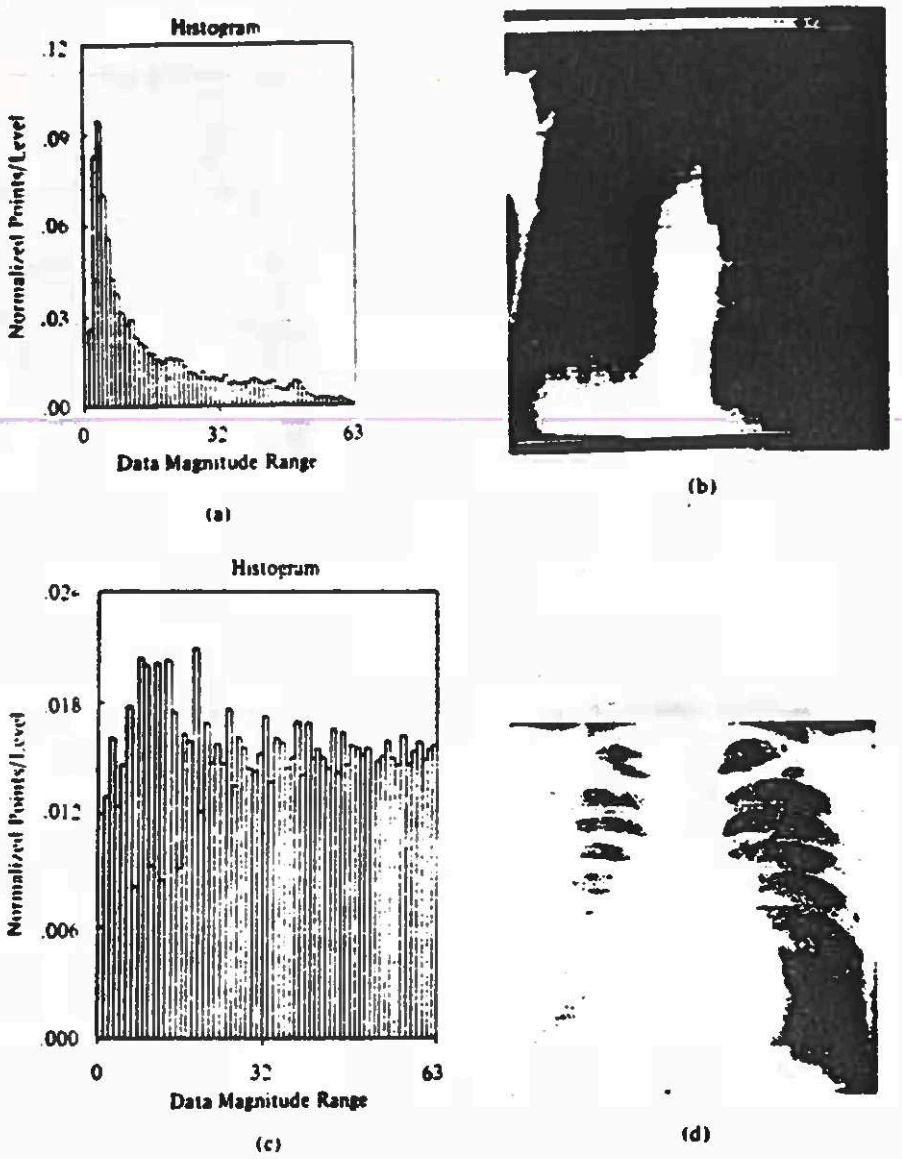


Figure 11 Pictorial example of histogram equalization (a chest x ray): (a) histogram of original chest x ray, (b) original chest x ray, (c) histogram of enhanced chest x ray, and (d) enhanced chest x ray.

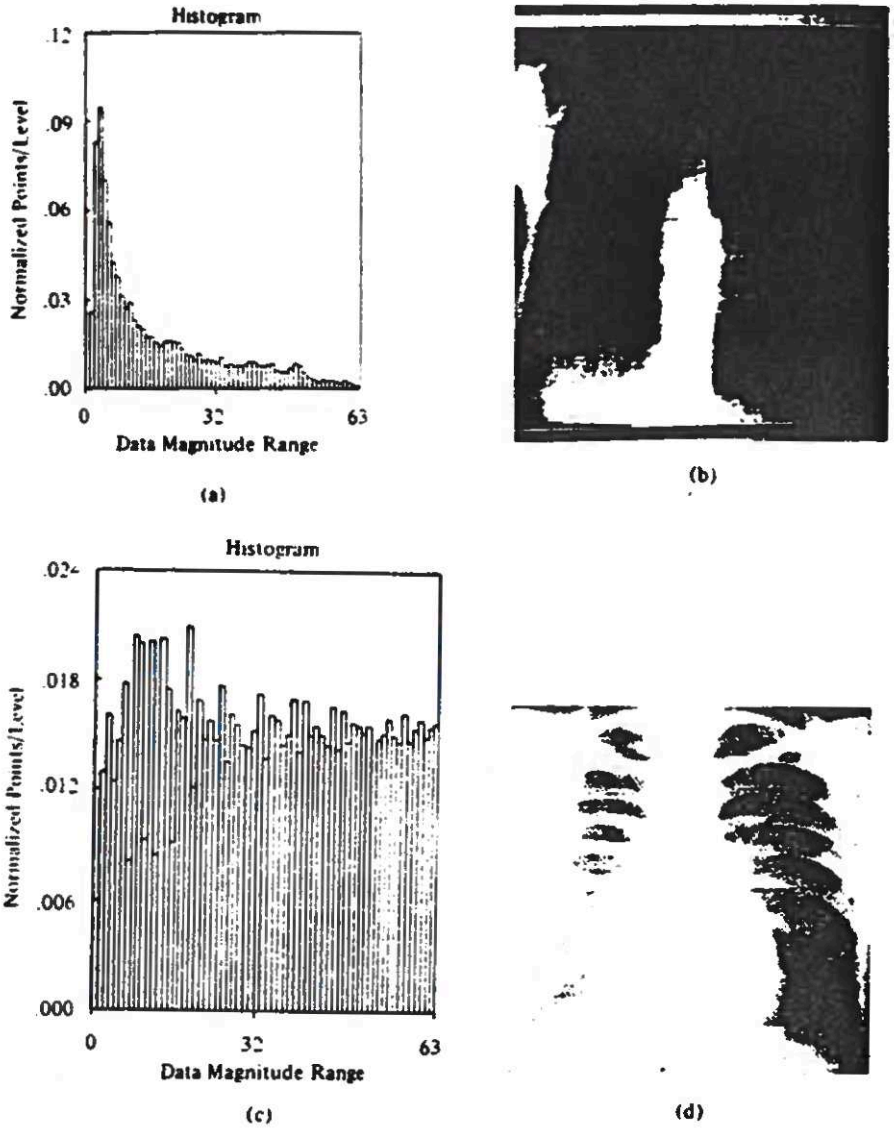


Figure 11 Pictorial example of histogram equalization (a chest x ray): (a) histogram of original chest x ray, (b) original chest x ray, (c) histogram of enhanced chest x ray, and (d) enhanced chest x ray.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

AI2 JIM HOWE

Processing Binary Images

In many industrial applications, objects can be successfully discriminated by means of shape parameters derived from 2-D binary (silhouette) images. In this note we will begin with the generation of binary representations. Thereafter we will consider the problem of extracting and interpreting shape information from the binary representation.

Building the binary image

Our starting point is the gray-level description of the scene of interest. The binary image is made by setting a threshold value. Cells in the gray-level description where the luminance value exceeds the threshold value give rise to 1s in corresponding positions in a new two dimensional array, and those below the threshold give rise to zeros. Note that it is a matter of choice whether 1 and 0 represent bright and dark cells, or dark and bright cells. An example of a binary image is given as Figure 1 (from Horn, 1986).

We have glossed over the problem of selecting the threshold value. It could be chosen by human experimentation: select a value, build a binary representation, then convert the binary representation into a binary image on a graphics screen, for examination by the human experimenter. However, an automatic method of selecting the threshold would be preferable to minimise time and cost. A popular approach is to use histogramming techniques as tools for selecting threshold values. These include the gray-level histogram (see Figure 2, from Horn, 1986) which records the numbers of picture cells at different gray-levels, and the cumulative gray-level histogram which records the number of cells at or below a given gray-level.

The prototypicel situation is an object of uniform brightness lying on a background of uniform brightness. Owing to the presence of noise, the gray level cells corresponding to the object will not have exactly the same value. The same is true of background values. In both cases, there will be a spread of grey scale values and a median value. Provided the spreads are small

enough, it should be possible to identify a value separating the two groups of gray-levels. In the histogram, this would appear as two peaks separated by a valley, where one peak corresponds to the object and the other to the background. In an ideal case, there will be a gap between the peaks, but even when there is some overlap the threshold can be set where the histogram has a minimum.

The technique suffers from three main difficulties. (1) The first difficulty with the method is deciding the width of the histogram's bins. If each gray level value is given a bin, the histogram is likely to be too flat and ragged. On the other hand, if the bin is too wide, the gray-level resolution will be too low. (2) The picture cells bordering an object introduce the next difficulty since they will have intermediate gray-levels (see discussion of edge characteristics in earlier note). Their effect is to smear and merge the skirts of the two peaks in the histogram. The magnitude of the effect depends upon the fraction of cells that fall on the boundary, where that fraction is inversely proportional to the square root of the ratio of the area of the object to the area of a picture cell. (3) The final difficulty occurs when the image area occupied by the object is much larger (or much smaller) than that occupied by the background, since the smaller peak may become submerged in the larger peak's skirt. Also, there may be no detectable minimum value for setting the threshold.

When the fraction of the area occupied by the object is known, the cumulative histogram can be used. In this case, the threshold is set at the gray level value corresponding to that fraction.

Processing the binary image

We will begin by considering a binary representation of a single object. We will refer to the group of contiguous pixels of the same colour that represent the object in the binary representation as a blob. Also, we will assume that our objective is to recognise the blob as an instance of a known object whose features have been recorded and stored as a set of parameters. The task, therefore, is to derive a feature description of the unknown blob and match that description with stored descriptions.

A variety of global feature parameters can be derived, depending on the method used to describe the shape. They include simple measures such as area, perimeter length, area to perimeter ratio, compactness (ratio of perimeter to square root of area), and more complex measures, such as maximum and minimum dimensions, for which area, position and orientation data are required. The latter information is usually obtained by forming a series expansion of an exact representation of the shape, and then using the first few terms of the series. The co-efficients of these terms constitute the description of the shape. But more of this later.

We will start by considering two methods of finding blobs in the binary description.

* connectivity analysis
* boundary tracking

Connectivity analysis is concerned with the local connectivity around each pixel in the binary description: blobs are built up on a pixel-by-pixel basis using run-length coding. **Boundary tracking** on the other hand infers the connectivity of pixels by following the boundary of a blob to determine its outline. If the object has hole(s) in it, or if more than one object is represented by the binary description, the former method is more efficient since all the processing is done in a single raster scan whereas the boundary tracker alternates between boundary following which requires random access to the image and raster scan search for new blobs. The main advantage of the second method is that it produces a chain-code description of the boundary of each blob. As we shall see later, a number of complex shape parameters can be derived from a chain-code description.

Definition of connectivity

These two methods share a common problem, that of the definition of connectivity. Adjacent pixels are connected if they are of the same colour. But which pixels should be regarded as being adjacent to the one of immediate interest? In a rectangular grid, the usual connectivity relationships are 4-connectivity and 8-connectivity. These alternatives are shown in Figure 3 (a) and (b). Unfortunately, neither of these methods is entirely satisfactory. Consider the binary description of a closed curve, given in Figure 4. If we apply 4-connectivity, the result will be six separate blobs, two made up of zeros (background and central blobs) and four made up

from pairs of adjacent 1s! If, instead, we apply 8-connectivity, the result will be two blobs, one comprising the background and the central areas (0s) and the other the continuous curve (1s). The problem here is that linking the central area of the background is a breach of the Jordan curve theorem which states that a simple closed curve should separate the image into two simply connected regions. For the purposes of automated recognition, the central area is a feature of the object, not a part of the background.

One solution which is often adopted is to use 6-connectivity, as shown in Figure 3(c). However, this also fails when applied to Figure 4, since it will yield three blobs, one made from 0s (background and central area) and two made from 1s (two halves of the closed curve). Another solution to the problem is to apply 4-connectivity to one colour and 8-connectivity to the other colour. If these are background and object colours respectively, the required connectivity relationships can be obtained.

Connectivity analysis

Connectivity analysis is achieved by combining one of the above methods with run-length coding. This latter method exploits the fact that along any particular scan line there will usually be long runs of zeros or ones. Individual bits can be replaced by numbers indicating the length of such runs. Typically, each run of consecutive 0s or 1s on a line is encoded as a three word record where the first word encodes the starting position (in x dimension) of the run, the second the number of pixels in the run, and the third the name of the blob that these pixels belong to. Each line of the binary description can be described by a list of these run-length records (see Figure 5).

The connectivity analysis algorithm scans the images from left to right, top to bottom, updating the descriptions of each blob which intersects the current scan line. At the end of each run of 0s and 1s, the run-length list is updated and blob statistics are computed. If any pixel of the run just completed is connected to a pixel of the same colour on the previous line, an existing blob is extended to include this run and its statistics are updated. Otherwise a new blob record is allocated.

A blob may have more than one name associated with it (see Figure 6). The two blobs 1 and 2 will not be combined to form a single blob until the scan reaches line A. At that point, the statistics for blobs 1 and 2 are combined. The nesting relationship between the larger and smaller blobs in Figure 6 can also be described. In this case, the second blob might be characterised as CHILD of the first blob since it is wholly enclosed within the first blob. Such relational descriptions are useful, and will be discussed in more detail in a later note.

Value of run code method

The value of the run code method is the ease with which a limited set of measurements can be generated. Whereas area is easy to calculate (by adding run lengths), centre of mass and perimeter are hard to calculate. Accordingly, Batchelor (1985) recommends that the method should be used for objects that can be characterized along one dimension. He gives the example of a bottle, as shown in Figure 7. This is described quite adequately in terms of:

- simple diameter measurements (e.g. comparing values z lines from top with y lines from bottom)
- maximum/minimum diameter
- straightness
- uprightness
- minimum-area rectangle enclosing shape.

The run code method described above is restricted because the measurements are derived from the row data, ignoring column data. Recently, Horn has suggested a method for calculating the column values from the projection of the first differences of the rows of the binary image, as shown in Figure 8 (from Horn, 1986). This enables first and second moments to be calculated by summing I, J, I^2, IJ and J^2. From these totals, position and orientation can be calculated (see below).

Boundary tracking

A similar procedure is followed in the case of boundary tracking. The essential difference is that a tracking algorithm traverses the boundary of each blob in the image, starting from the first boundary point detected in the raster scan of the binary representation. When the tracking algorithm returns to its starting point, the raster scan is resumed, and so on.

The boundary tracker operates as follows. First, each pixel is viewed as a unit square, bounded by four edges. A boundary pixel is one which has one or more edges in common with a pixel of a different colour. The boundary of a blob is a sequence of directed boundary segments (composed of boundary pixels) identified by means of a chain coding scheme. There are two chain coding schemes, one with four directions and the other with eight directions, as shown in Figure 9. The boundary traversal is carried out in a clockwise direction with respect to the interior of the blob. At each step, the tracking procedure has to decide whether to turn to the left, or to the right or to continue straight ahead. In the case of four direction chain coding, this is determined by examining the two pixels that lie ahead and on either side of the current boundary segment. The decision rules are shown in Figure 10 (for 4-connectivity: rules for 6- and 8-connectivity are more complex).

Value of chain code

Blob statistics are obtained as the boundary of the shape is traversed. At each point, partial sums can be calculated for the perimeter i.e. the length of the chain. For 4-connectivity, this is a simple calculation; for 8-connectivity, links coded as 1,3,5 or 7 add $\sqrt{2}$ to the length of the perimeter. Chain height and chain width (i.e. the height and width of a box drawn round the shape so that the outline touches on all four sides) can also be easily calculated by taking differences between maximum and minimum co-ordinate values in both dimensions.

However, the real advantage of chain coding is that a variety of analytical measures of a shape's area, position and orientation can be derived from the boundary description by generating its moments. While we can generate an infinite series of moments, the low order moments are the most robust. In particular, the first six moments can be used to obtain:

* m_{00} - number of points making up the shape, i.e. area

* $\frac{m_{10}}{m_{00}}$ - x value of centroid of shape

* $\frac{m_{01}}{m_{00}}$ - y value of centroid of shape

* $\frac{1}{2}\tan^{-1}\left[\frac{2(m_{00}m_{11} - m_{10}m_{01})}{(m_{00}m_{20} - m^2_{10}) - (m_{00}m_{02} - m^2_{01})}\right]$ - orientation of shape's major axis

Moments of odd degree, such as m_{01} and m_{10}, provide information about the balance of gray levels between the left and right, or upper and lower, half planes; moments of even degree, such as m_{20} and m_{02}, provide information about the spread of gray levels away from the y or x axis. The major axis is the line through the centroid about which the spread of gray levels is least.

In the context of boundary tracking (using 4-connectivity) moments are calculated according to the following formulae:

$$m_{00} \text{ (Area)} = P_1$$

$$m_{01} \left(\sum I\right) = P_2/2$$

$$m_{10} \left(\sum J\right) = P_3/2$$

$$m_{20} \left(\sum I^2\right) = P_4/3$$

$$m_{11} \left(\sum IJ\right) = (2P_5 - P_6)/4$$

$$m_{02} \left(\sum J^2\right) = P_7/3$$

where the terms P_1, P_2 etc are accumulated according to the following formulae, where the value of each term is computed from its previous value and the current pixel edge co-ordinates:

If boundary code = 0

$$X = X + 1$$
$$P_3 = P_3 - Y^2$$
$$P_7 = P_7 - Y^3$$

If boundary code = 1

$$Y = Y + 1$$
$$P_1 = P_1 - X$$
$$P_2 = P_2 - X^2$$
$$P_4 = P_4 - X^3$$
$$P_5 = P_5 - X^2Y$$
$$P_6 = P_6 + X^2$$

If boundary code = 2

$$X = X - 1$$
$$P_3 = P_3 + Y^2$$
$$P_7 = P_7 + Y^3$$

If boundary code = 3

$$Y = Y + 1$$
$$P_1 = P_1 + X$$
$$P_2 = P_2 + X^2$$
$$P_4 = P_4 + X^3$$
$$P_5 = P_5 + X^2Y$$
$$P_6 = P_6 + X^2$$

More about moments

The purpose of obtaining these feature measurements is to compare them with stored measurements. However, this comparison is made much easier if the feature parameters are converted into a form that is not affected by a blob's position, orientation or scale. This is achieved by generating moment-invariants.

The first step is to generate central moments by shifting the co-ordinate system so that its origin co-incides with the centroid $(\bar{x},\bar{y})$. For a uniformly coloured region, R, its (p+q)th order central moment, μ_{pq} is defined by:

$$\mu_{pq} = \iint_R (x - \bar{x})^p (y - \bar{y})^q dxdy$$

Thus,
$$\mu_{00} = m_{00} = \mu$$

$$\mu_{10} = \mu_{01} = 0$$

$$\mu_{20} = m_{20} - \mu\bar{x}^2$$

$$\mu_{11} = m_{11} - \mu\bar{x}\bar{y}$$

$$\mu_{02} = m_{02} - \mu\bar{y}^2, \text{ and so on.}$$

Using these, for example, the variances in X and Y directions can be computed

$$\sigma_x = \left[\frac{\mu_{20}}{m_{00}}\right] \qquad \sigma_y = \left[\frac{\mu_{02}}{m_{00}}\right]$$

The angle θ which the major axis of the shape makes with the horizontal direction can be calculated:

$$\theta = \tfrac{1}{2} \arctan\left[\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right]$$

Finally, the eccentricity is given by the expression:

$$\left[\frac{\mu_{02}\cos^2\theta + \mu_{20}\sin^2\theta - \mu_{11}\sin 2\theta}{\mu_{02}\sin^2\theta + \mu_{20}\cos^2\theta + \mu_{11}\cos 2\theta}\right]^{1/2}$$

To achieve size and orientation independence, as well as position independence yielded by the central moments, the second step is to generate the moment invariants. These are:

$$M_1 = \mu_{20} + \mu_{02}$$

$$M_2 = (\mu_{20} - \mu_{02})^2 + 4\mu^2_{11}$$

$$M_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2$$

$$M_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2$$

and so on.

The first six moments, M_1 to M_6, are invariant under rotation and reflection. (For further details, see Levine, 1985, page 526)

Using 2-D shape measures

One example of the use of 2-D shape measures is given in Cheng et al., 1986. Each shape is enclosed by a rectangular box, from which a number of dimensionsless values are generated. These are recorded in a look-up table, for a range of orientations of the shape. An unknown shape is identified by comparing values extracted from its binary description with values in the look-up tables.

Another example of the use of 2-D shape measurements is given by Kruger and Thompson, 1981. Seven features were extracted for each type of object. They were:

x_1 Perimeter
x_2 Square root of area
x_3 Total hole area
x_4 Minimum radius
x_5 Maximum radius
x_6 Average radius
x_7 Compactness (x_1/x_2)

These features were abstracted from the foundry castings shown in Figure 11 and analysed by the decision tree shown in Figure 12.

Limitations of chain code

In conclusion, the advantages of chain coding is that it is compact, easily constructed, easy to understand and useful for 2-D shape recognition. However, it does have some limitations. As we saw earlier, a shape cannot be chain coded in a single raster scan of the binary description. This is a drawback when trying to analyse changing scenes in real time. Also, some basic operations, such as rotations and scale changes, are difficult to perform.

REFERENCES

Cheng, R.M.H. and Montor, T., 1986. Synchronization of an Industrial Robotic Manipulator Using Camera vision. In Proceedings of Conference on Intelligent Autonomous Systems, Amsterdam

Horn, B.K.P., 1986. Robot Vision. Cambridge, Mass. : MIT Press.

Kruger, R.P., and Thompson, W.B., 1982. A Technical and Economic Assessment of Computer Vision for Industrial Inspections and Robotic Assembly. Procs. of IEEE, 68, 1524-1538.

Levine, M.D., 1985. Vision in Man and Machine, New York : McGraw Hill.

Figure 1 If the background is bright, with little light falling on the subject of interest, a binary image can be obtained easily by thresholding the brightness values. This particular picture may lead us to believe that mere silhouettes can convey a great deal of information about three-dimensional objects. The artist's carefully chosen viewpoint and our familiarity with the subject matter conspire to give this impression. Silhouettes of unfamiliar objects, taken from randomly chosen points of view, are typically quite difficult to interpret. (Reproduced from *Silhouettes*, edited by C. B. Grafton, Dover, New York, 1979.)

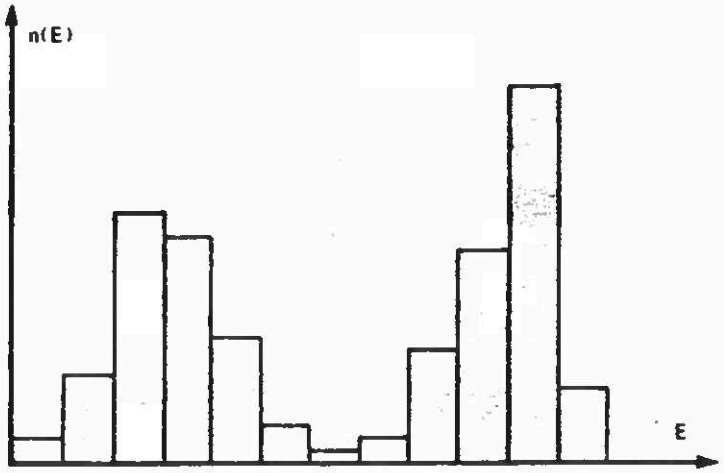


Figure 2 A histogram of gray-levels is sometimes useful in determining a threshold that can be used to segment the image into regions. Here $n(E)$ is the number of picture cells that have gray-level E.

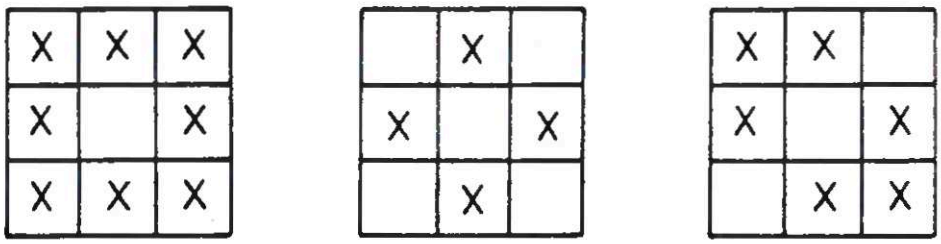


Figure 3. The x's indicate which pixels in a 3 by 3 neighborhood are connected to the center pixel using (a) 8-connectivity, (b) 4-connectivity, and (c) 6-connectivity.

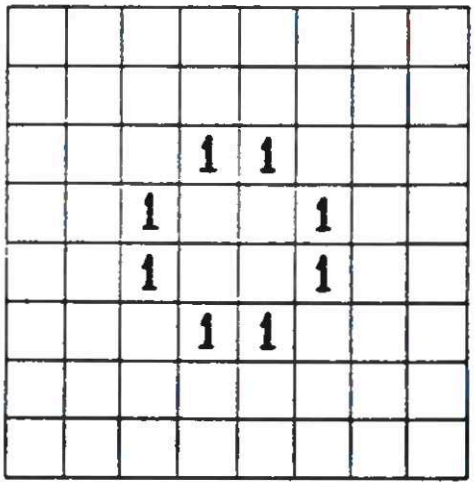


Figure 4. Connectivity in digital plane

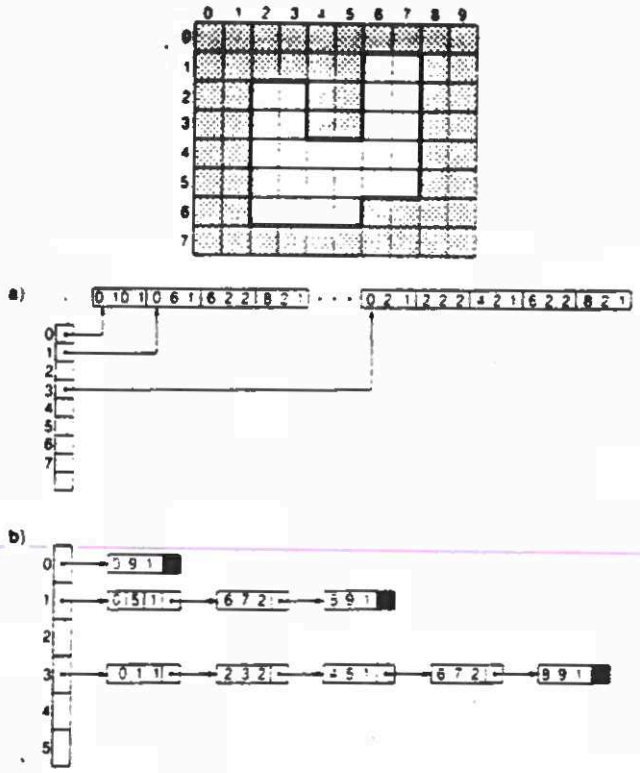


Figure 5 The blob finding algorithms generate a run-length list which encodes each run of consecutive 0's or 1's on a line as a three word record. The run-length list data structure for the connectivity analysis algorithm is shown in (a). Assuming the background is blob 1 and the object is blob 2, the three numbers in each record are interpreted as the start of a run, the number of pixels in the run, and number of the blob these pixels belong to, respectively. Part (b) shows the data structure for the boundary tracking algorithm. In this case, the second word is the right end of the run rather than its length.

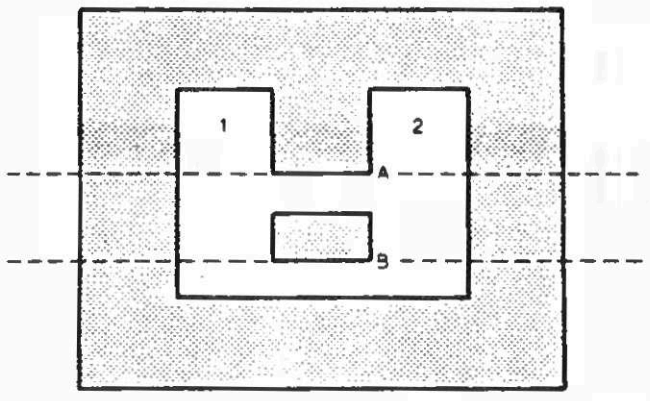


Figure 6 In the connectivity analysis algorithm, separate blob records are maintained for the components labeled 1 and 2 until the raster scan reaches the point labeled A, at which point the two records merge into one. When the scan reaches point B, it discovers that the same blob has surrounded an interior hole.

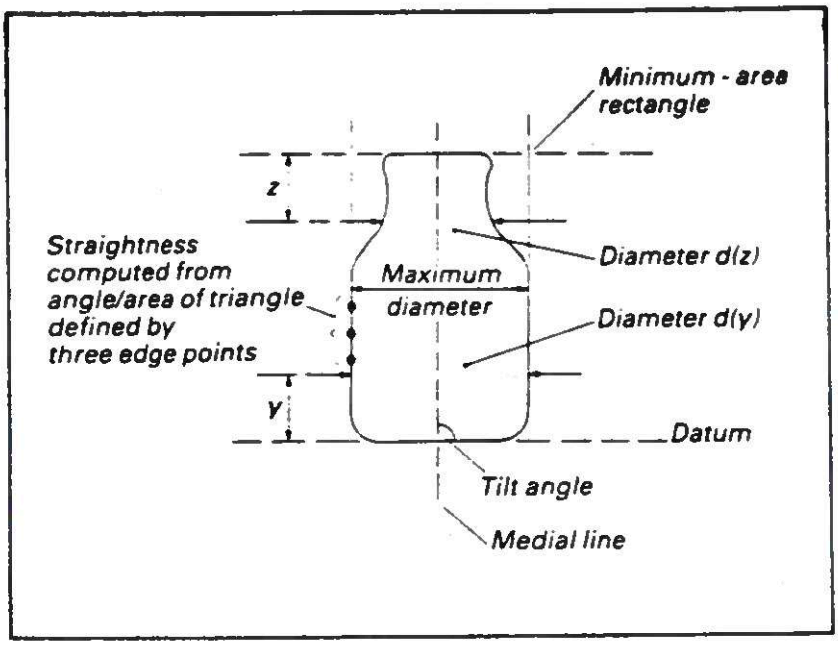


Fig. 7. Simple shape measurements defined on a bottle silhouette. The medial line is computed by finding the centre points of two horizontal chords, say y rows from the base and z rows from the top

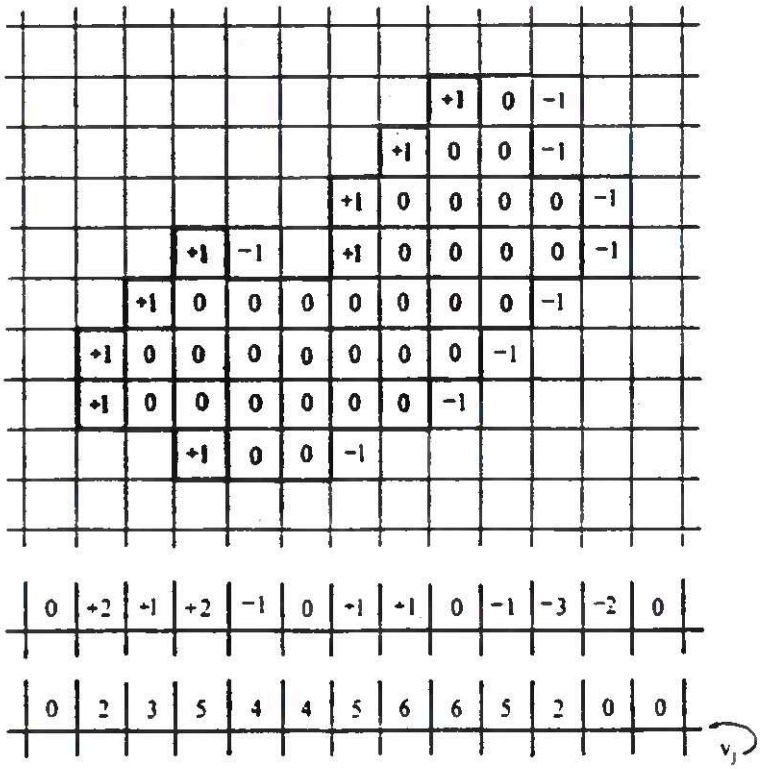


Figure 8. The first difference of the vertical projection can be computed from the projection of the first differences of the rows of the image. The vertical projection itself can then be found by summing the result from left to right.

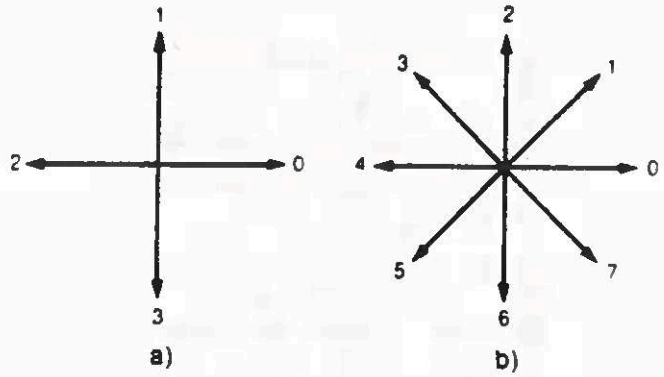


Figure 9. The boundary tracking algorithm uses a four direction chain-coding scheme with the four directions numbered as shown above in (a). The numbering for eight direction chain-coding is shown for comparison in (b).

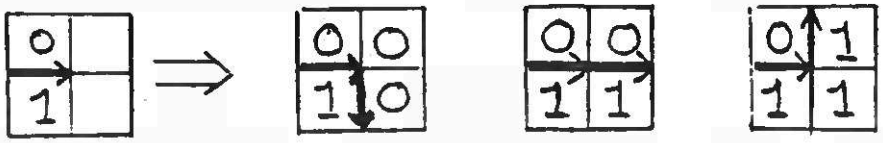


FIGURE 10. RULES FOR SELECTING NEXT BOUNDARY SEGMENT (0 = BACKGROUND)

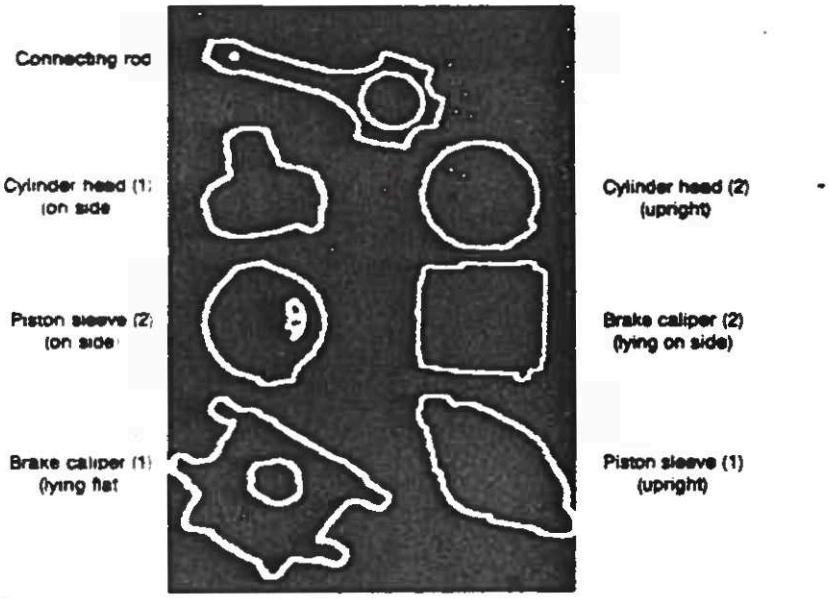


Figure 11 Edge-detected parts on a conveyor.

Source: R.P. Kruger and W.B. Thompson, "A Technical and Economic Assessment of Computer Vision for Industrial Inspections and Robotic Assembly," Proceedings of the IEEE, Volume 69, Number 12, December 1982, Figure 10, page 1530. ©1982 IEEE. Used by permission.

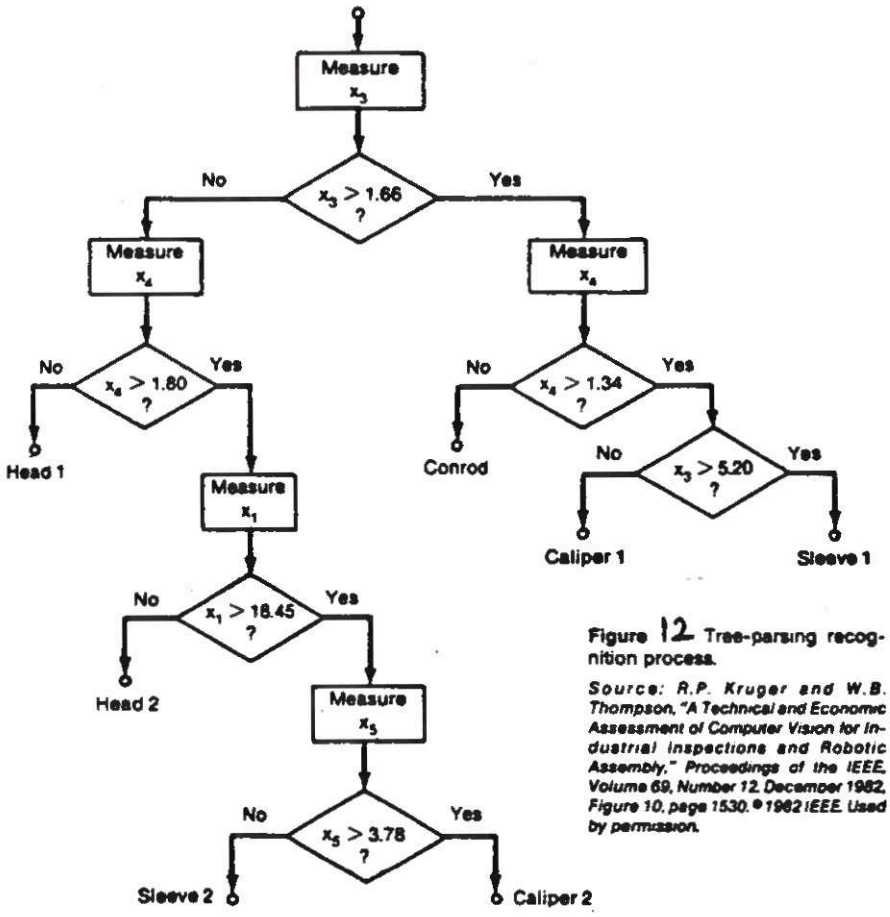


Figure 12 Tree-parsing recognition process.

Source: R.P. Kruger and W.B. Thompson, "A Technical and Economic Assessment of Computer Vision for Industrial Inspections and Robotic Assembly," Proceedings of the IEEE, Volume 69, Number 12, December 1982, Figure 10, page 1530. ©1982 IEEE. Used by permission.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

AI2 JIM HOWE

Detecting edge information

Having dealt with the recognition of 2-D shapes, we are going to work through techniques which will enable us to program a computer to recognise polyhedral objects. Our task will be broken down into four levels of analysis:

* The first level is concerned with extracting features, in particular the positions of discontinuities in the gray-level description.
* The second level is concerned with the construction of edge descriptions.
* The third level deals with segmentation of the edge descriptions into separate edge groups corresponding to distinct bodies in the scene.
* The fourth level is concerned with recognition of these bodies.

In this note, we will tackle the first level, extracting edge information from the gray-level description. This task can be broken down into two sub-tasks

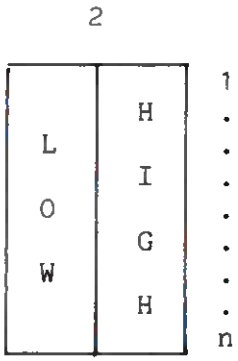
* Locating candidate edge points
* Grouping candidate edge points together to form continuous edges.

We will start by considering methods for locating candidate edge points.

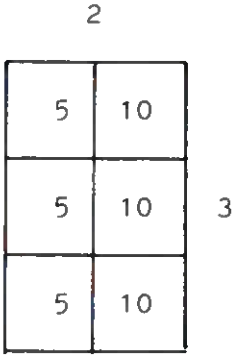
1. Detecting edge information: templates

Having built a gray-level representation of a scene, the next step is to extract interesting "local" features of the picture, i.e. features that can be discovered while looking at only a small portion of the entire representation. We will begin by considering how edge information might be extracted. The method that we will look at first of all combines the task of detecting the presence of edge segments and measuring their orientation. It is based on the use of an edge template, which is matched to the shape of a straight edge in the gray-level representation.

The template takes the form of an n x 2 array of cells, corresponding in size to an n x 2 sub-array of cells in the gray-level representation of the image. If the dark patches are represented by low values and light patches by high values, this template will detect a left-to-right transition of dark-to-light along a vertical edge. The template region labelled "low" would have values near the dark end of the gray scale, and the region labelled "high" would have values near the light end.



The system's task is to search for sets of values in the gray-level representation which match the sets of values in the template. It does so by applying a match rule. Suppose the luminance values in the representation lie in the range 0 (black) to 15 (white). Suppose, also, that we choose to use a 3 x 2 edge template. We will assign 10 as the value of the high region and 5 as the value of the low region.



The system chooses a 2 x 3 sample of luminance values from the digitised representation

Next, it compares the values in this sample with the values in the template, using a match rule. One example of a suitable match rule is as follows:

"Add 1 to the value of the match for each cell which corresponds spatially to a high value cell in the template and has a luminance value of 10 or more, and add 1 to the value of the match for each cell which compares spatially to a low value cell in the template and has a luminance value of 5 or less. The entire template is said to match at any position for which the total value of the match is 4 or more".

Given that a match is established, the description obtained is in the form of (x,y) co-ordinate values of the end points of each edge segment, and an associated orientation value of 90°.

Separate edge segments are linked together into larger segments, corresponding to edges of objects, by applying grouping rules:

> If the end point of one segment is adjacent to (e.g. above, below, to-the-left of, to-the-right of) the end point of another segment, and
> If the orientation of the first is the same as the orientation of the other, and
> If the combined segments (combined points) are collinear
> Then link the segments (to form a larger segment).

Note 1. "Same" must be given a value in practice.

2. The collinearity test must be assigned an error threshold

Junctions (corners) are detected as follows:

If the end points of two (or more) segments are adjacent, and
If the orientation of one is different from the other(s)
Then combine the segments (to form a junction).

Note "Different" must be given a value in practice.

At first sight, this method is attractive but the major difficulty is that different templates and/or different match rules are needed for left-vertical edges, right-vertical edges, horizontal edges, bright edges, dim edges, sharp edges, fuzzy edges and edges at arbitrary orientations. In other words, the process of using templates is computationally costly.

2. Detecting edge information: Gradient operators

Detecting evidence of the existence of an edge in an image involves estimating the magnitude and direction of the gradient at various positions within the gray-level description. The gradient is a vector, whose magnitude G and orientation θ can be expressed as

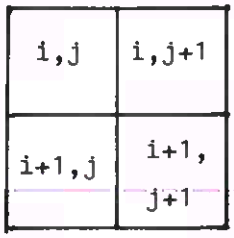
$$G(x,y) = \{(\delta f/ \delta x)^2 + (\delta f/ \delta y)^2\}$$
$$\theta(x,y) = \tan^{-1} (\delta f/ \delta y)/(\delta f/ \delta x)$$

For obvious reasons, the process of registering abrupt changes in the gray-level representation is known as sharpening, edge-enhancement or spatial differentiation.

(1) Cross Operator

Our first illustration of spatial differentation is the approach taken by Roberts (1965). He used solid polyhedral objects, namely cubes, wedges and prisms. These were specially prepared and specially lit so that the surfaces were very homogeneous. In other words, he was trying to make sure that the luminance variations in the image due to the object/background boundary would resemble the luminance variations produced by a step change in intensity. Since his objects were specially prepared, no initial smoothing was attempted.

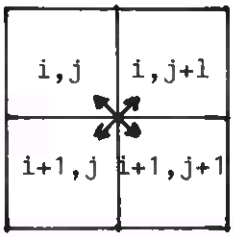
Because he had taken so much care to optimise the transformation from image feature to gray-scale description, the gradient was measured over the smallest possible area of the gray-scale representation, a 2 x 2 window:



by computing the sum of squares of the differences between diagonal pixels:

$$R(i,j) = \sqrt{(f(i,j) - f(i+1, j+1))^2 + (f(i+1,j) - f(i,j+1))^2}$$

Notice that Roberts is taking the difference between diagonal elements in the window



Because of this, the operator is popularly known as the Cross Operator. Notice that since diagonal neighbours are $\sqrt{2}$ times as far away from (i,j), a diagonal difference in the magnitude of the luminance change tends to be larger than the magnitude of the luminance change taken in a horizontal (or vertical) direction, given that the slope components in the different directions are equal. This is compensated for by taking the square root of the differences.

Let us consider the operator's qualitative behaviour. If the point (i,j) is in a region of uniform luminance, the value R(i,j) is zero. If there is a discontinuity between columns j and (j+1), then R(i,j) has a large value, and similarly if there is a discontinuity between rows i and (i+j).

In practice, this technique constructs a new representation whose points lie between the points of the original picture (e.g. $i+\frac{1}{2}, j+\frac{1}{2}$).

In passing, we should note that the cross operator is often simplified for computational efficiency by using absolute magnitudes rather than square or square roots. The operator becomes

$$R(i,j) = |f(i,j) - f(i+1,j+1)| + |f(i,j+1) - f(i+1,j)|$$

So by applying this cross operator to the digital representation, he produces a new representation, called a gradient representation. This contains a set of candidate edge points. Obviously low values correspond to areas of uniform luminance whereas high values are associated with changes in the luminance which may correspond to edges in the original scene.

In passing, note that the direction of the gradient, α, can be computed as follows for each candidate edge point

$$\alpha = -\frac{\pi}{4} + \tan^{-1}\left[\frac{f(i,j+1) - f(i+1,j)}{f(i+1)(j+1) - f(i,j)}\right]$$

Since the direction of an edge is normal to the direction of its gradient, edge direction must be computed for each point. These values are stored, for later combination to generate edges.

As we saw, Roberts took great care to minimise data degradation. He did this for the simple reason that the cross operator is not able to cope with poor quality data, either in the form of isolated noise points or blurred edges.

(2) High Pass Filter

We turn now to look at another method of spatial differentiation which is less susceptible to the effects of noise than Roberts' 2 x 2 operator. The way in which this is done is to combine averaging and differencing.

This has led to the development of a number of gradient estimators using windows of various sizes and differing weights.

As an example of a gradient operator using a 3 x 3 window, consider the following operator which is usually referred to as a high-pass filter (introduced by Prewitt):

a	b	c
d	e	f
g	h	i

We define Dx by

$$Dx = (c+f+i) - (a+d+g)$$

and Dy by

$$Dy = (g+h+i) - (a+b+c)$$

Then we define the gradient at the point e by either $Ge = \left(Dx^2 + Dy^2 \right)^{1/2}$ or if we want a more computationally efficient definition

$$Ge = |Dx| + |Dy|$$

The direction of the edge, θ, is given by

$$\theta = \tan^{-1} \left(\frac{Dx}{Dy} \right)$$

For example, suppose a small portion of a digitised array carries the following values:

Columns	A	B	C	D	E	F	G
Rows 11	1	1	2	3	5	5	6
12	1	1	1	5	5	6	5
13	1	1	1	5	6	5	5

If we pass the 3 x 3 window across the whole array we obtain the following values:

B12	Dx = 1,	Dy = 1	G = 2	-
C12	Dx = 11,	Dy = 0	G = 11	$\alpha = 84°$
D12	Dx = 12,	Dy = 1	G = 13	$\alpha = 85°$
E12	Dx = 2,	Dy = 2	G = 4	-
F12	Dx = 0,	Dy = 0	G = 0	-

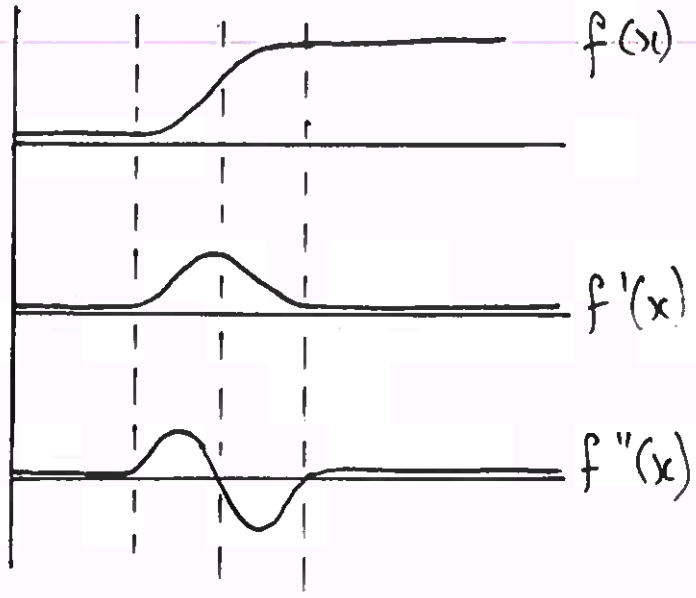
Larger weights may be given to the pixels adjacent to the central pixel (e), as in the case of the Sobel operator:

a	2b	c
2d	e	2f
g	2h	i

Note that these operators, and the Roberts' operator, are not perfectly isotropic, i.e. edges of the same strength but differing orientations give different edge magnitude outputs. Also, the selection of candidate edge points is done by setting an a priori threshold and eliminating any points whose gradient value is less than the threshold value.

(3) Laplacian operator

The problem of threshold selection can be avoided by using an operator that computes the second derivative since the zero crossings indicate edges:



Taking first differences in the x and y directions as

$$fx(i,j) = f(i,j) - f(i-1,j)$$
$$fy(i,j) = f(i,j) - f(i,j-1)$$

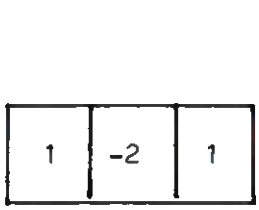
where **f** is the image intensity and **i** and **j** are row and column co-ordinates, the higher order differences are derived by repeating the first order differences:

$$\begin{aligned} fxx(i,j) &= fx(i+1,j) - fx(i,j) \\ &= f(i+1,j) + f(i-1,j) - 2f(i,j) \\ fyy(i,j) &= f(i,j+1) + f(i,j-1) - 2f(i,j) \end{aligned}$$

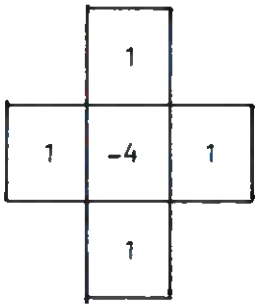
The sum of the second order differences, ∇^2 (the Laplacian) is

$$\begin{aligned} \nabla^2 f(i,j) &= fxx(i,j) + fyy(i,j) \\ &= [f(i+1,j) + f(i-1,j) + f(i,j+1) + f(i,j-1)] - 4f(i,j) \end{aligned}$$

The second differences can be represented by the one dimensional windows



and the Laplacian operator ∇^2 by



Note that the Laplacian operator is isotropic.

The disadvantages of the Laplacian are:

(i) Useful directional information is not available.
(ii) It doubly enhances any noise in an image.

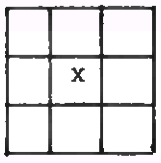
The operators discussed above have assumed (almost) perfect edge data. However, as we have seen earlier, edge data is often far from perfect. This means that an edge can be distributed over a number of pixels in the gray-level description. Such an edge will not be "seen" by the operators described above. Instead, operators with larger windows are required. Now, it's commonplace to use a range of these operators to process the description at several levels of spatial resolution.

Extracting edges

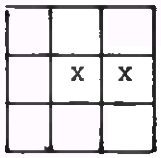
The second problem is that of grouping candidate edge points to form a continuous contour that can be segmented into lines and junctions. Several methods are available.

(i) Tracking

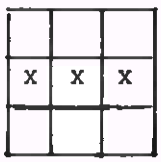
In the last note, we discussed methods for tracking the boundary of a blob. A set of decision rules can also be developed to track through candidate edge points, using 8-directional chain coding. Some of the more important are as follows:



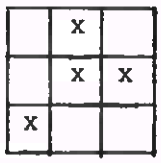
- A point with no neighbours. Isolated value.



- Example of point with one neighbour at edge. This is an end point.



- Example of point with two neighbours



- Example of point with three or more neighbours. Fork.

Their application is illustrated in Figure 1.

Some problems with the above include:

(i) the implicit assumption that the edge contour is but one pixel wide

(ii) due to noise, false edge elements are found, as are gaps in the edge

(iii) tracking is a local process which is difficult to control, bridge gaps, etc. See Shirai's approach in a later note on Knowledge Guided Segmentation.

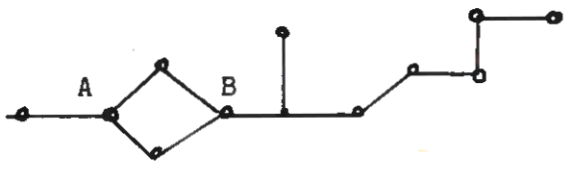
(2) Grouping/line fitting

In his classic program, Roberts uses grouping rules to combine candidate edge points into edges.

The rules are:

If two edge points are adjacent, and
If the orientation of one is the same as (i.e. within ± 10°) of the other, and
If the new point is collinear with the existing points (as judged by fitting a straight line to the data using a sequential least-mean square-error fitting routine).
Then link the new edge point to the existing point(s).

These grouping rules produce a network of edge segments which must be thinned, for example:



All dangling segments (i.e. connected at one end only) are removed. Quadrilaterals are replaced by lines linking connecting points (A and B).

After thinning, the edge description may be cleaned up by re-applying the line fitting algorithm, filling in gaps, locating junctions where two or more lines share a common point, and so on.

(3) Hough transform

To detect edges, the Hough technique can be used in preference to the grouping operations described previously. The basic concept underlying this technique is that by transforming input data from the (X,Y) domain into a different domain, related features in the input data will show up as clusters of features. The advantages of the Hough technique are that it is relatively unaffected by gaps in the edge data and by noise (both of which can only be handled in an ad hoc way by the earlier grouping techniques by introducing additional rules to relax the adjacency criterion, etc.).

We begin the explanation by applying the technique to the detection of points lying on a straight line i.e. points satisfying the equation $y = mx + c$. Since **m** might be infinite, the normal form equation of the straight line is preferred, viz:

$$x \cos\theta + y \sin\theta = r$$

where **θ** is the angle made between the x axis and a normal to the line, and **r** is the length of this normal. A graphical interpretation of this equation is given in Figure 2.

Consider, now, the edge element **l'**, in Figure 3, which has four collinear points. Although an infinite number of lines can be drawn through each of these points, let us suppose that line **l** , through (x', y'), is a typical line. As explained above, it can be characterised by the two paraemters (θ, r). Thus, we can see that any arbitrary point (x, y) on line **l** is constrained by the equation $x \cos\theta + y \sin\theta = r$. Suppose we fix (x, y) in the equation at (x',y'). Now, the equation will define a relationship between θ and r. This function is sinusoidal, as shown in Figure 4.

Each (θ, r) pair in the graph parameterizes one of an infinite number of straight lines passing through the point (x' y'). This can be verified by imagining line **l** rotating around (x',y'), whereupon θ will change through 360° and r will vary between two limiting values. In other words, the point (x',y') in the original X-Y image space has been projected into the (θ,r) Hough transform space as a sinusoidal curve. If the transform is applied to all four points in Figure 3, the family of sinusoidal curves shown in Figure 5 is generated. Since each of the points lies on the same straight line **l'** parameterized by (θ,r'), each of the transformed curves must pass

through the point (θ',r'). The intersection characterizes the value of (θ,r) which defines the line passing through all four image points.

To discover the point clusters produced by the intersecting lines, an accumulator array is set up, with bins corresponding to the different possible combinations of discrete parameter values. For each candidate edge point, a 'vote' is placed in every bin whose corresponding parameter set could have given rise to that instance. At the end of the voting process, each local peak in the accumulator array will correspond to a group of collinear edge elements in the image. The parameters on the line on which the elements lie will be given by the indices of the peak, and an estimate of the number of points on the line by the sum of accumulator values at, or very close to, the local maximum. In the case of Figure 2, the array cell (θ',r') would register a count of 4; all other cells would register a 1 or a 0.

Obviously, the performance of the Hough transform method is affected by the quantization chosen. If it is too coarse, it will fail to distinguish lines that are close together. If too fine, it will be intolerant of errors in collinearity. Also, in cases where the orientation of candidate edge points is known, the technique can be simplified. Then, a single point, rather than a sinusoid, can be computed in the (r,θ) space for a given edge point.

REFERENCES

Nevatia, R. (1982). Machine Perception. Englewood Cliffs, New Jersey : Prentice Hall.

Roberts, L.G. (1965) Machine Perception of Three Dimensional Solids. In Optical and Electro-optical Information Processing, Chapter 9. Boston, Mass : MIT Press.

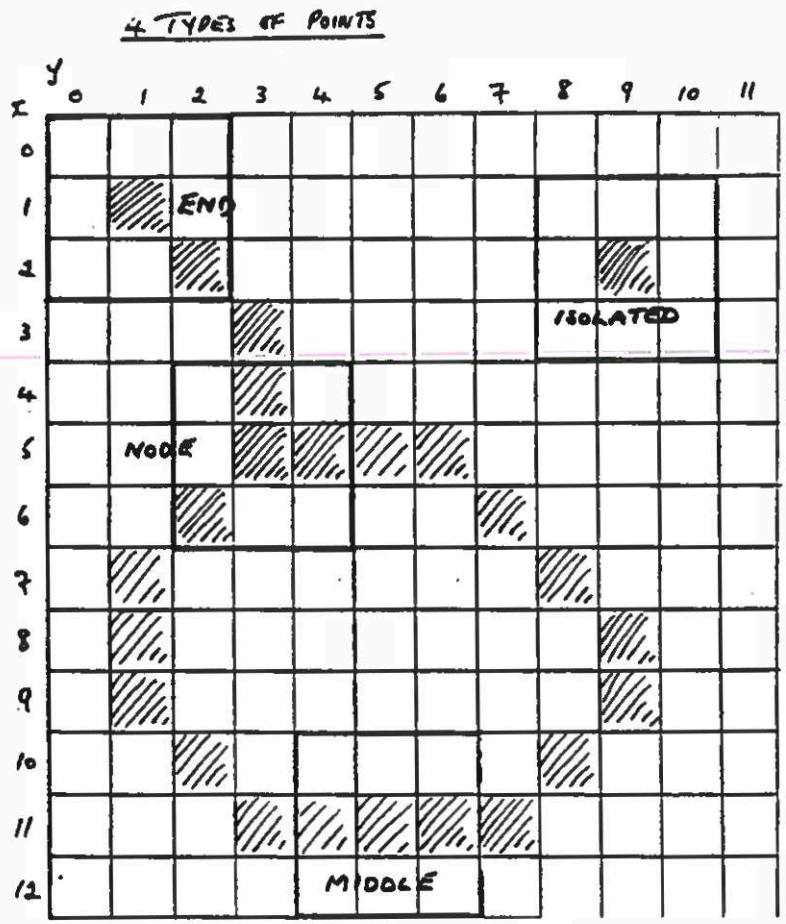


Figure 1. Decision Rules

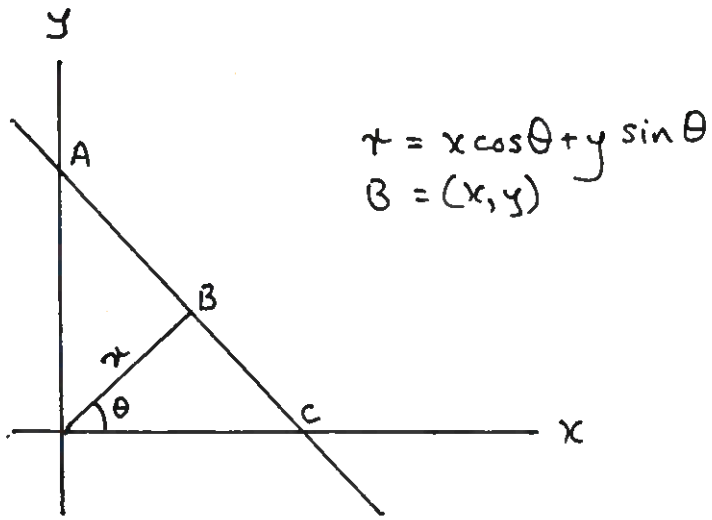


FIGURE 2

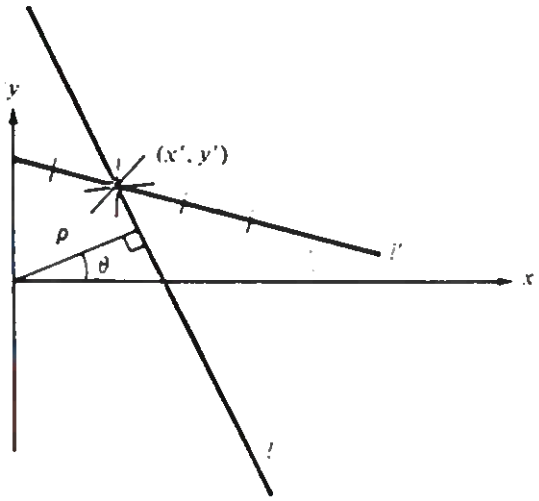


FIGURE 3

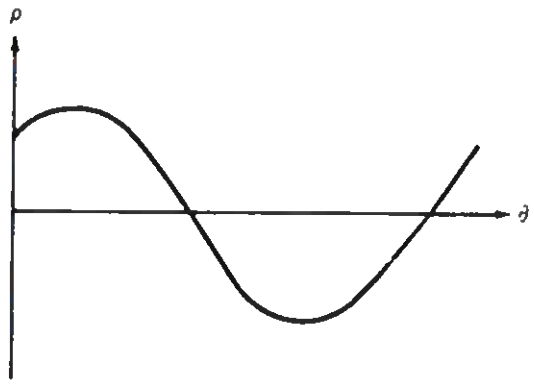


FIGURE 4

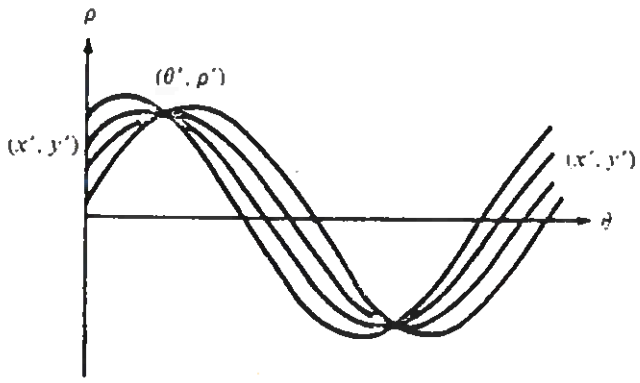


FIGURE 5

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

A I 2

JIM HOWE

How SEE Sees

So far, we have only considered single outline shapes. Now we want to handle more complicated scenes, containing multiple, overlapping bodies. When we look at a line drawing of blocks world objects, we can readily see which regions (corresponding to surfaces) belong to each of the bodies present in the scene. The task of partitioning a set of regions into bodies is known as the segmentation task, and the difficulty of the task in the case of polyhedra depends upon the degree of overlap between bodies, i.e. it depends upon the amount of occlusion in a scene (the amount by which one region is obscured by another one).

One of the earliest programs developed to tackle the segmentation problem was Guzman's SEE (Guzman, 1967). We will examine SEE in some detail since it was the first of a series of programs, each of which built on the ideas or experiences with the previous one, gradually reducing the need for ad hoc rules by providing a better theoretical justification of the underlying processes.

In SEE, Guzman assumed as starting point the existence of a perfect edge representation of a polyhedral scene. A typical example is the scene called BRIDGE, shown in Figure 1. This is input to the program in the form of unordered lists of object regions, background regions and vertices. Notice that the program does not have to separate objects from background: this information is provided by Guzman.

To parse the scene into bodies, SEE follows a two part strategy. First, it collects evidence for linking regions. Then second, it evaluates this evidence and groups regions to form objects.

We will begin by considering the first part strategy, namely collecting evidence. It is based on the fact that some places in a picture contain more information than others, namely the points at which several lines meet - the vertices or picture junctions. Guzman noticed that the shape of a junction was a pretty reliable indicator of its three-dimensional significance. For example, a three-line junction which looks like an ARROW-head is usually the corner of a convex object, where only two out of the three surfaces of the body are visible. In practice, Guzman classified junctions into four basic types:

1. Vertices where two lines meet, e.g. L
2. Vertices where three lines meet, e.g. ARROW, FORK, T
3. Vertices where four lines meet, e.g. K, X
4. Other vertices, e.g. PEAK, MULTI

Examples are shown in Figure 2.

With each type of vertex there is an associated set of links which constitute the evidence for conglomerating adjacent regions in the scene. These links are of two types, namely strong and weak links. The strong links associated with each vertex are as follows:

1. Ls, Ks, MULTIs and single Ts have no links.

2. FORK. Links are planted between the three regions, meeting at a vertex of the FORK type, except
   (a) if one region is the BACKGROUND no link is placed;
   (b) if one of the lines is connected to an L, or to the barb of an arrow, or forms the bar of a T, the regions on either side of that line are not linked.

3. ARROW. Links are placed between the two regions on either side of its shaft, except
   (a) if the shaft of the ARROW is connected to an L, the regions on either side of the shaft are not linked;
   (b) if the shaft of the ARROW is connected to a background FORK, or to the stem of a background T, the regions on either sides of the barbs are linked.

4. X. Two cases are distinguished.
   (a) If the X is formed by the intersection of two lines, no links are planted.
   (b) If the X is formed by four lines, two of which are collinear, the regions on either side of the collinear lines are linked.

5. PEAK. All regions, except the one containing the obtuse angle, are linked to each other.

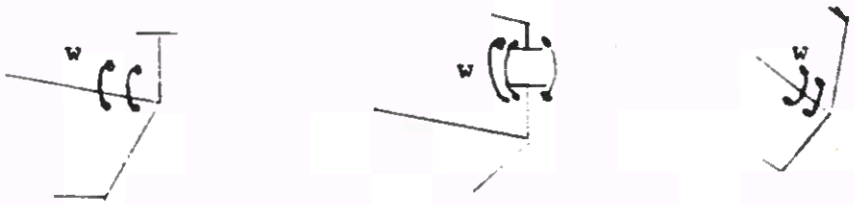
6. T pairs. Facing pairs of Ts with collinear stems are linked, provided the area between the bars is not BACKGROUND.

7. 3-parallel T. The regions on either side of the stem of the T are linked in the case of a 3-parallel T.

Weak links, planted in addition to strong links, are associated with the type of vertex called LEG.

LEG is an ARROW where one of the barbs of the ARROW is connected to an L which has one line parallel to the shaft of the ARROW (if necessary through a chain of matched Ts).

e.g.



Examples of the links associated with these junction types are given in Figure 3.

Having classified the vertices in the scene, as shown in Figure 4, the second step is to combine and group the link evidence to partition the scene into its constituent bodies. The evidence for the scene BRIDGE is shown in Figure 5, in which the regions are depicted by circles. Strong links are represented by solid arcs; weak links by dotted arcs. All the links to the background (:30) have been deleted since the background cannot be part of any body.

Now the program attempts to form nuclei, where a nucleus is either a region or a set of nuclei which has been formed by the following rule: if two nuclei are connected by two or more strong links, they are merged into a larger nucleus by concatenation. For example, in Figure 6, regions :24 :25 :27 :12 and regions :21 and :9 are put together. As a consequence, nucleus :24 :25 :27 :12 has two links with nucleus :21 :9, so they are combined in turn to form a new nucleus :24 :25 :27 :12 :21 :9 as shown in Figure 7. So, the nuclei are allowed to grow and merge until no new nuclei can be formed. When this is the case, the scene has been partitioned into several "maximal" nuclei: between any two of these, there are zero or, at most, one link.

The program has still to consider the effect of weak links. The rule is that if a strong link joining two maximal nuclei is reinforced by a weak link, these nuclei are merged, as shown in Figure 8. For example, in scene BRIDGE, the following weak links exist: :13 to :15 :14 to :15 :3 to :17 :7 to :4 :8 to :11 :10 to :4 :5 to :6 :28 to :29 :18 to :19 :25 to :27 :22 to :26 :23 to :26

Notice that nucleus :16 is linked to nucleus :18/:19 by a single strong link. This invokes another rule to the effect that a strong link joining a

nucleus and another nucleus composed by a single region is sufficient evidence for the nuclei in question to be merged if there is no other link emanating from the single region. This yields the final parsing shown in Figure 9. In summary:

i. Form nuclei from regions connected by two or more strong links.

ii. Amalgamate nuclei joined by two or more strong links until no new nuclei can be formed.

iii. Amalgamate nuclei joined by one strong and one weak link.

iv. Amalgamate a nucleus joined to a single region nucleus by a strong link.

Ignoring the single links between nuclei which remain after parsing, the program returns the results

```
(BODY1. IS :24 :9  :21 :27 :12 :25)
(BODY2. IS :22 :26 :23)
(BODY3. IS :17 :3  :20)
(BODY4. IS :1  :2)
(BODY5. IS :14 :15 :13)
(BODY6. IS :19 :18 :16)
(BODY7. IS :29 :28)
(BODY8. IS :8  :11 :5  :6  :4  :10  :7)
```

How good is SEE? Since it requires two pieces of strong evidence to join two nuclei, it is conservative, i.e. it will almost never join two regions that belong to different bodies. Its errors are almost always of the same type: regions that should be joined are left separate. This suggests that more heuristics should be added to provide additional linking evidence. The problem is that adding a heuristic can cause repercussions: it may solve the difficult case but in turn cause other difficulties. Rather than continue to derive rules in an ad hoc way, it would be preferable to derive them from an explicit 2D/3D representational theory which takes into account the overall geometry of polyhedral bodies. This is what we will consider next.

References

Bundy, A. et al. (1978) Artificial Intelligence: An Introductory Course. Section 4.3, pp.141-147.

Guzman, A. (1967) Decomposition of a visual scene into bodies. A.I. Memo 139, Artificial Intelligence Laboratory, M.I.T.

Nevatia, R. (1982) Machine Perception. New Jersey: Prentice Hall. Chapt. 5, pp. 41-45.

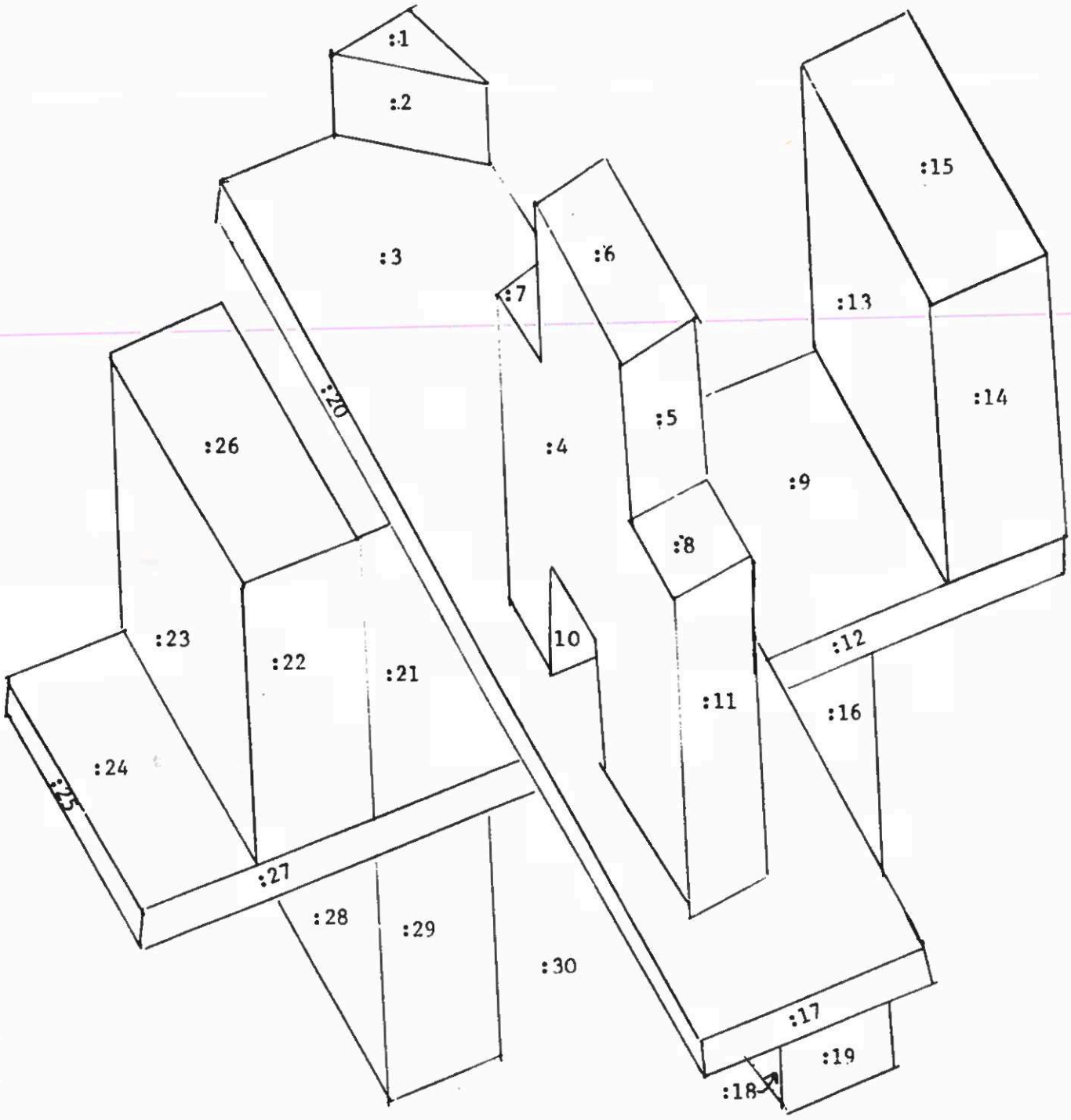
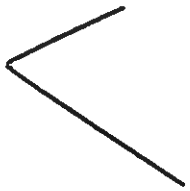


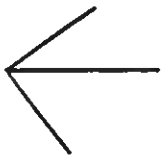
Figure 1. 'B R I D G E'

The long body :25 :24 :27 :21 :9 :12 is correctly identified.



L

FORK



ARROW

T

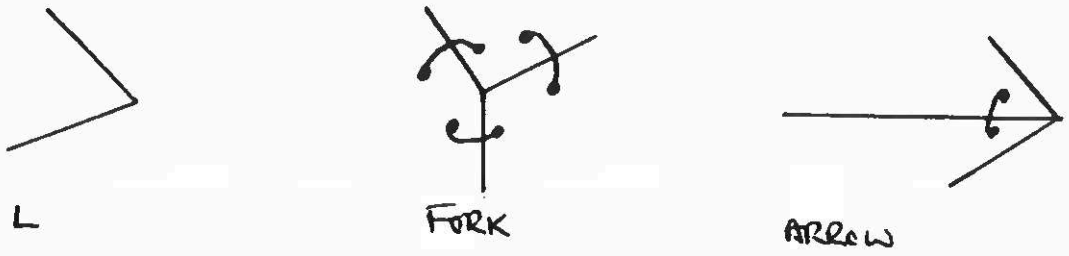
K

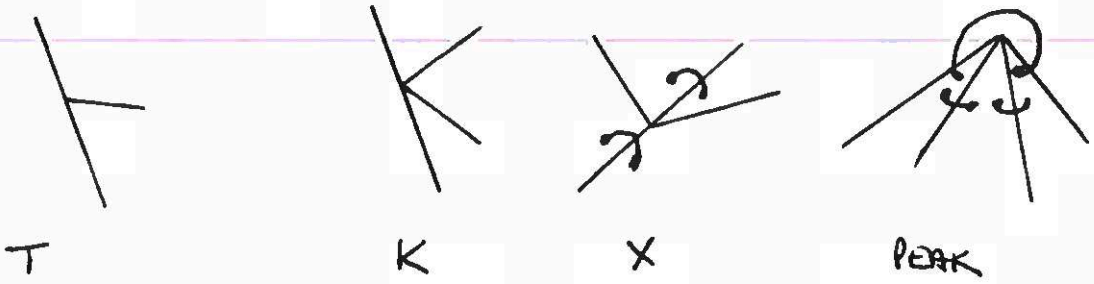
X

PEAK

MULTI

Figure 2. Junction Types.





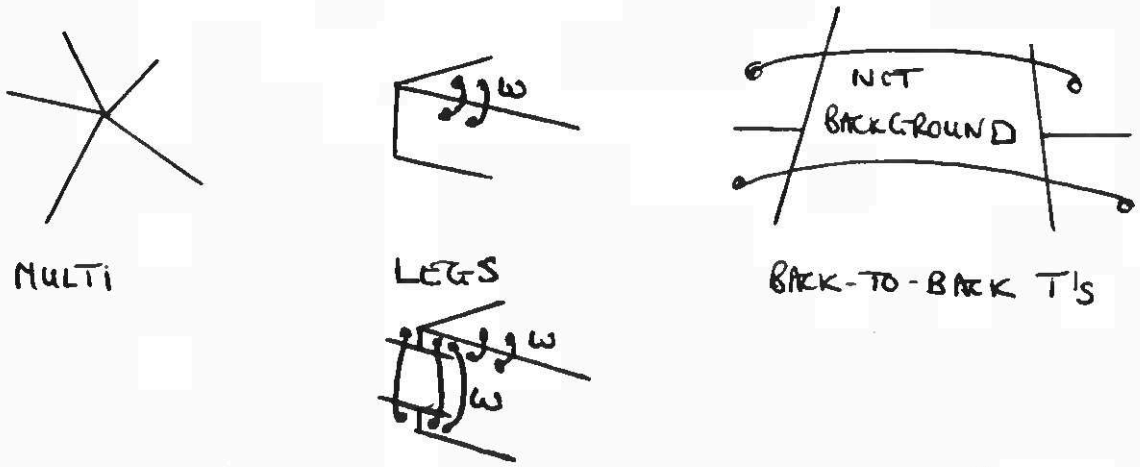


Figure 3. PLANTING LINKS

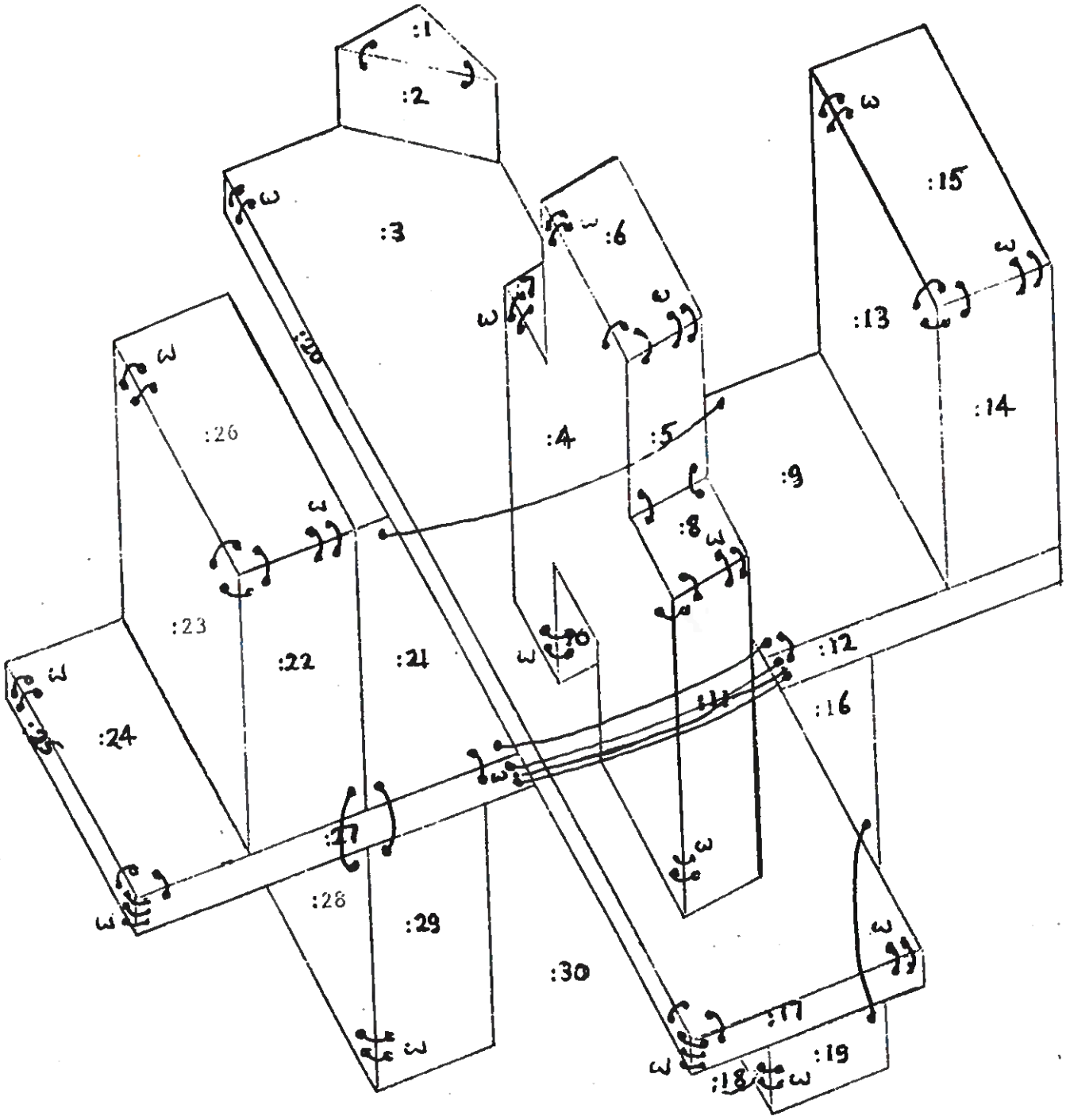


Figure 4 'B R I D G E'.

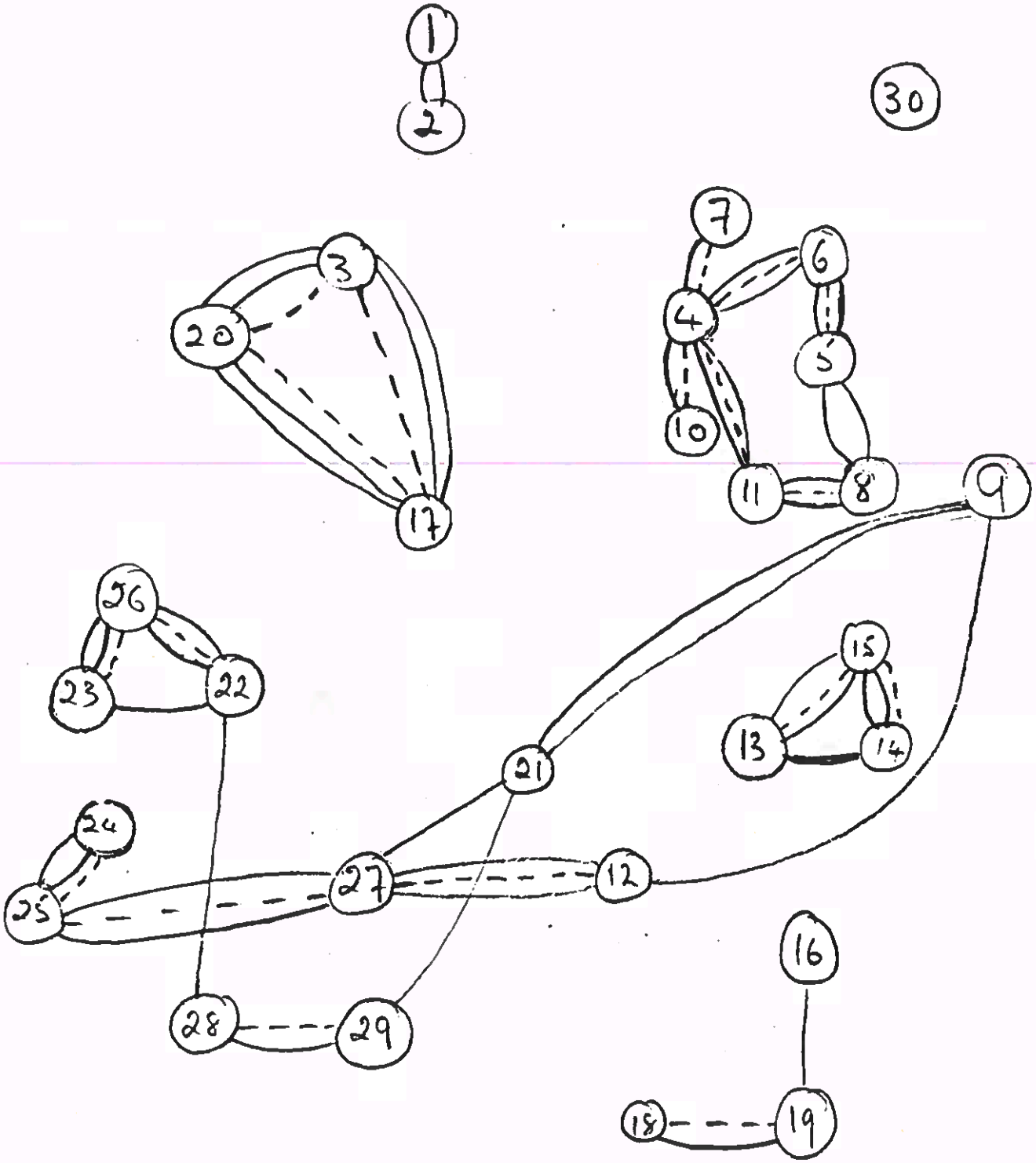

1
2
30
3
20
17
7
6
4
5
10
11
8
9
26
23
22
15
13
14
21
24
25
27
12
16
28
29
18
19
Strong link
Weak link

FIGURE 5

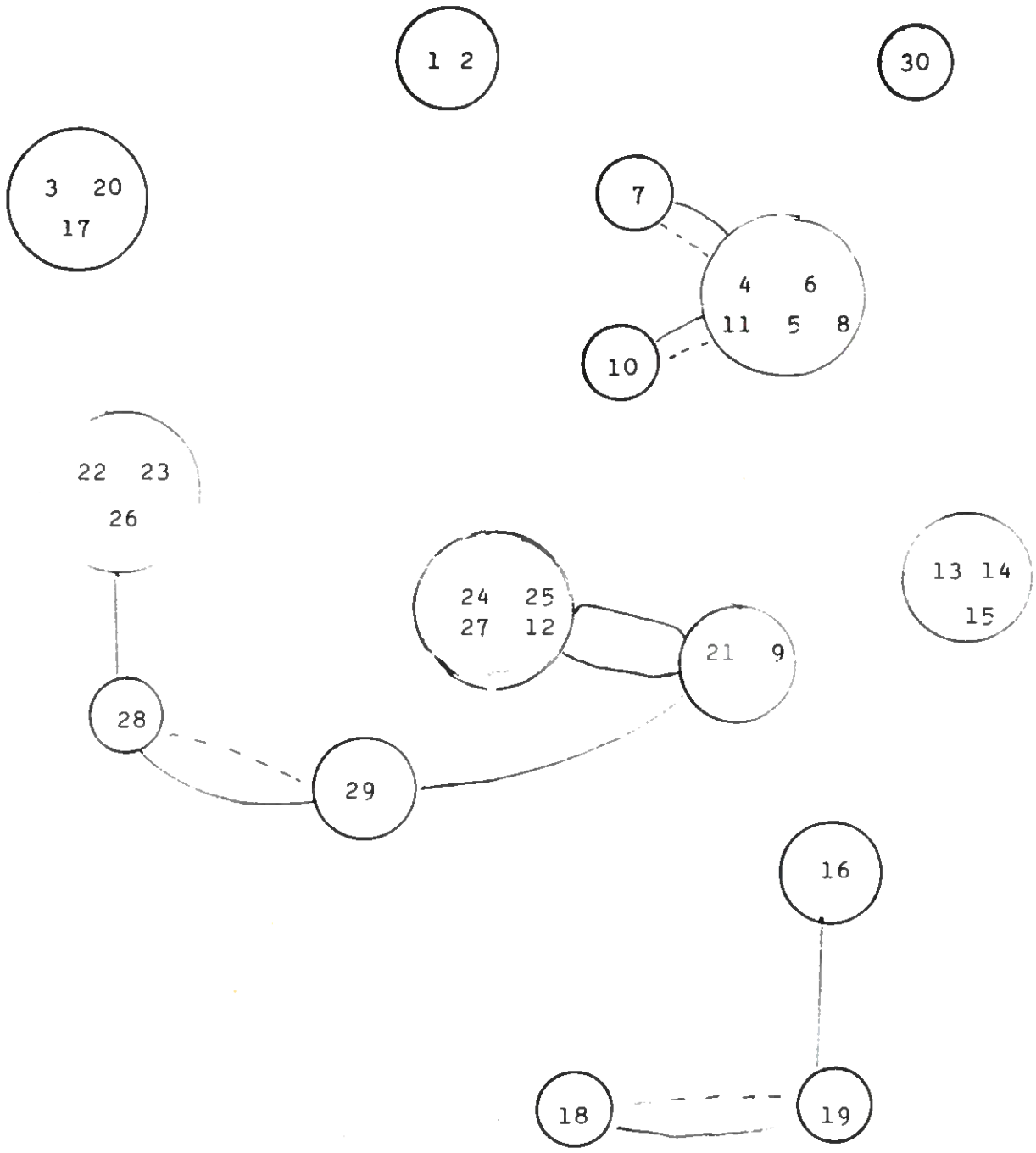


Figure 6. 2 strong links

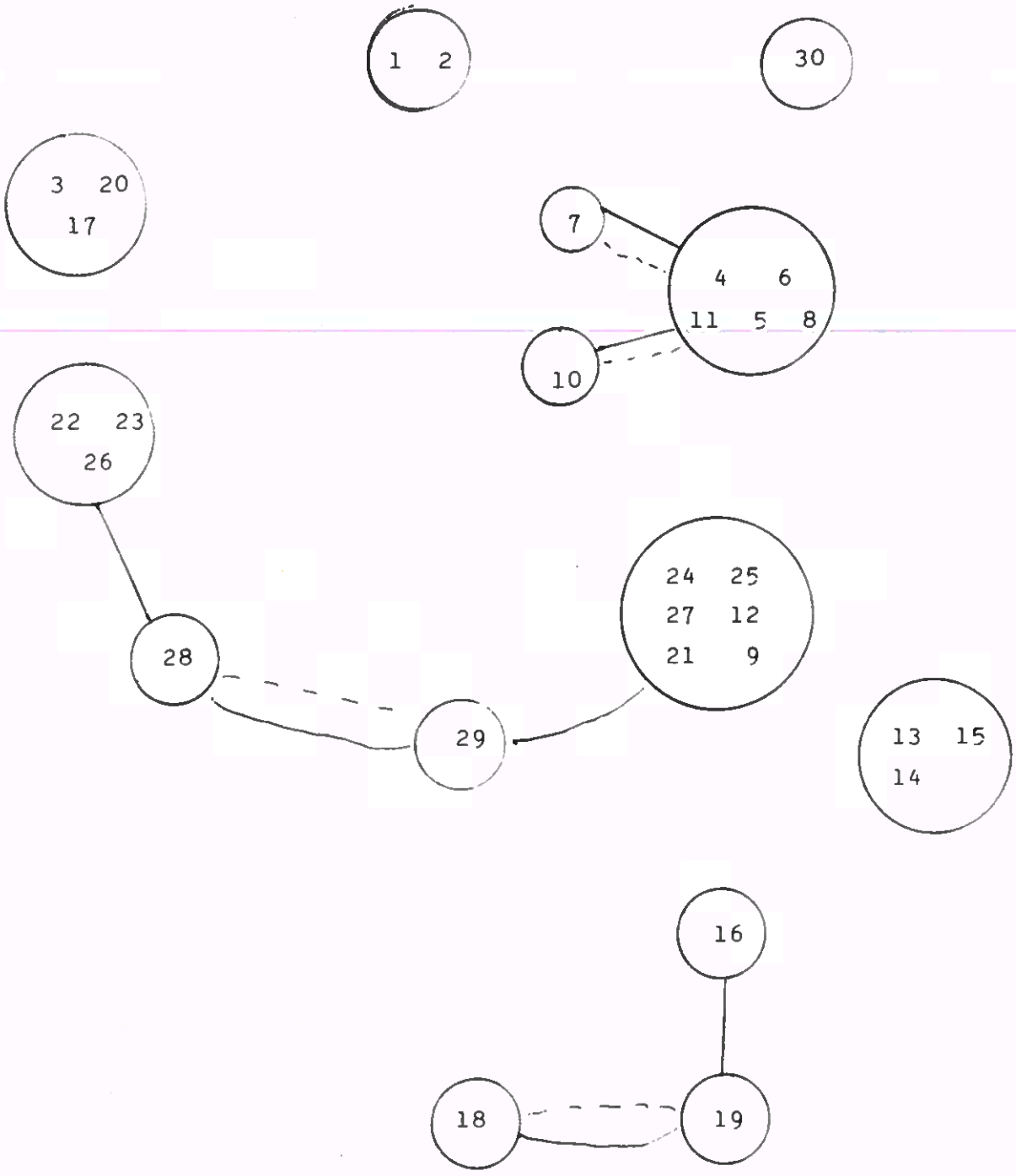


Figure 7. 2 strong links.

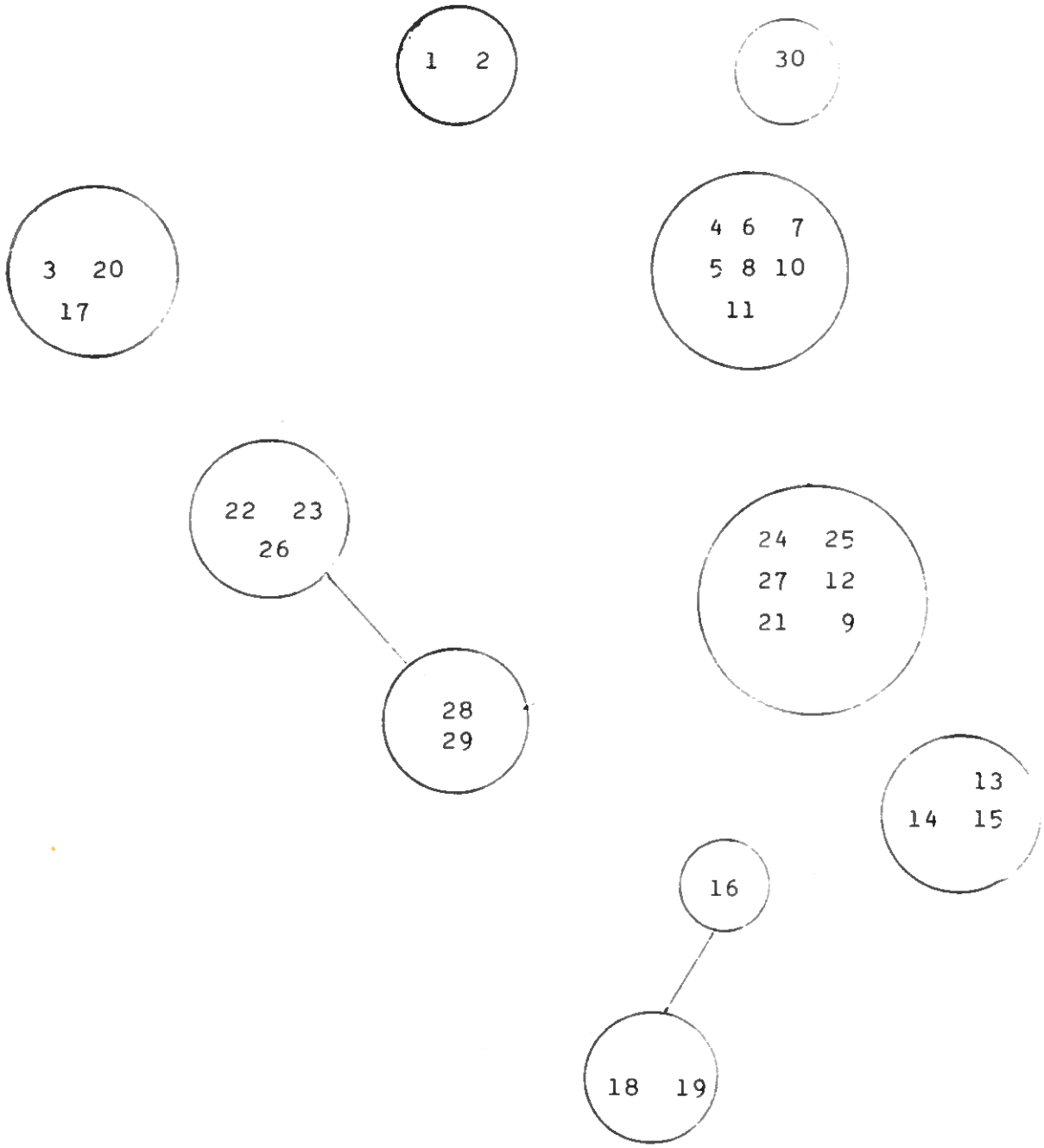


Figure 8. 1 strong and 1 weak link.

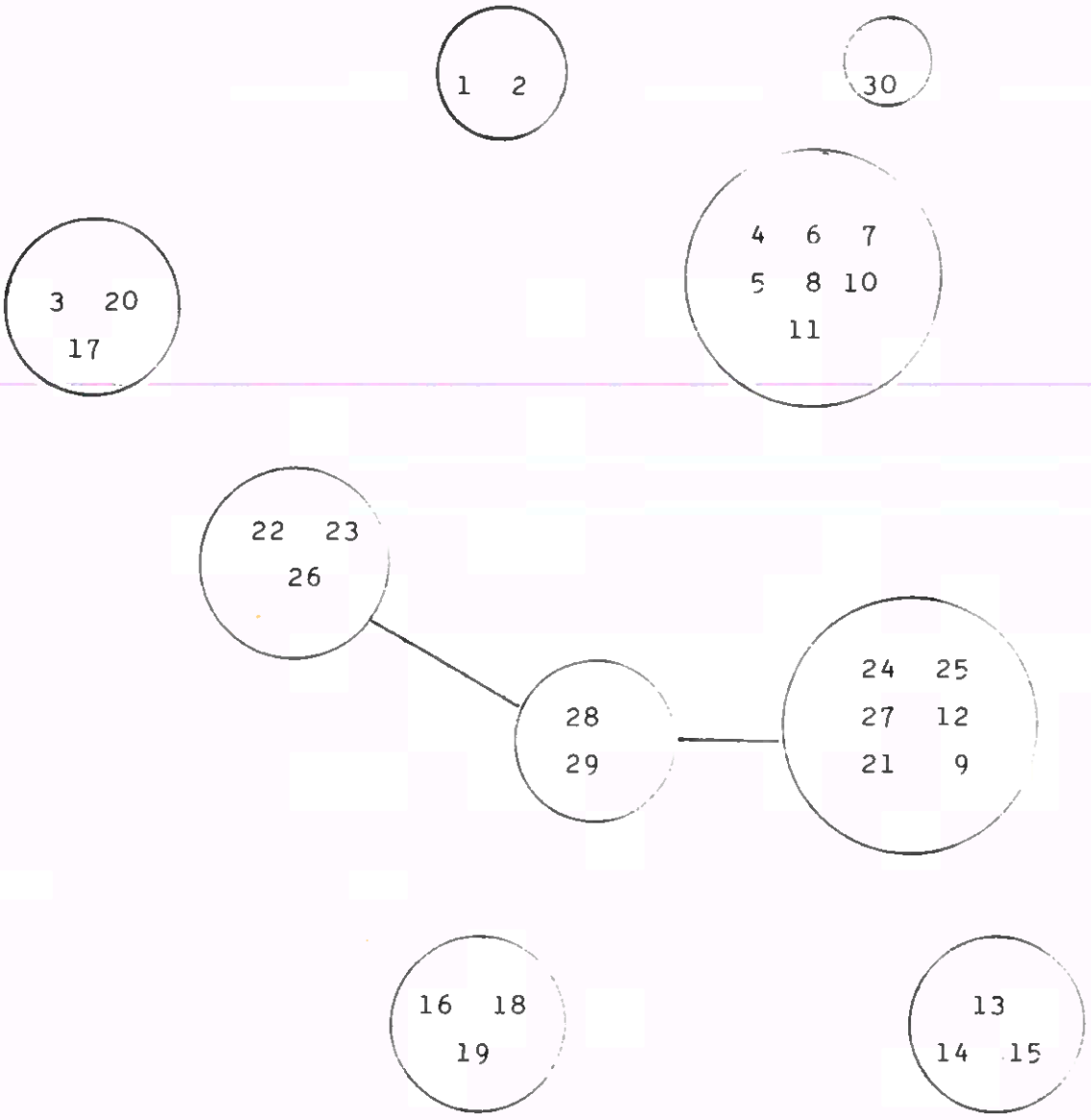


Figure 9. Single region nucleus with 1 strong link.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

Exploiting physical constraints

When we discussed Guzman's program, we found that its rules for *linking regions* depended on the *shape* of the local junction. In contrast, we turn now to consider later work by Huffman, Clowes and Waltz who realised that by devising rules for *describing and linking junctions*, not only could they obtain a segmentation of the scene into bodies, but they could also derive information about the 3-D shape of the bodies.

As we have already noted, SEE made most use of trihedral vertices - the so-called ARROW and FORK junctions. Now, *a trihedral vertex is a point of intersection of three planes which partition the surrounding space into eight octants*. This is shown graphically in Figure 1. Some of the ways in which these octants can be filled by three surfaces which meet at a vertex are shown in Figure 2, where the number of octants actually occupied yields a type number. For example, type 1 is like Guzman's ARROW, and type 7 is like his FORK junction.

Imagine now that you can view a vertex from each unoccupied octant. The possible views for the four vertex-types are shown in Figure 3. Labels are associated with lines in these drawings. Let's see what these labels denote.

1. a '+' marks a *convex* edge which has both corresponding planes visible.
2. a '-' marks a *concave* edge which has both corresponding planes visible.
3. an '<' marks an *occluding* edge where one plane is hidden, the *visible* plane being to the right of the direction in which the arrow is pointing.

By understanding the three-dimensional nature of the scene, we are able to apply the labels of the drawing. The important question is *can a program use line-labelling to help it to understand the three-dimensional nature of line drawings?* The answer is that it can do so by applying to the vertices in a drawing the set of labelled line configurations which we obtained by labelling the possible views for the four-vertex types. The set of twelve possible configurations is shown in Figure 4. Notice how this approach limits the number of labellings for the different configurations. For example, given four labels, there should be 16 ways of labelling an "L" junction but there are only *6 legal* labellings shown.

Huffman (1971) was interested in showing that the use of the labelled-line configurations (which we will refer to as corner models) would enable us to tell when certain kinds of drawings are impossible. If we look at the picture given in Figure 5, it will be rejected as a possible line drawing for a plane-faced object because there is no set of labels which will consistently label it.

Labelling the outer contour is straightforward - the only allowable labels are arrow-type labels. If we move on now and consider the two arrow-type vertices, we find that the contour labelling has assigned arrow labels to the lines on either side of the shaft in both cases. Inspection of our list of legal corner models shows that there is only one arrow-type vertex with arrow labels assigned to bounding lines. Selecting this forces a + label for the shaft which is entered accordingly. If we now consider the L vertex between the shafts of the two arrows, we find that each leg of the L has been assigned a + label. But inspection of the list of corner models indicates that this is *not* a legal corner model - there is no L configuration with plus labels on each leg so we conclude that the drawing does not represent a three-dimensional object of trihedral vertices.

Similar considerations apply when we examine the eight objects shown in Figure 6. In the case of example (a), the labelling of the outer contour forces us to label the shafts of the arrows with a + since this is the only legal corner model which can be applied. This forces + labels on all three lines forming the arrow in the middle, and inspection of the list of corner models indicates that it is *not* a legal labelling. Exactly the same problem crops us in examples (b) and (f), and in the case of example (c) we see a recurrence of the labelling problem encountered in the case of the drawing of the incomplete cube, seen earlier. Example (d) is a second example of an illegal L model, whereas example (e) has an illegal fork junction. Examples of other types of illegal arrow labellings are shown in examples (g) and (h).

Huffman only considered single objects, using a hand-worked analysis. Clowes (1971), working independently on the problem, devised a computer program "OBSCENE" to perform this kind of analysis. Since it was designed to handle scenes with multiple objects, involving consideration of additional fork and T-junctions, Clowes' program was equipped with a larger set of corner models.

Working at M.I.T., David Waltz (1975) generalized the Huffman/Clowes ideas in two fundamental ways:

1. He expanded the set of line labels to capture more information about

the physical situation, and

2. He devised a filtering procedure that reduces the search problem by quickly converging on the possible interpretations.

Waltz's system consists of a working set of computer programs which categorize the lines in a scene as boundary, shadow, convex, concave and crack types. In addition, the system groups regions which belong to the same object, calculates object orientation, and notices such relations as contact or lack of contact, support, in-front-of, and behind. Not only can it give a 3-D description of the shadowy scene shown in Figure 7, it can also recognize the very different picture in the next Figure 8 as representing the same scene.

Line labels

Let's begin by considering the extensions of the system of labelling. In the scenes analysed so far, the problem of handling shadows has been deliberately avoided. Waltz, on the other hand, wanted to be able to analyse less artificial scenes, and decided to try to deal with shadows and cracks . *Shadows* are indicated by labelling the shadow line with a short arrow, *pointing into* the shadow. This adds two new labels since a shadow can be on the right or on the left of a line. A *crack* appears where there is only one plane but two faces, and this is indicated by the label, C. These are shown on the left-hand side of Figure 9. Now that we have 7 labels for a line, the number of possible combinations at an L should be 49. However, we already saw that with 4 labels the number of legal labellings was restricted to six. By a somewhat elaborate process, similar in principle to the one described previously, the number of legal ways to label an L, given seven possible labels, is only 9.

The important point to notice at this stage is that the amount of constraint is increasing as we make our descriptions more detailed. For example, we've increased the number of line labels from 4 to 7, the number of vertex possibilities for an L has increased from 16 to 49 but the number of *legal* vertex labels has only increased by 3, from 6 to 9. As Winston points out, the percentage of legal vertices has decreased from 30% to about 20%.

A further classification can be made according to whether or not each edge can be the bounding edge of an object (see Figure 10). One effect of doing this is to increase the number of *concave* edge types from 1 to 4. What is the difference between them? The answer is that the difference is determined by what objects look like when we separate them. If we look back at the right-hand side of Figure 9, the only change in the case of object (a) is

the addition of the - sign to the bottom edge to indicate the support relationship, i.e. the edge denoted by the arrow labels is a separable two-object concave edge.

Whereas example (a) was a single object, examples (b) and (c) are in two parts which can be separated. In case (b), imagine we pull the objects apart. Previously the concave line between the objects was labelled with a *minus*; and the line between the two adjoining faces was labelled as a *crack*. Now we see that concave edge corresponds to a separable two-object concave edge, where the labelling relates to the cube shape, i.e. the arrow points in the direction which maintains the visible surface on the right. This labelling suggests either a support relationship, as in (a), or an occlusion situation where one object partially occludes another object. The crack edge is labelled with a downward pointing arrow to indicate that the object to the right partially obscures the object to the left of the edge.

Applying the same reasoning to case (c), the edge labelling of the concave edge separating the two objects changes to indicate that the edge belongs to the upper object, and the crack label is changed to indicate the *support* relationship between them.

Finally, example (d) explains the third new label - the double arrow. This is used in situations where *three* objects meet along a single edge, and it can be seen that the labelling of the concave edge in example (d) incorporates the labels used in examples (a),(b),(c).

Now that we have a clear picture of the way that vertices constrain the labelling of objects, can we find any further constraints to help with the analysis? Waltz decided to examine the effects on regions of the illumination of the scene. Let us assume that the scene is brightly lit by a single light source. If the light hits a surface directly, it is an *illuminated surface* (I). If, however, the body of an object interposes between a surface and the light source, the surface is a *self-shadowed* (SS). Finally, where a surface would be illuminated by the light source if it were not for the interposition of another object, it is classified as a *shadow-projected surface* (SP).

To ascertain if there are constraints among region illumination, let us look at the example in Figure 11 which has already been appropriately labelled. Consider the lower, right-hand vertex. Let's begin by assessing what we already know about the situation. First of all, the shadow line implies the existence of a shadow-projected region and an illuminated region - in

practice, regions C and B. Looking again at the labelling we see the regions B and A are related by a *concave* edge which implies that if region B is illuminated so also must region A.

But what about region D? Again we know from the labelling that one edge has a plus label. What this means is that region D could be self-shadowed, or illuminated but it could not be shadow projected. To be shadow projected, the edge label would be an arrow, not a plus, so a projected shadow is not a possible labelling for region D.

What else do we know? Well, we know that the adjoining region C is a projected shadow, so can D be a projected shadow? The answer is yes, it can be a projected shadow, and equally it cannot be illuminated because the edge between C and D is a minus edge. As we already saw, if one side of a minus edge is illuminated, then the other side is also illuminated. Consequently since C is a shadow, region D must also be a self-shadow or projected shadow.

So the relationship between regions A and D suggests that D is illuminated or self-shadowed, and the relationship between C and D suggests that D is a self shadow or projected-shadow region. The only possibility that satisfies both is self-shadow.

What we have seen is that there is an intimate relationship between edge type and scene illumination. Waltz took advantage of this to define new edge labels which included information about the lighting on both sides of the edge. He was including at the edge level (a very local level) information which constrains all edges bounding the same two regions. Put another way, wherever a line can be assigned a single label which includes this lighting information, then the junctions which can appear around either of the regions which bound this line are highly constrained.

Figure 12 gives tables relating region illumination types. For example, if either side of a concave or crack edge is illuminated, both sides of the edge must be illuminated. These tables were used to expand the set of allowable junction labels. Prior to the addition of region illumination, the total number of legal labels in the data base was 717. After including the region illumination, the total increased to 3,256. The breakdown by junction type is shown in Figure 13.

Since Waltz assumed that each scene would be of blocks on a horizontal table top, any line segment separating the background (table) from the rest of the scene can only be labelled in a very few ways, as shown in Figure 14. Because of this fact the number of junction labels which could be used to label junctions on the scene/background boundary can be greatly restricted, as shown in Figure 15.

We have now dealt fully with the way in which Waltz extended the number of labelling possibilities for junctions. Out task now is to understand how possibilities are eliminated to get a unique parsing for an object or set of objects. The strategy which Waltz follows is in two parts.

1. He uses *selection rules* to eliminate as many labels as possible on the basis of relatively *local* information such as location of a junction relative to the background or region brightness. We have already seen that only about one-tenth of the physically possible vertices can occur on the scene background boundary, whereas all of them can be found inside the scene. For example, only two of the ten PEAK interpretations make sense on the scene-background boundary. Consequently, Waltz's program locates this boundary *before* attempting any labelling. This can be achieved by finding all the regions that touch the edge of the field of view and combining them, or by finding the contour that has the property that every junction lies on or inside it.

Turning now to region brightness, in theory a line in a scene can be assigned any of 57 possible labels. But once it is known that there is an arrow at the end of it, as shown in Figure 16, the number of possibilities drops to 19. Suppose also the relative brightness of regions R1 and R2 are known. There are three possibilities:

i. R1 is darker than R2
ii. R2 " " " R1
iii. R1 is equal to R2

If (i) is true, if L-A-B is a shadow edge, R1 must be the shadowed side and R2 the illuminated side. If (ii) is true, the opposite is the case. If (iii) is true, then L-A-B is not a shadow edge.

At any rate, if the brightnesses of R1 and R2 are known, no more than 18 and as few as 15 labels remain. Also, if the brightness of R3 is known, then as few as 5 and no more than 18 labels will remain possible.

All the junctions lying on the scene-background boundary are labelled first. Next, the program labels the junctions that bound regions that share an edge or junction with the background, since these will be more constrained by the background neighbours than by their internal neighbours. Finally, the more central junctions are labelled (analagous to building a jig-saw puzzle from edge to middle).

2. Next, Waltz filters out labels which cannot be part of any total scene labelling. To understand the filtering process, we assume that the selection

rules have been applied to reduce the lists of possible candidate models for each vertex of the shape. The filtering program then computes the possible lebels for each line, using the fact that a line label is possible if and only if there is at least one junction label at each end of the line which contains the line label. Let's see how it works in the simple example of a cube, shown in Figure 17.

Step 1

Compare A and B for mutually exclusive junctions. Since there are no outgoing arrows in A, we have no ingoing arrows in B.
Eliminate B1 and B6.

Step 2

Compare remains of B, viz. B2 B3 B4 B5 with C. Since there are no ingoing arrows in C, eliminate outgoing arrows in B.
Eliminate B5
Now there are no + labels in B, so
Eliminate C3 and Eliminate A3.

Step 3

Compare remains of C, viz. C1 and C2, with D. Since there are no + labels or outgoing arrow labels in C, there can be no + labels on ingoing arrows in D, so
Eliminate D1, D5 and D6.

Step 4

Compare remains of D, viz. D2, D3 and D4, with E. Since there are no + labels or outgoing arrows in D, there can be no + or ingoing arrow labels in E, so
Eliminate E3.

Step 5

Compare E1 and E2 with F. Since there are no + or outgoing arrow labels in E, there can be no + or ingoing arrow labels in F, so
Eliminate F1, F5 and F6.

Step 6

Compare remains of F with A.
No further elimination, so filtering is complete.

Optional heuristics can be invoked to select a single labelling from among those which remain after all the other knowledge in the program has been used. These heuristics find a "plausible" interpretation if required. For example, one heuristic eliminates interpretations that involve concave objects in favour of ones that involve convex objects, and another prefers interpretations which have the smallest number of objects; this heuristic prefers a shadow interpretation for an ambiguous region to the interpretation of the region as a piece of an object. Also, special case heuristics deal with the labelling of non-trihedral vertices, the accidental alignment of edges, and missing lines in the picture.

The program has reached the stage where it successfully handles scenes such as those shown in Figure 18. The segments which remain ambiguous after its operation are marked with stars.

We are now in a position to understand why Guzman's program works. You will remember that we noticed that it worked best on scenes with convex trihedral vertices, that is with *convex* objects. Accordingly, we can eliminate from Huffman's corner interpretations all corners with *concave* edges, including those for the L that imply a hidden *concave* edge, leaving the set shown at the bottom of Figure 19. Notice that L, FORK and ARROW junctions now have unique corner interpretations, where the + labels, which indicate *convex* edges, also match Guzman's links, i.e. *we can derive Guzman's links by planting a link at a convex edge and no link at an occluding edge.*

Also, link suppression rules (no link is placed across a line at a junction if its other end is a barb of an ARROW, a leg of an L, or the crossbar of a T) are equivalent to the rule that the opposite ends of a line must have the same labelling. Indeed, the accumulation of link evidence based on the existence of *two* links between surfaces means in effect that both ends of an edge must agree that it is convex for it to be so taken. If only one end says so, i.e. one link, there is a conflict which must be heuristically resolved in Guzman's system.

References

Waltz, D. (1975) Understanding Line Drawings of Scenes with Shadows. In *The Psychology of Computer Vision* (ed. Winston).

Huffman, D.A. (1971) Impossible Objects as Nonsense Sentences. In *Machine Intelligence 6*, (eds. Meltzer and Michie).

Clowes, M.B. (1971) On Seeing Things. In *Artificial Intelligence, 2*, 79-116.

Nevatia, R. (1982) Machine Perception. New Jersey: Prentice Hall. Chapt. 4, pp. 45-50.

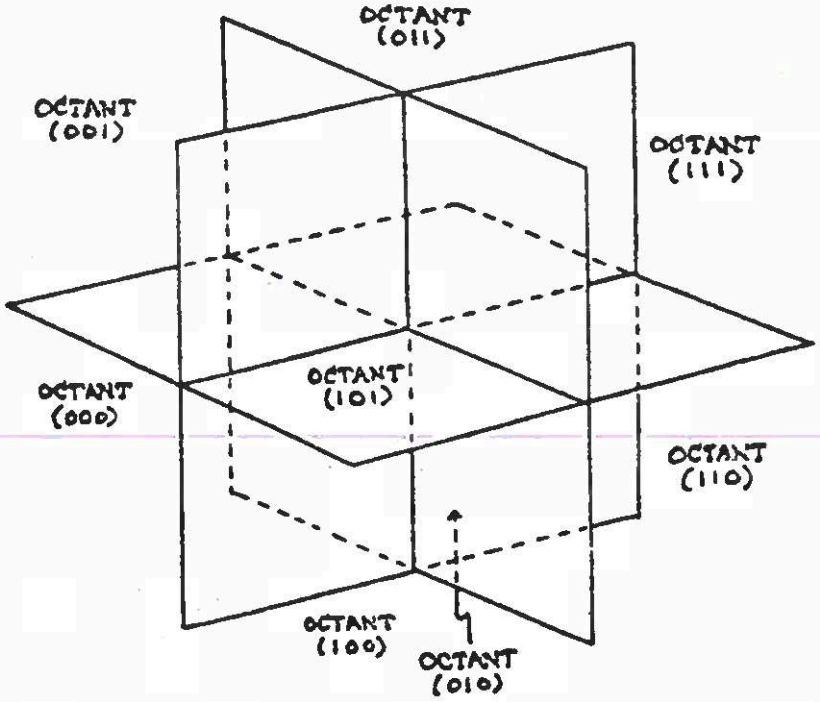


FIGURE 1

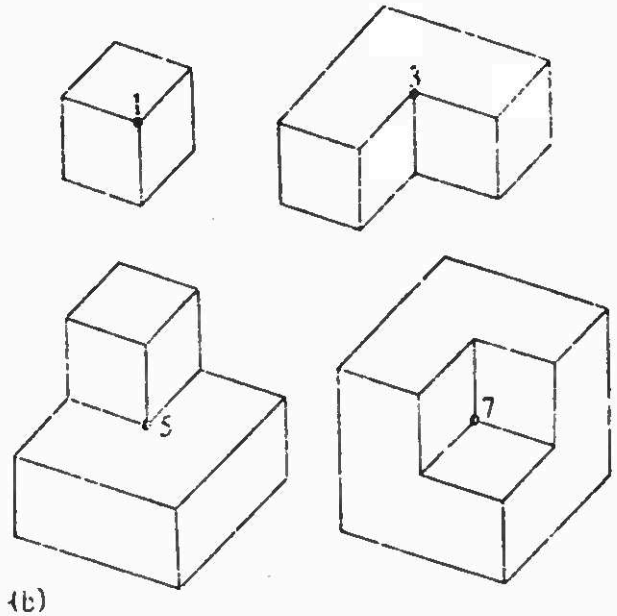


Figure 2 Illustrating the four types of vertices: (a) a fireplace and hearth; (b) the four vertex types

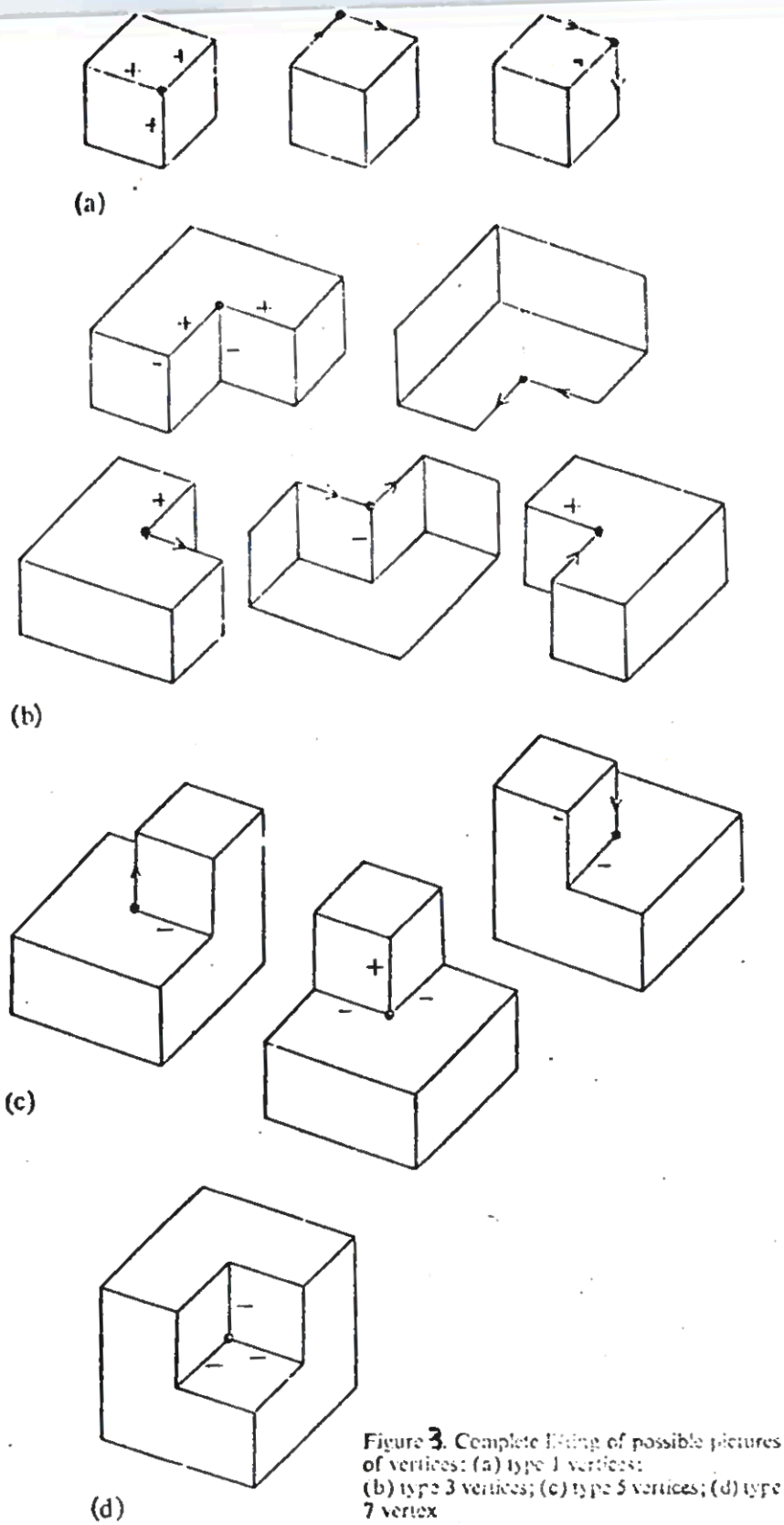


Figure 3. Complete listing of possible pictures of vertices: (a) type 1 vertices; (b) type 3 vertices; (c) type 5 vertices; (d) type 7 vertex

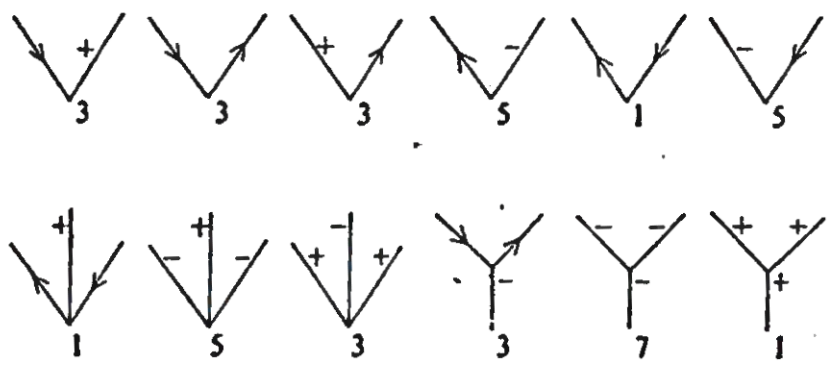


Figure 4 Possible labelled-line configurations around a picture node

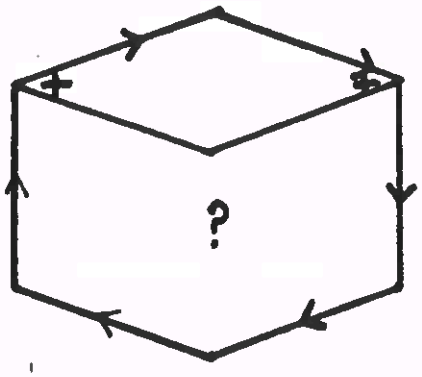


Figure 5

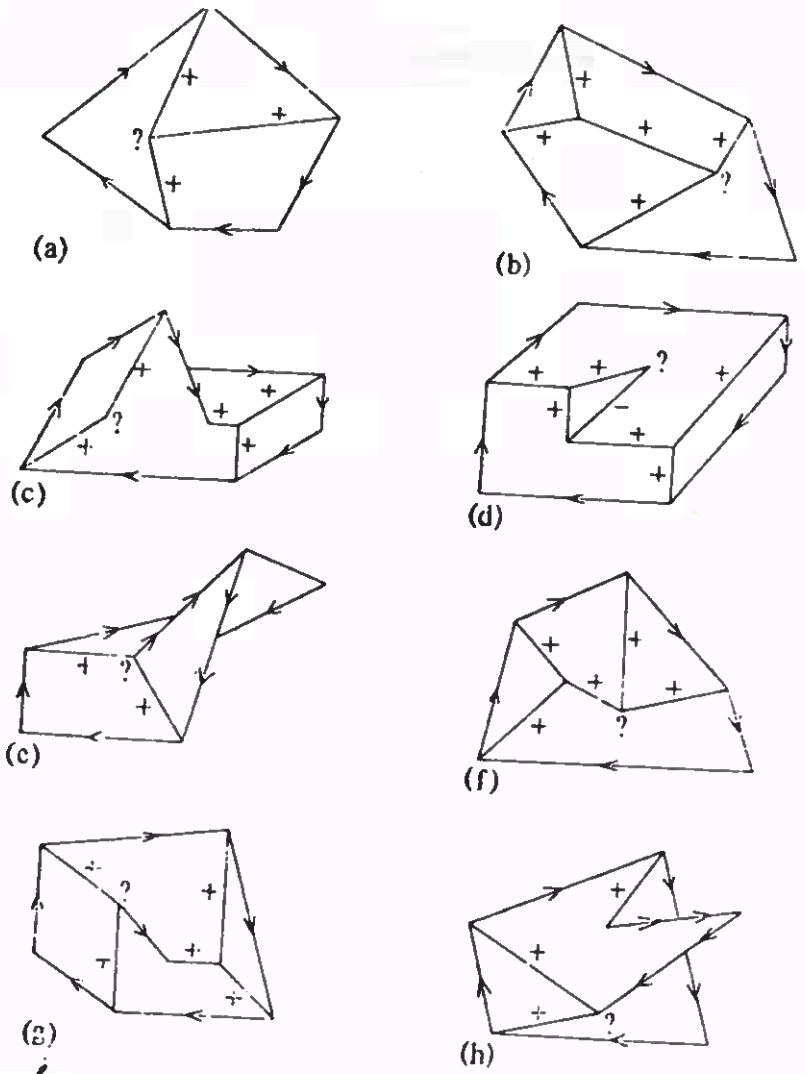


Figure 6 Pictures of objects for which there is no labelling possible

FIGURE 7

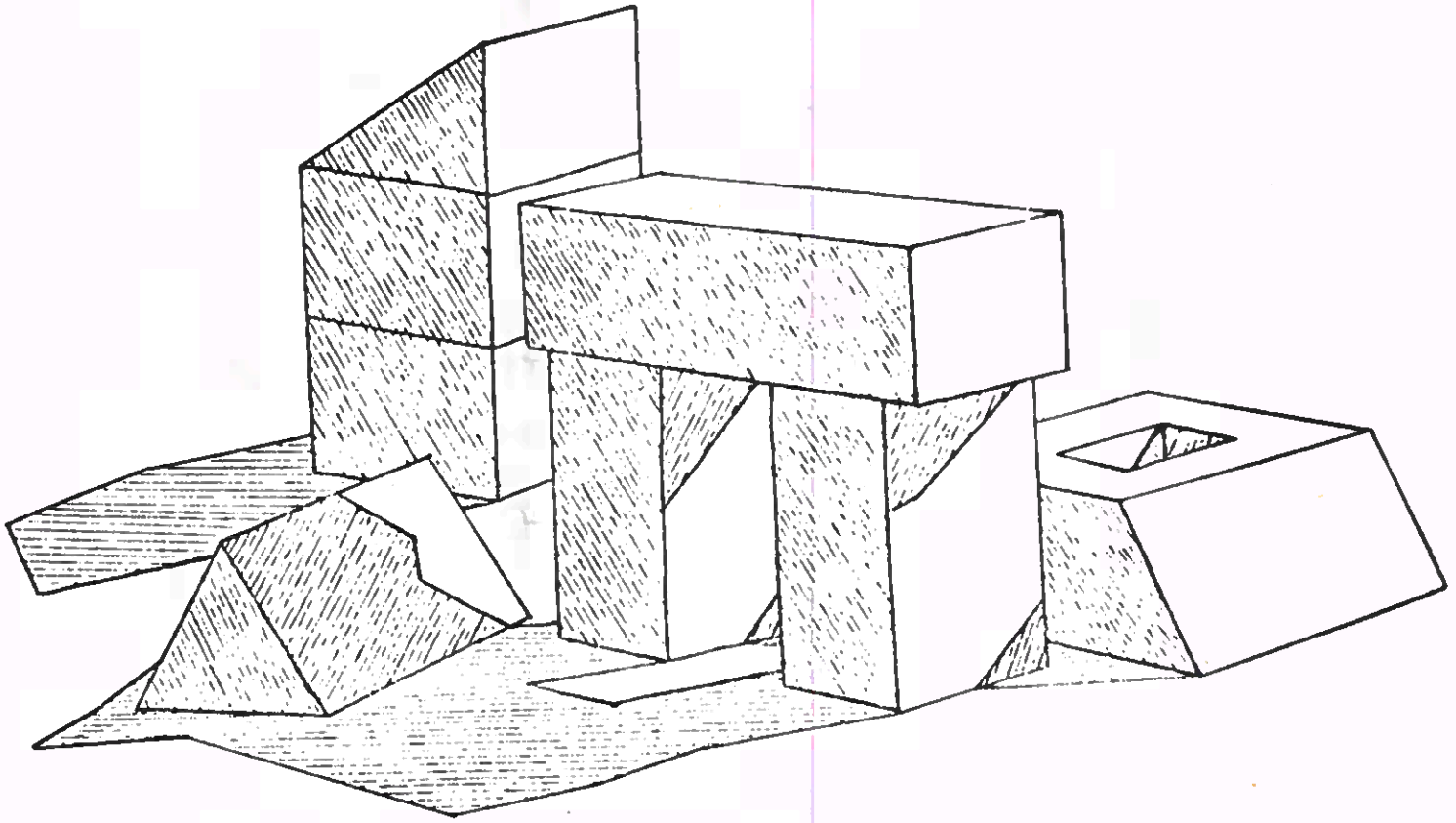


FIGURE 8

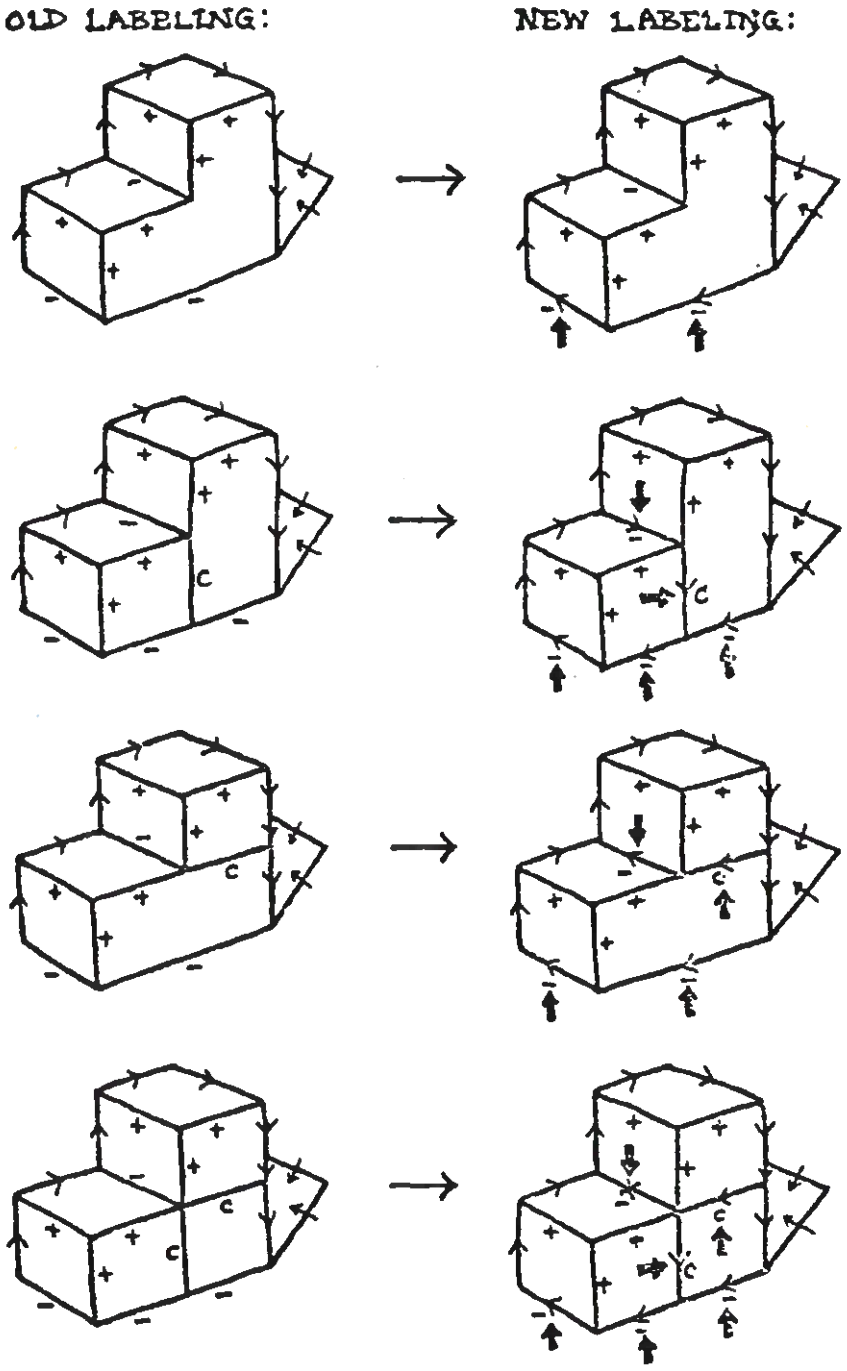


FIGURE 9

INTERPRETATION:

R1 / R2 — AN INSEPARABLE CONCAVE EDGE; THE OBJECT OF WHICH R1 IS A PART [OB(R1)] IS THE SAME AS [OB(R2)].

R1 / R2 < A SEPARABLE TWO-OBJECT CONCAVE EDGE; IF [OB(R1)] IS ABOVE [OB(R2)] THEN [OB(R2)] SUPPORTS [OB(R1)].

R1 / R2 > SAME AS ABOVE; IF R1 IS ABOVE R2, THEN [OB(R2)] OBSCURES [OB(R1)] OR [OB(R1)] SUPPORTS [OB(R2)].

R1 / R2 X A SEPARABLE THREE-OBJECT CONCAVE EDGE; NEITHER [OB(R1)] NOR [OB(R2)] CAN SUPPORT THE OTHER.

R1 / R2 C> A CRACK EDGE; [OB(R2)] IS IN FRONT OF [OB(R1)] IF R1 IS ABOVE R2.

R1 / R2 C< A CRACK EDGE; [OB(R2)] SUPPORTS [OB(R1)] IF R1 IS ABOVE R2.

SEPARATIONS:

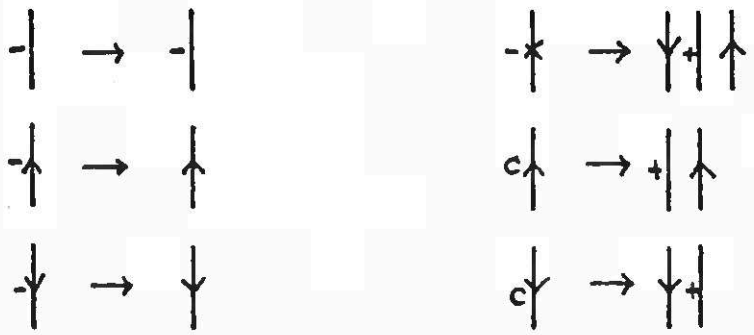


FIGURE 10

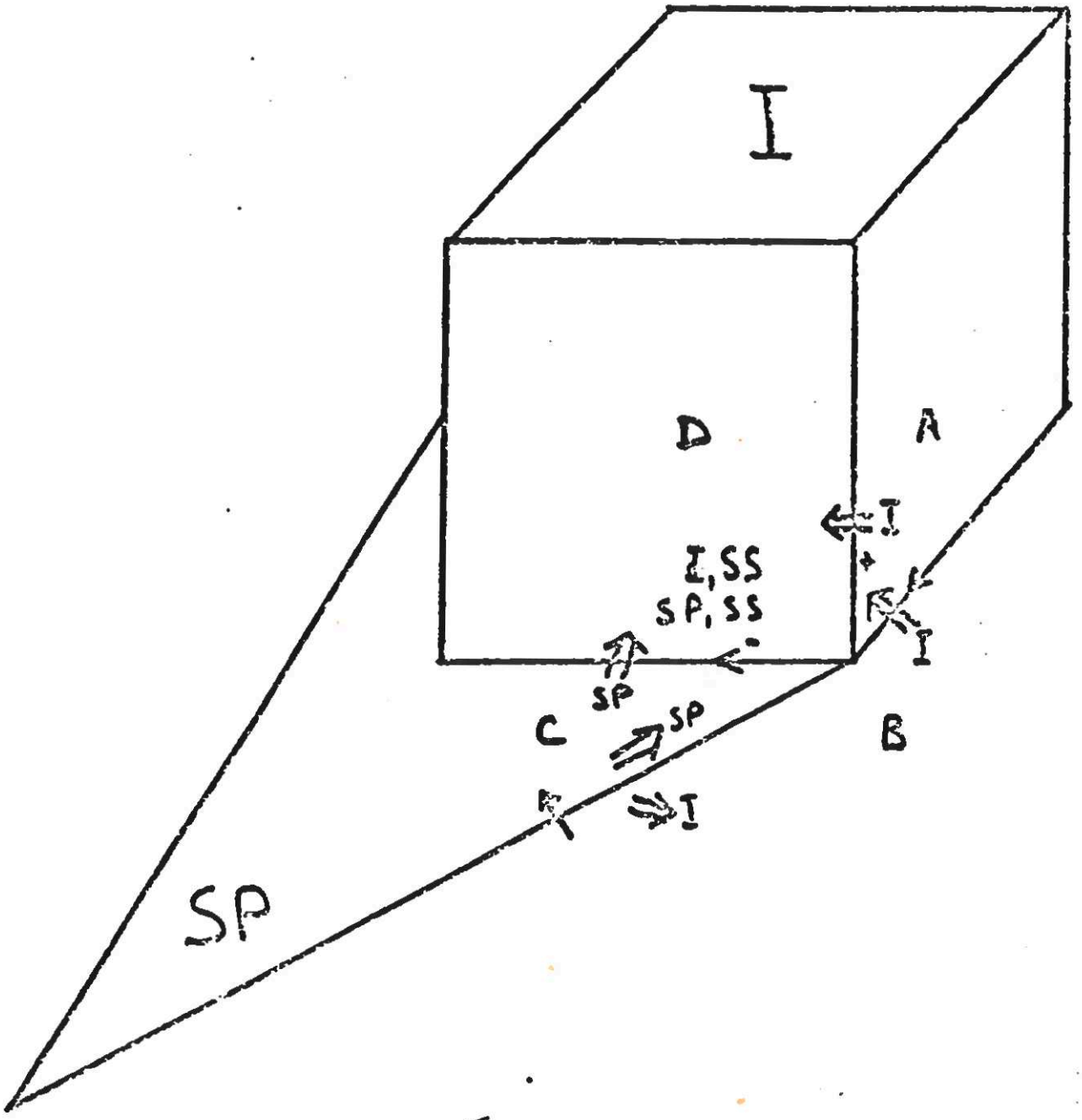
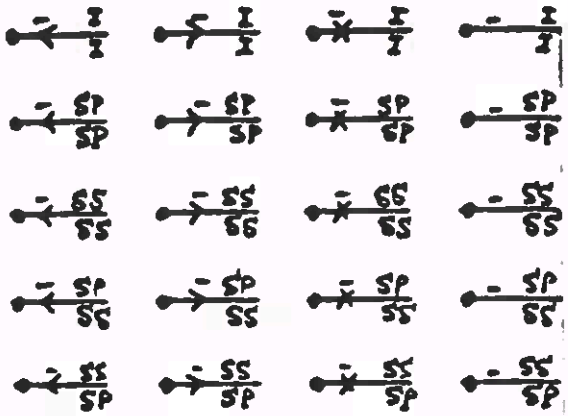


Figure 11.

CONCAVE (ANY)

1 | 2 (−)

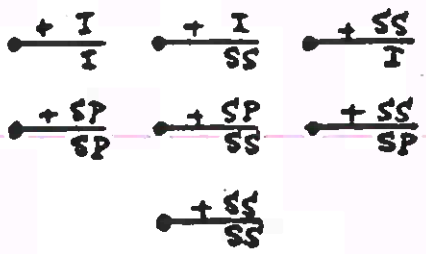
1 \ 2	I	SP	SS
I	YES	No	No
SP	No	YES	YES
SS	No	YES	YES



CONVEX

1 | 2 (+)

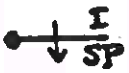
1 \ 2	I	SP	SS
I	YES	No	YES
SP	No	YES	YES
SS	YES	YES	YES



SHADOW C.C.*

1 | 2 (→)

1 \ 2	I	SP	SS
I	No	YES	No
SP	No	No	No
SS	No	No	No



SHADOW CL.*

1 | 2 (←)

1 \ 2	I	SP	SS
I	No	No	No
SP	YES	No	No
SS	No	No	No

*C.C. = COUNTERCLOCKWISE;
CL. = CLOCKWISE

Figure 12

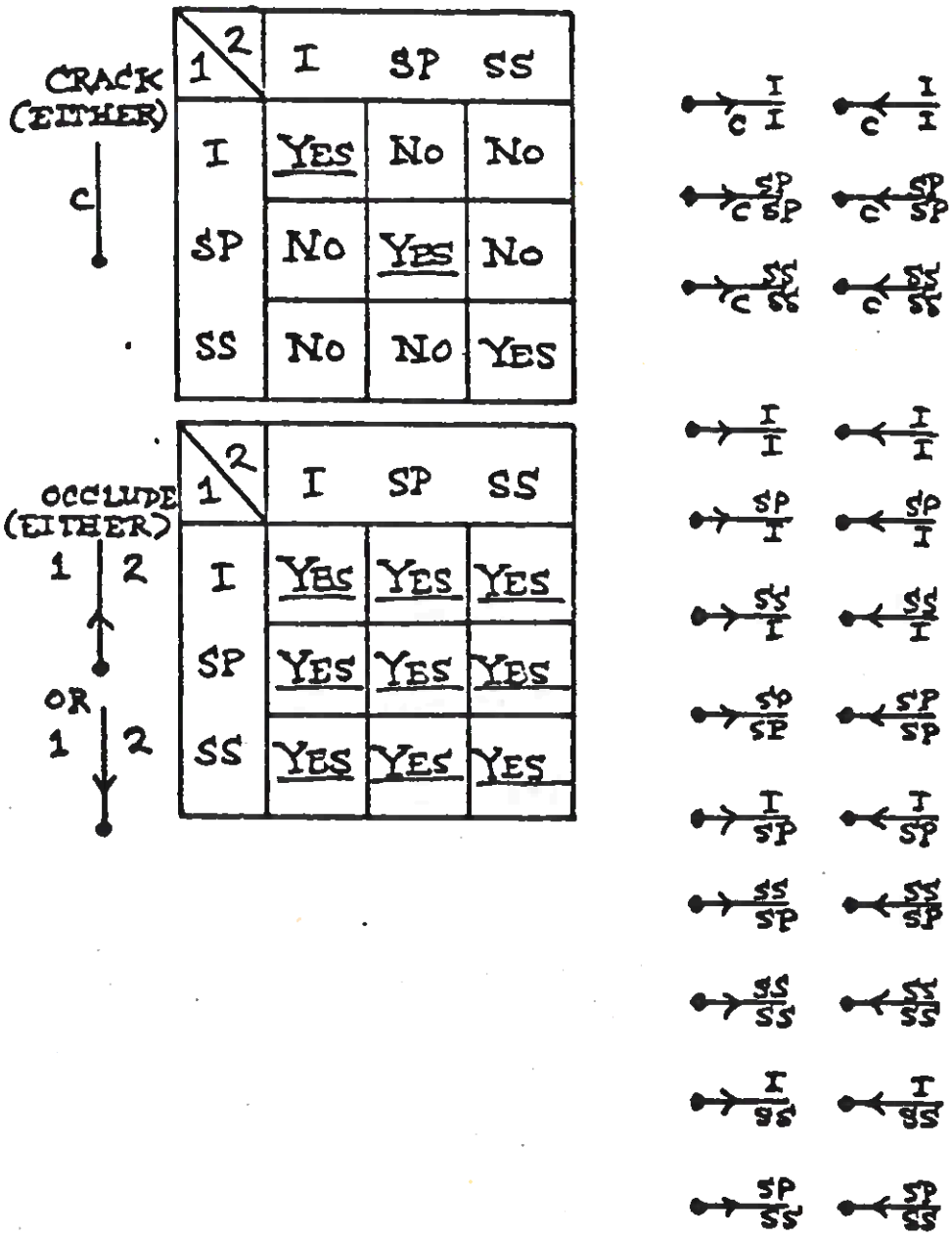


Figure 12 (cont'd)

CAN ONLY BE LABELED IN ONE OF THE FOLLOWING WAYS:

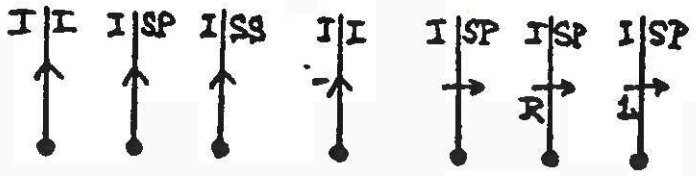


Figure 14

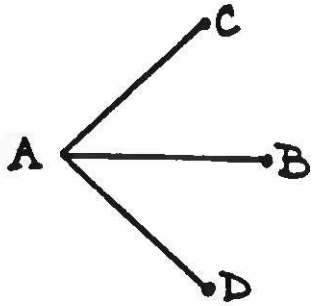
TOTAL NUMBER OF LABELS IN DATA BASE FOR EACH JUNCTION TYPE		
JUNCTION TYPE	# OF LABELS BEFORE ADDING REGION ILLUMINATION	# OF LABELS INCLUDING REGION ILLUMINATION
L	24	92
ARROW	24	86
T	91	623
FORK	116	826
PEAK	10	10
K	42	213
X	129	435
XX	40	128
MULTI	96	160
KA	20	20
KX	60	76
KXX	25	121
SPECIAL	40	466
TOTALS	717	3256

Figure 13

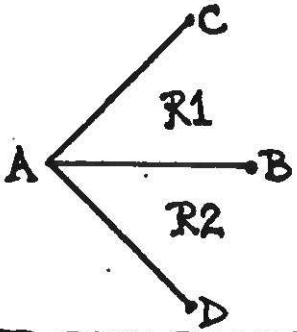
	TOTAL NUMBER OF TRIHEDRAL JUNCTION LABELINGS WHICH CAN APPEAR ON:	
TYPE OF JUNCTION	THE INTERIOR OF A SCENE:	THE SCENE/ BACKGROUND BOUNDARY:
L	92	16
ARROW	86	12
T	623	96
FORK	826	26
PEAK	10	2
K	213	2
X	435	72
XX	128	3
MULTI	160	8
KA	20	—
KX	76	8
KXX	121	—
SPECIAL	466	—
TOTALS	3256	245

Figure 15

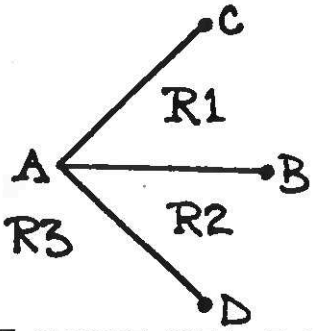
57 LABELS ARE POSSIBLE FOR L-A-B



ONLY 19 LABELS ARE POSSIBLE FOR L-A-B IF IT IS KNOW TO BE THE MIDDLE BRANCH OF AN ARROW.



IF THE BRIGHTNESS IS KNOWN FOR R1 AND R2, THEN NO MORE THAN 18 AND AS FEW AS 15 LABELS WILL REMAIN POSSIBLE.



IF THE BRIGHTNESS OF R3 IS ALSO KNOWN, THEN AS FEW AS 5 AND NO MORE THAN 18 LABELS WILL REMAIN POSSIBLE.

Figure 16

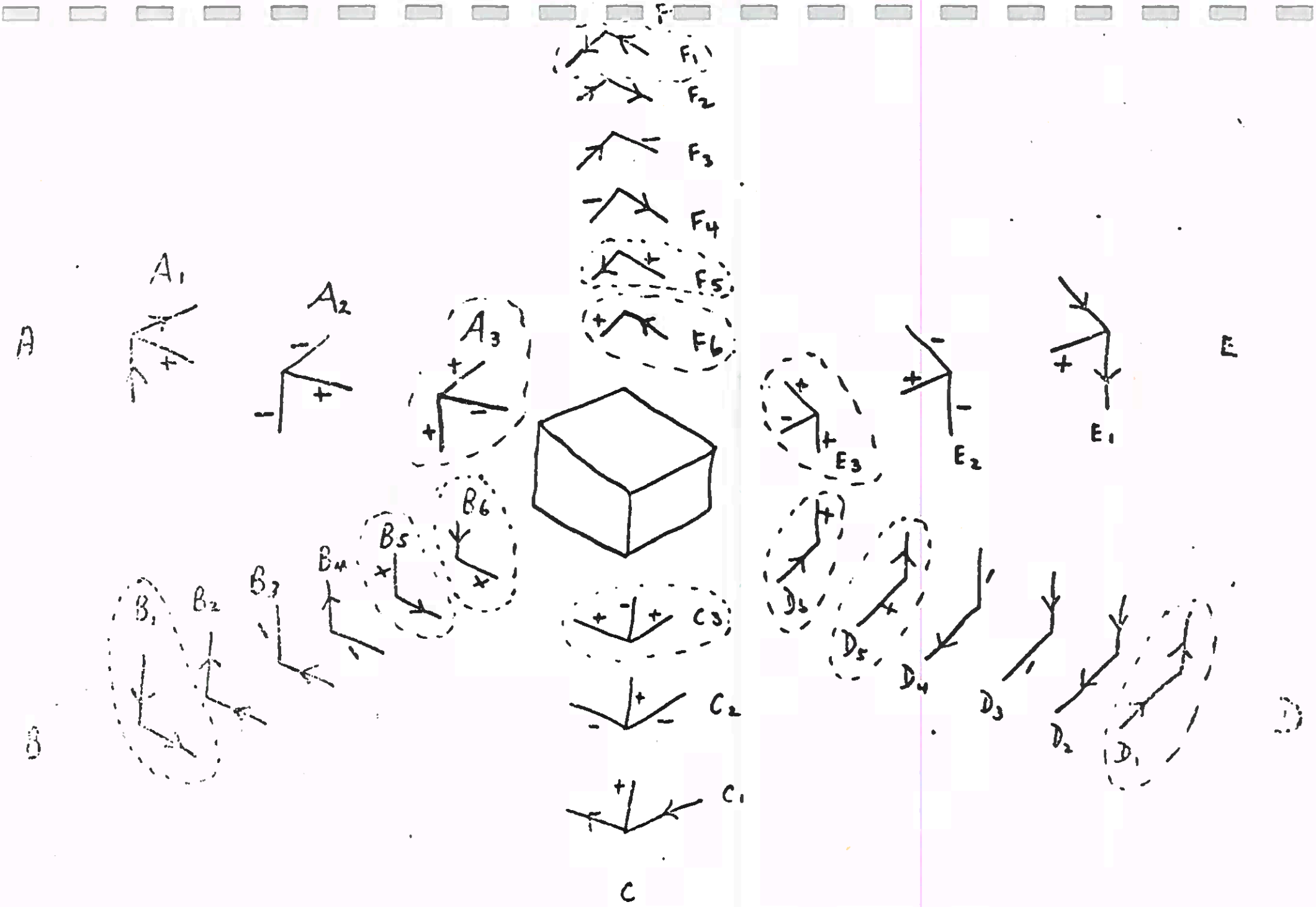
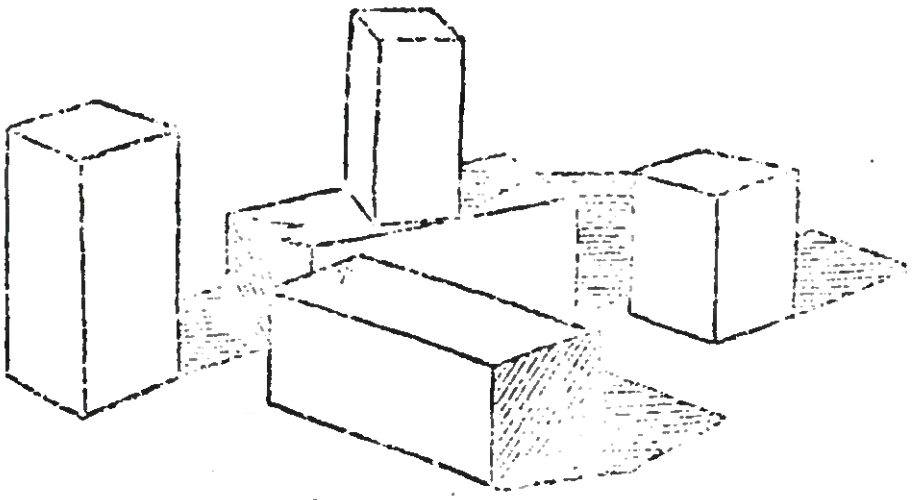
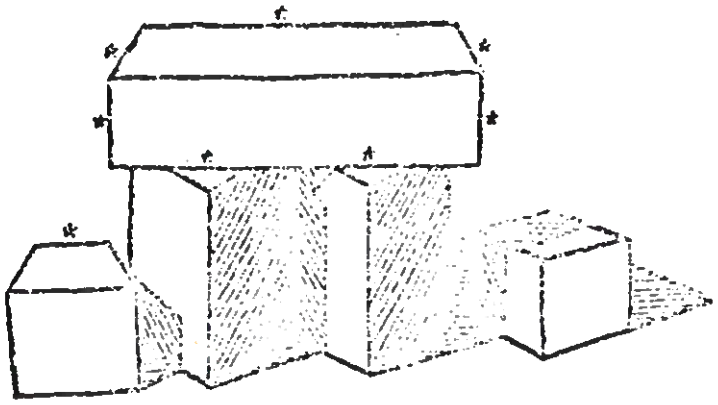

F
F1
F2
F3
F4
F5
F6
A
A1
A2
A3
B
B1
B2
B3
B4
B5
B6
C
C1
C2
C3
D
D1
D2
D3
D4
D5
E
E1
E2
E3

Figure 17



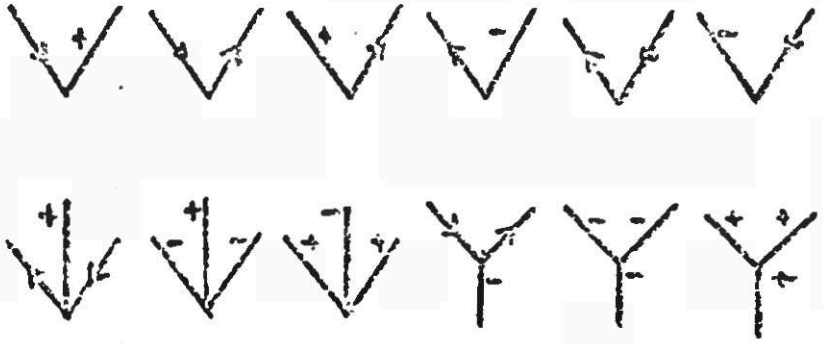
(?? SECONDS)



(48 SECONDS)

Figure 18

Guzman's Vertices



Huffman's Vertices

Figure 19

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

AI 2

JIM HOWE

Applying labelling to curved objects

Here we describe work by Turner (1974).

The consequences of handling surfaces which are not planar are as follows:

1. Surfaces. One cannot tell from a line drawing just what type a surface is. Curved surfaces may be locally described as parabolic (e.g. cones, cylinders), elliptic (e.g. spheres) or hyperbolic (e.g. torus) but many change from one type to another without any trace appearing in the line drawing. The simple geometric properties of planes and the absence of any other surface type contributed significantly to the success of Waltz's program.

2. Edges. Curved surfaces introduce another edge type, known as the generator edge (Figure 1). It corresponds to some invisible line in the surface (around outside of sphere, cylinder etc.). A disturbing feature of curved edges is that they may undergo transition in interpretation without any visible change (Figure 2). Waltz's program relies on lines being ascribed the same interpretation by the junctions at each end: this is not true in general. Finally, polyhedral scenes give rise to lines connected at both ends to other lines, with each pair of junctions connected by at most, one line. Curved objects also violate these rules, as shown in Figure 3.

3. Corners. Junctions may be attached to only one line, and may be joined by more than one line (Figure 3b). There can even be lines without junctions (outline of sphere). Generator edges give rise to a new kind of junction, where a generator meets another generator or a true edge (Figure 3(b)).

4. Illumination. The reflected intensity is (ideally) the same at all points of a plane surface. This is a powerful binding force in Waltz's program since all the edges round a surface must assign it the same illumination label. In contrast, a curved surface may receive all possible types of illumination (Figure 4). Also shadow lines may be non-existent (e.g. in Figure 4, the line separating the illuminated and non-illuminated halves of the cylinder) or may peter out along their length (e.g. in Figure 4, the shadow cast across the cylinder).

These problems indicate that curved objects cannot be treated by a straightforward extension of polyhedral techniques. Instead, various restrictions were imposed.

1. Background — The background is assumed to be an illuminated plane with no holes.

2. Lighting — Illumination is by a single point-source of light (producing sharp shadow edges).

3. Viewpoint — The scene is contained wholly within the picture, with fixed viewpoint.

4. Surfaces — Surfaces are smooth and opaque: creases are not permitted. Surface points must all be of the same type (i.e. parabolic, elliptic or hyperbolic). No surface marking is allowed.

5. Cracks — Cracks are not permitted so separable edges are not handled. (Cracks are rare with curved objects).

6. Corners — Only corners enclosing a single volume in space are allowed.

The mechanism for generating curved object function labels is based on the observation that two planes may approximate a curved surface in the vicinity of a corner. A corner composed of both plane and curved surfaces may therefore be approximated by a purely polyhedral one. Note that a convex (concave) surface will give rise to a convex (concave) edge, and that the convexity (concavity) of the other edge will be preserved. This process may be applied in reverse, a polyhedral corner being regarded as generating one with curved surfaces. The fact that convexity and concavity are preserved means that the labels of the non-planar corner can be easily derived from those of the planar one. To determine the labels for a certain class of curved objects, the procedure is to obtain the labels for the appropriate polyhedral corner and apply a plane-to-curved transformation. For example, in Figure 5, a 3-positive FORK (depicting the corner of a cuboid with all the surrounding faces visible) can be transformed into a 2-positive curve with a *notional* junction in the middle, the stem of the fork having got lost as the cuboid is transformed into a cylinder with no vertical edges. So, in a sense, the program has not only to identify and label vertices explicitly present in the picture, it also has to "see" junctions that are not there as such.

In practice, junction labels must be generated for the cases of corners, tees, shadowed corners, shadowed tees, shadows cast on surfaces and shadows cast across edges. Junction labels have also been derived for certain interactions between planar, conical, cylindrical and elliptical surfaces. Some typical labellings are shown in Figure 6.

As indicated above, the illumination over a curved surface may vary from directly-illuminated to self-shadowed. Also shadows cast over a convex surface may peter out. This means that illumination information

must be associated with lines, in the neighbourhood of junctions, rather than with areas of the picture. This does not eliminate the problem since the nature of the illumination may vary at opposite ends of a line, as may the types of edge. The solution is to relax the consistency requirement that the interpretation of a line must be the same at all points along it. Instead, transition rules are used which specify how illumination and edge labels can sensibly transform into others along the length of an edge. Example transitions are shown in Figure 7, together with the rules which deal with them.

Finally, Figure 8 is typical of the kind of scene that can be analysed by these methods. The program takes about four times longer than Waltz's program does to analyse a polyhedral scene. This speed difference stems from the increased size of the label data-base, the greater complexity of the consistency rules, and the diminished value of illumination information. Indeed, ignoring illumination information does not give rise to much ambiguity with curved objects: consistency of surface type is the main cohesive force. But irregularly curved surfaces would be more difficult to handle, and illumination might become a significant cure once again if Turner's program was expanded.

Reference

Turner, K J (1974) "Computer perception of curved objects using a T.V. camera. Ph.D. Thesis, University of Edinburgh.

Figure 1: Generator edge labels

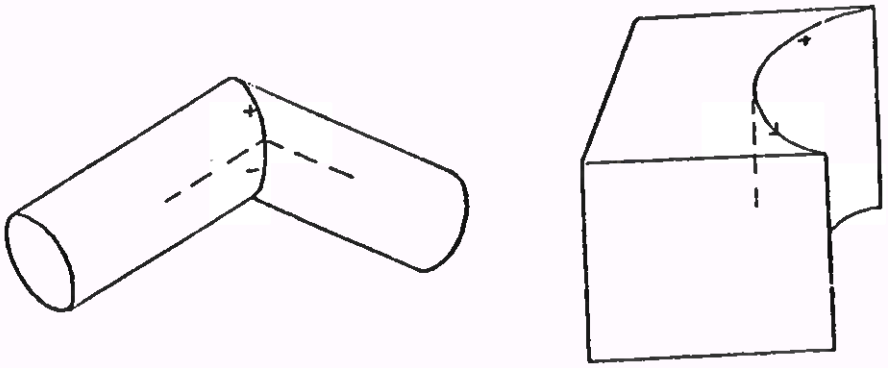
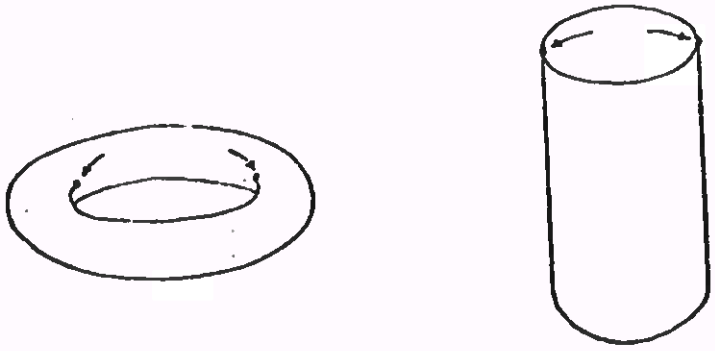


Figure 2: Edge type transitions.



(a) (b)

Figure 3.

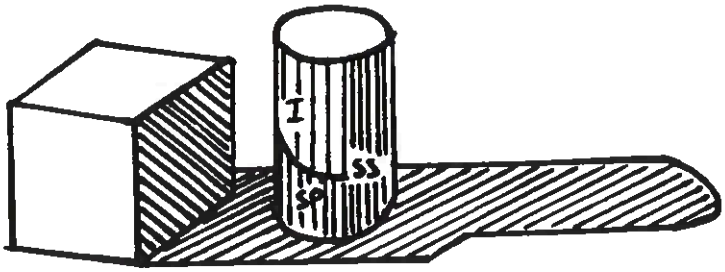
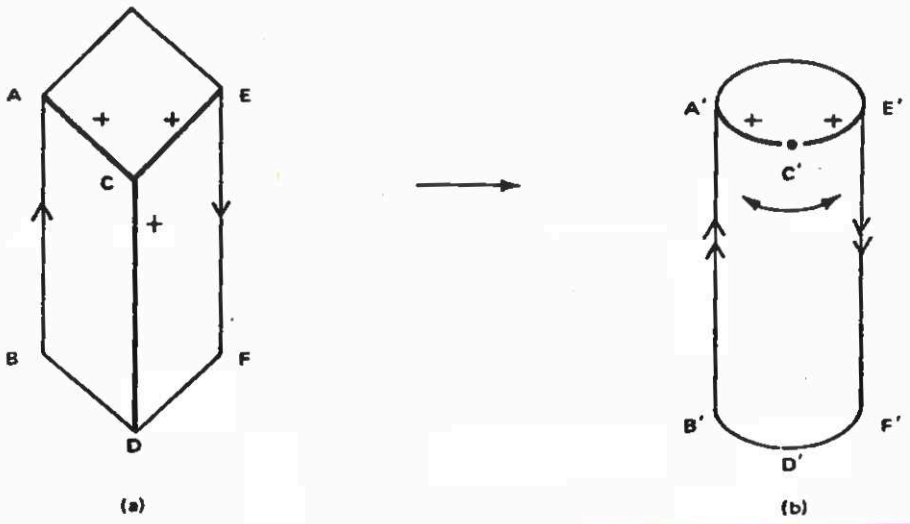
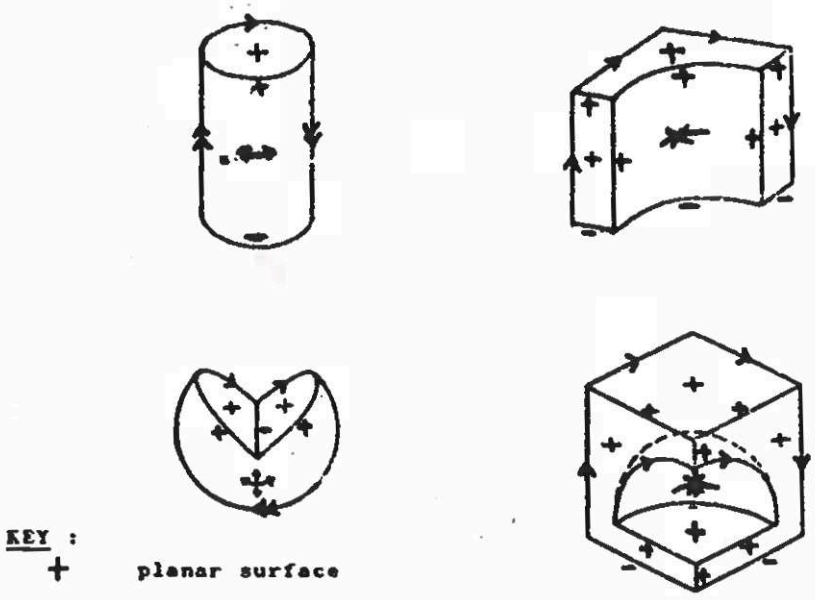


FIGURE 4.



As the cuboid is transformed into a cylinder, the actual junction at C becomes the notional junction at C'. Line CD disappears. Lines AB and EF, which are both genuine edge-lines depicting physical discontinuities in the cuboid, are transformed into lines A'B' and E'F', which (as the double arrow labels mark) do *not* correspond to "real" edges in the cylinder, but merely to the visible outline of it.

Figure 5.



KEY :

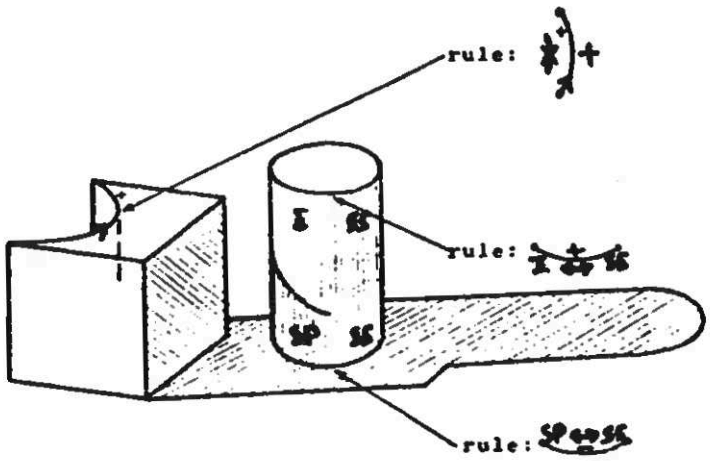
+ planar surface

convex/concave conical or cylindrical surface

convex/concave elliptical surface

Some representative labellings

Figure 6.



Some edge- and illumination-type transition rules

Figure 7.

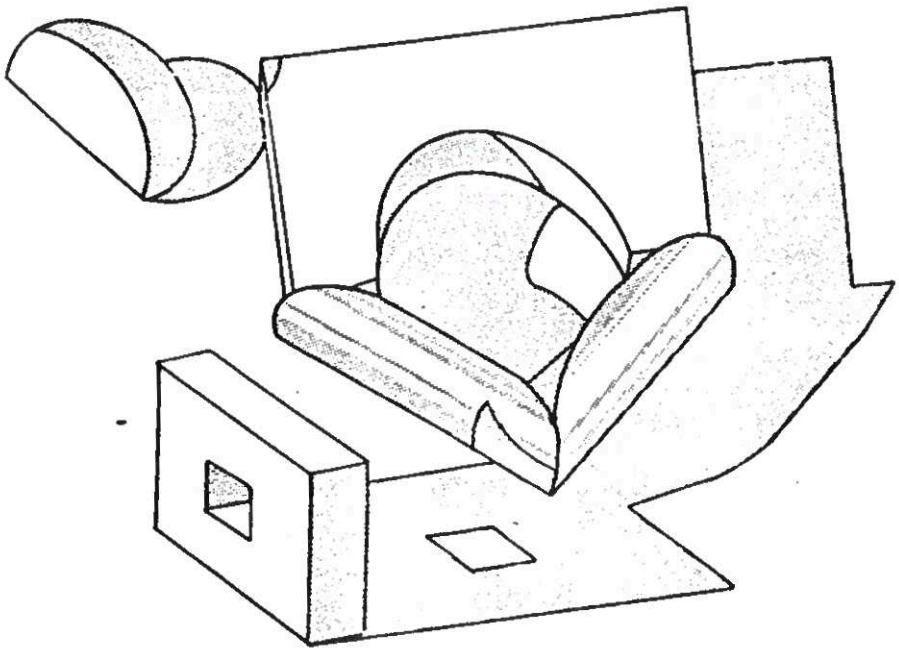


Figure 8.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

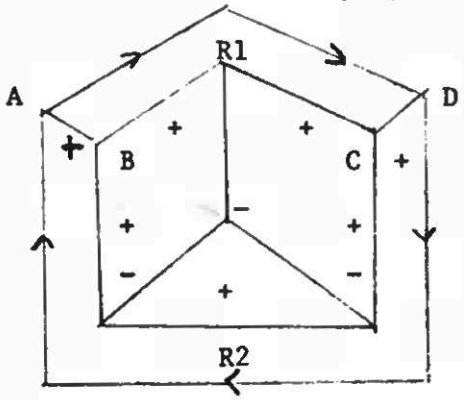
AI 2 JIM HOWE

Reasoning about Surface Orientations

Line labels are subject to two limitations.

1. Line labels provide a qualitative shape characterization. While the convex label + indicates that two plane surfaces meet and form a convexity, it is unspecific about the angle at which the planes meet.

2. Legal labellings can be generated for line representations that are not realizable as polyhedra. For example,

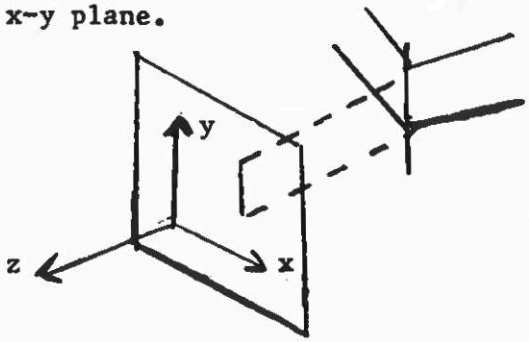


R1 and R2 cannot make convex edges at both AB and CD

In addition to the syntactic constraints introduced by the labels, geometric constraints are required. The use of gradient space for this purpose was proposed by Huffman and used by Mackworth in his POLY program. Before describing POLY, we will define the gradient space.

Gradient space

We begin by assuming that the viewer is at the origin of the projection system, and that z-axis is the viewer's optical axis, and that the picture plane is at z=0 and parallel to the x-y plane.



Let us denote a surface in the space as

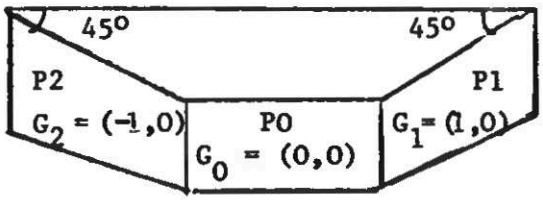
$-z = \left(\frac{a}{c}\right) x + \left(\frac{b}{c}\right) y + \frac{d}{c}$, assuming c is not 0, where -z is the depth of the surface point. An increase in -z represents an increase in the distance from the viewer.

The gradients of the z components of the points in the plane in x and y directions

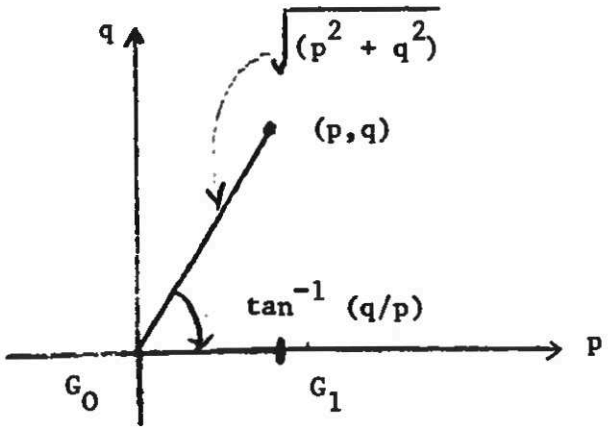
are ..... $G_x = \frac{\delta z}{\delta x} = -\frac{a}{c}$ $G_y = \frac{\delta z}{\delta y} = -\frac{b}{c}$

The gradient measures the instantaneous change in the depth of a surface at a point (x,y) or it measures the tilt of the surface at that point with respect to the z-axis. (G_x, G_y) may be viewed as a two-dimensional gradient space, the orientation of each plane in the (x,y,z) space being uniquely represented by a point in the gradient space (p,q), except when C is 0. A point at the origin of the gradient space represents a plane parallel to the (x,y) plane; a point on one of the axes of the gradient space represents a tilt of this plane along the x or y axes, and a set of points in gradient space represents a curved surface.

Consider the following illustration:



The depth (-z) of P_0 does not change, so its gradient $G_0 = (0,0)$. Plane P_1 tilts away from the viewer at 45°, so its gradient is (1,0). The gradient space representation is:



Note that the axes of this space are p and q, not x and y, where the p-axis represents surface rotations about the vertical y-axis, and the q-axis represents surface rotations about the horizontal x-axis. Combinations of p and q rotations are represented by points lying off the p and q axes, e.g. (p,q). The direction of the vector from the origin to (p,q), i.e. $\tan^{-1}(q/p)$ describes the direction of steepest change in the depth of a surface; the distance to the origin i.e. $\sqrt{(p^2+q^2)}$, is the rate of change of depth along the direction of steepest change.

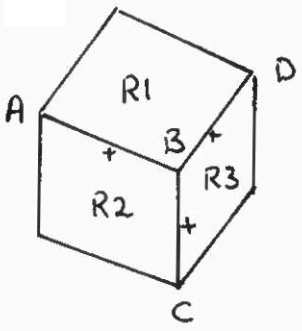
Line labels and gradients

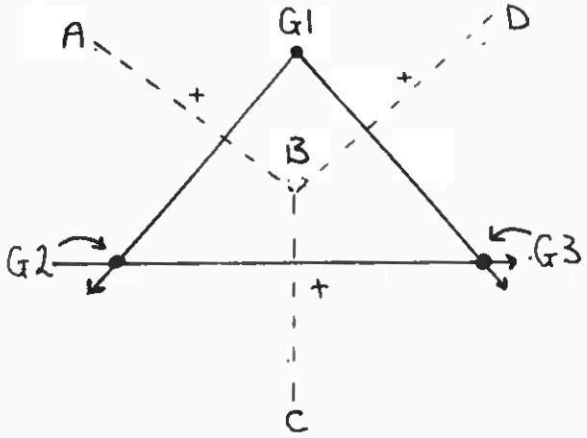
Although the gradient space representation does not make explicit the actual depth of the surface plane nor the spatial extent of the plane, it does describe the concave/convex relationship between adjoining surfaces.

For the remainder of the note, we assume an orthographic projection, i.e. no foreshortening when a point (x,y,z) in 3-D space is projected to a point (x,y) in the image plane.

The relationship between line labels and gradient space is that if two surfaces meet along a concave or convex edge, their gradients lie along a line in gradient space that is perpendicular to that edge in the image. This is known as a "dual line". For example, if two planes intersect at a vertical edge in the image plane, the gradients of the two planes must lie in a horizontal line in gradient space. Furthermore, if the gradients of the surfaces are on the same side of the edge as their causing surfaces, the edge is convex; if the gradients of the surfaces are on opposite sides of the edge from their causing surfaces, the edge is concave. (See Figure 1.)

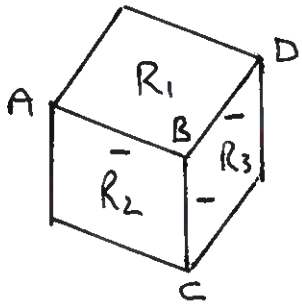
In this way, the line labels can be related with properties of the gradients. For example, the convex corner (below left) is represented by gradients in the dual space representation (below right).

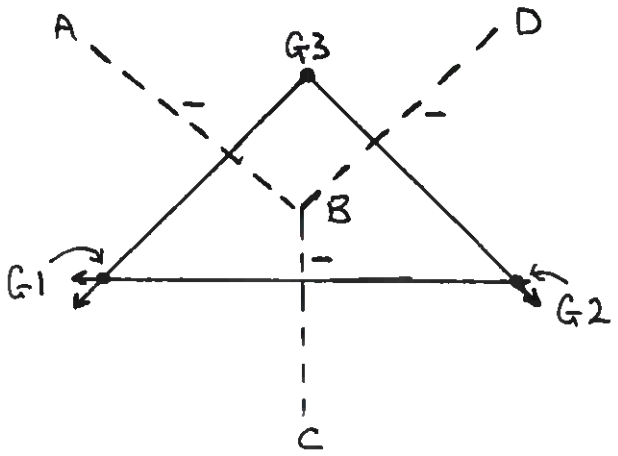




Suppose the gradient of R1 is at G1. Becuase R1 and R2 are linked by a convex line AB, the gradient of R2 must be on a line that passes through G1, perpendicular to AB. Suppose it is at G2. Region R3 is connected to both R1 and R2 by convex lines, BD and BC. So, its gradient G3 is given by the intersection of lines extending from G1 and G2, perpendicular to BD and BC respectively. Then G1, G2 and G3 form a triangle of a particular shape. In fact, the location and scale of the triangle are arbitrary, but the shape and orientation are strictly determined by the lines in the image. These are exactly the constraints that the labelling represents.

Similarly, the concave corner (below left) is represented by the dual space representation (below right).

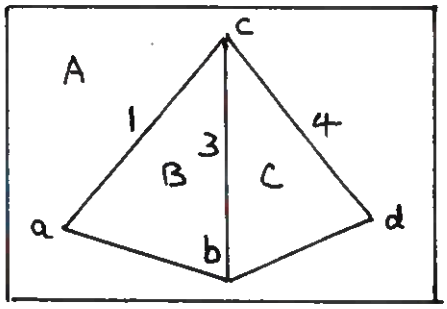




POLY

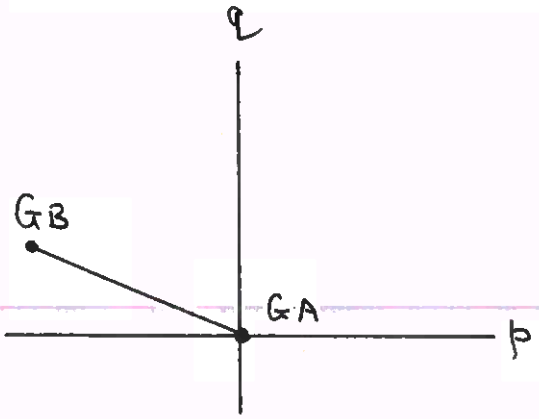
For an object to be physically realizable it should be possible to construct a consistent dual for it. This is the task tackled by Mackworth's program, POLY. Note that POLY checks for consistency only over connect (convex or concave) labels and uses them to assign the sense of occlusion to non-connecting edges. This is because connect labels are much more constraining than occlusion labels: reasoning about connect labelling involves the solution of numerically simple simultaneous equations whereas reasoning about occlusion involves consistency checking of relational structures expressing such concepts as 'in front of' and 'behind' which are not readily represented in the 2D of gradient space.

Let's see how POLY handles this example:

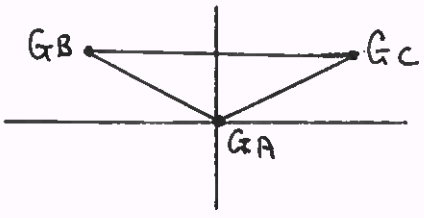


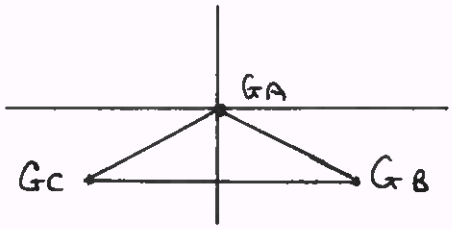
It begins with the background surface, A, and arbitrarily assigns a gradient (0,0) (origin of gradient space) to it. Next, it takes region B. Surfaces A and B are bounded by lines 1 and 2. Line 1 is considered. If it is a connect edge (either concave or convex), the gradient of B must be on a gradient-

space line (perpendicular to line 1) that passes through the gradient of A. The gradient of B is placed at unit distance from the origin (the origin and scale are arbitrary):



Next, line 2 is considered. To establish it as a connect edge, GB must lie on a line perpendicular to 2 through GA. But this contradicts the previous situation where GB lies on a line perpendicular to 1 . Thus line 2 is not a connect edge, but an occluding edge (assuming line 1 is a connect edge). Next, line 3 is considered. If line 3 is a connect edge, GC must lie on a line perpendicular to line 3 through GB. Region C shares lines 4 and 5 with region A. So both cannot be connect edges. The interpretation in which lines 1, 3 and 5 are connect edges and 2 and 4 are occluding edges is rejected by the rule that three non-collinear points in space (corners a, b, c) cannot simultaneously lie on two planes (A and B). So a legal interpretation is that lines 1, 3 and 4 are connect edges, but lines 2 and 5 are occluding edges. Now the situation in gradient space is one of two possible cases:





Now, POLY describes convexity or concavity of connect edges by making use of the gradient-space constraints established so far. For the left-handed case (above), lines 1 and 4 are concave and line 3 is convex, whereas, for the right-hand case, lines 1 and 4 are convex and line 3 is concave.

Finally, POLY looks at the non-connect edges (lines 2 and 5). If edges 1 and 4 are concave and 3 is convex, on the right side of line 1, B is always in front of A. Because of this, occluding edges are known to belong to surfaces B and C respectively.

POLY yields one other interpretation, in which lines 2, 3 and 5 are connect edges and 1 and 4 are occluding. Thereafter, it yields interpretations with fewer connect edges, e.g. line 3 as connect and lines 1, 2, 4 and 5 as occluding.

Whereas the gradient space method eliminates some of the impossible objects that were correctly labelled by Waltz's program, such as those shown in Figure 2, it is not foolproof. For example, Figure 3 is an instance of an impossible object that is accepted, using gradient space, because its inconsistencies are not captured by the technique. The essential difference between Figure 2b and Figure 3 is that the edges on either side of the notch in the latter shape are parallel. The gradient space method does not distinguish between the duals of parallel and collinear lines because (i) the gradient space does not represent parts of plane surfaces, and (ii) dual lines represent only the orientation of edges. Hence, surfaces A and B are represented as points on a single dual line.

Reference

Nevatia, R. Machine Perception, Chapter 4.

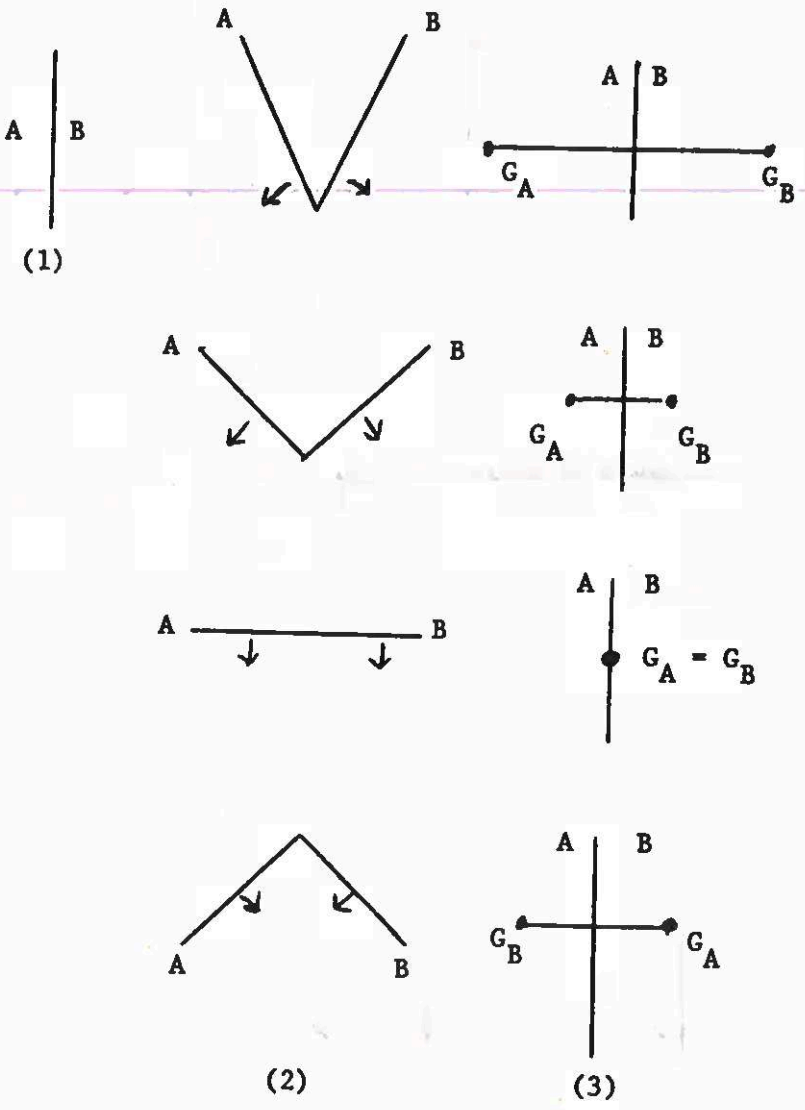


Figure 1

(1) Image

(2) Surface arrangement (plan)

(3) Gradients

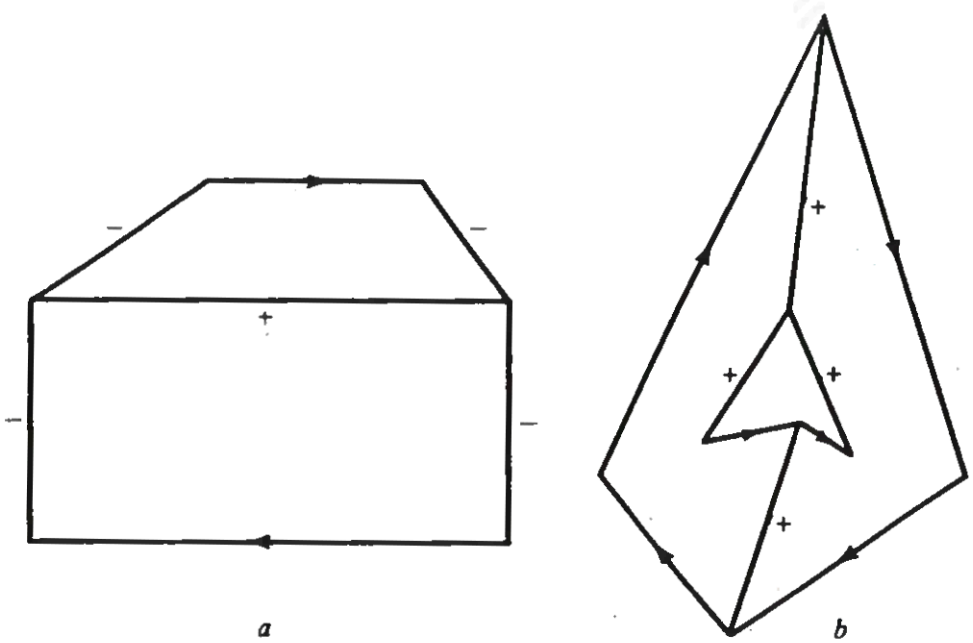


Figure 2 Labelling problems for Waltz's algorithm. (*a*) Anomalous (physically impossible) interpretation of a possible object. (*b*) False acceptance of an "impossible object." (*Source:* Adapted from Huffman, 1971.)

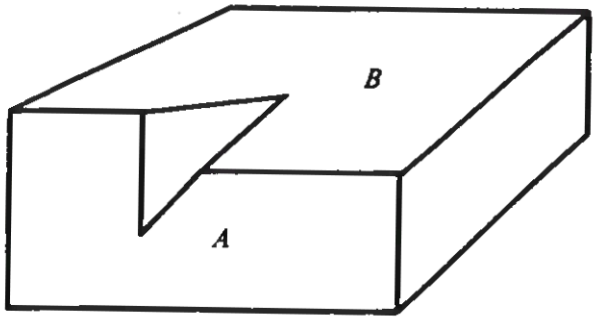


Figure 3 An impossible figure "accepted" by gradient space. (*Source:* After Huffman, 1971.)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

UNIVERSITY OF EDINBURGH

Knowledge guided segmentation

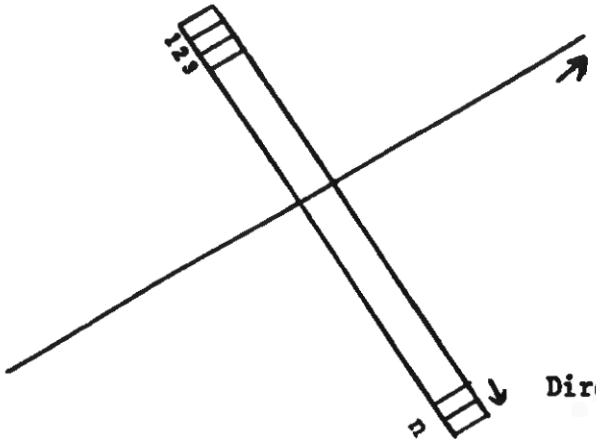
The programs which we have been reviewing so far have a hierarchical structure. First, they find feature points in an entire scene; next they make a complete edge representation using these feature points; finally, they segment the edge description to form separate bodies. This approach is susceptible to errors in the early stages (due to noise etc.); so later analysis based on the earlier results is likely to lead to serious mistakes.

What we will look at now is an attempt to overcome the limitations of the hierarchical structure, replacing it with a more flexible structure which abandons the rigid ordering in favour of a strategy of using knowledge to guide the processing. The particular research that we will be considering was carried out at M.I.T. by Shirai.

More about detecting edges

In the work considered previously, we assumed that the edges of bodies were represented as step (or near step) edges in the grey level representation. This was an over simplification. Typically a blocks world scene contains a variety of different kinds of edges. Besides the object background boundary edges, usually of high contrast (our step edge of before), there are lower contrast (more blurred) internal edges between adjacent surfaces of an object, and so on. The operators considered previously (e.g. Robert's cross, high pass filter) were designed to suit step-like edges: they perform relatively poorly on blurred edges which are characterised by luminance gradients which extend over a larger area of the grey scale representation. If we had a method that could factor out the different kinds of edges from the visual data, this information could be used by higher level processes to segment the scene into bodies, without invoking a labelling process. In fact, Shirai's program does exactly this. Let's examine how it does it.

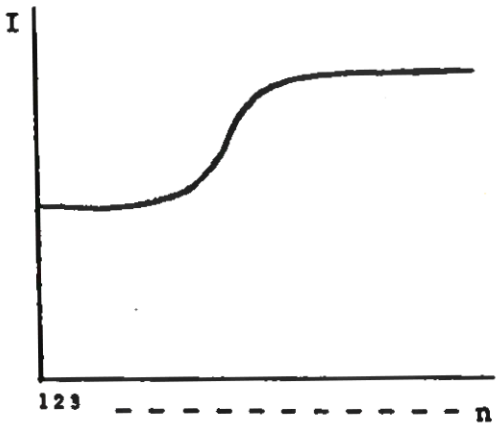
Consider an intensity profile of n points taken along a band perpendicular to the direction of a step edge:



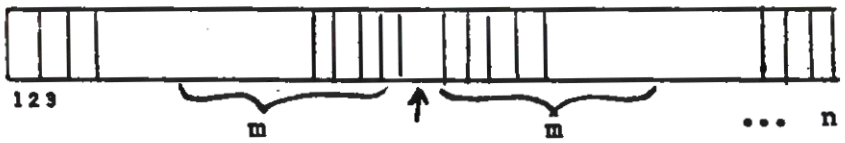
Direction of line segment

Direction of intensity profile

The intensity profile is represented as follows



The contrast function is calculated as follows. We define the contrast function of the i^{th} profile point, to be the difference between the sum of the m subsequent points and the sum of the m preceding points, where m is a parameter. Graphically,

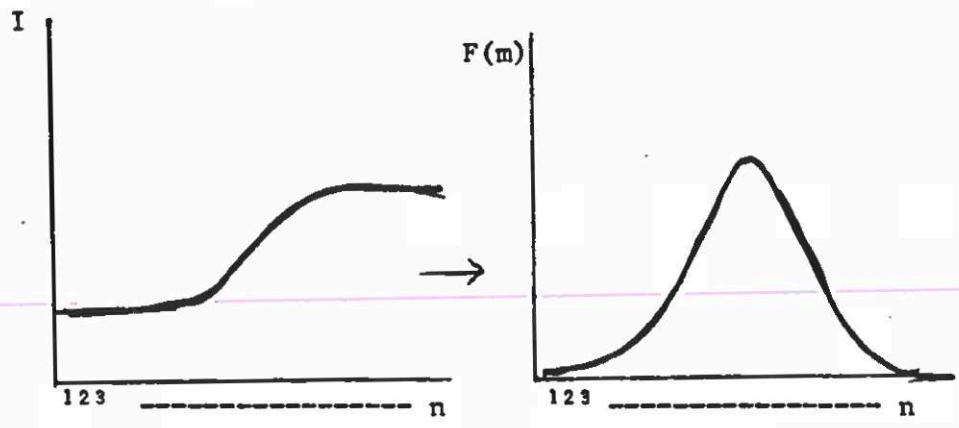


i^{th} profile point

$$S_1 = \sum_{r=i-m}^{i-1} \qquad S_2 = \sum_{r=i+1}^{i+m}$$

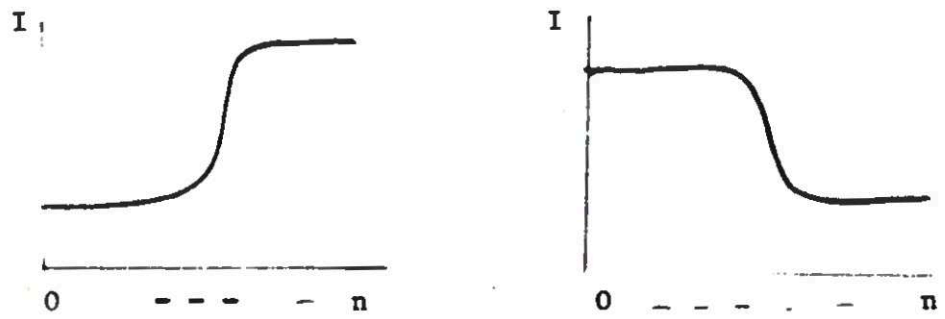
The contrast function at the i^{th} profile point is F(m) where $Fm(i) = S_1 - S_2$

But there are (n-2m) points for which values will be obtained, yeilding a contrast function of the edge.

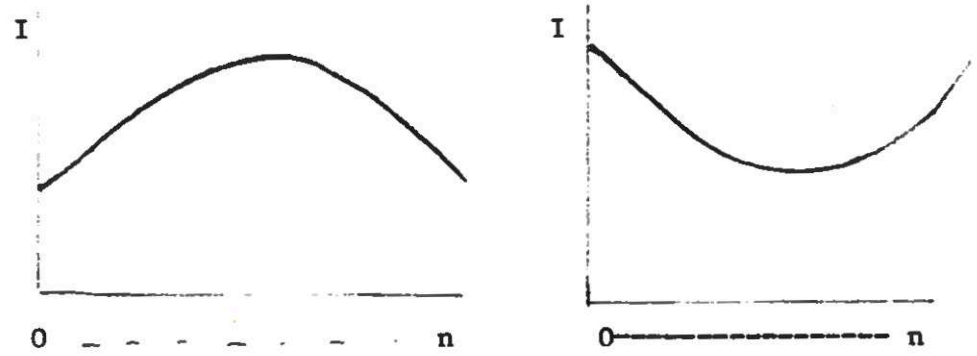


Intensity function Contrast function

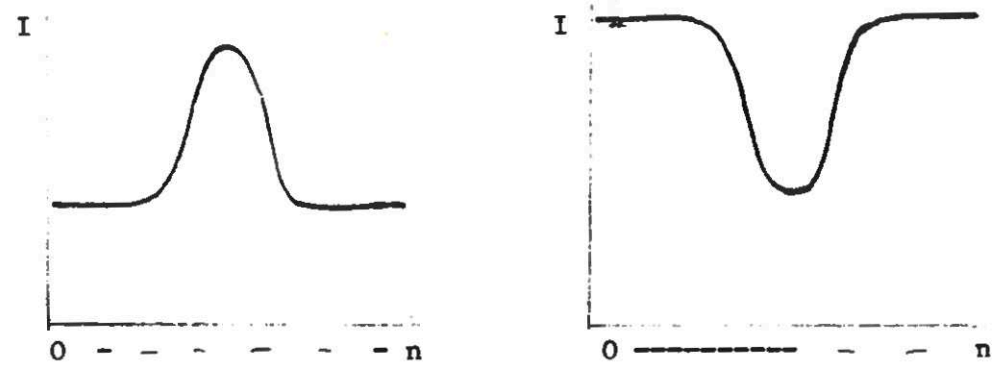
So much for the step edge, but as pointed out above not all real edges have similar cross-sectional intensity (luminance) profiles. Herskovits and Binford classified edges into three types, namely, step, roof, peak (or spike), according to the shape of their intensity profiles.



step



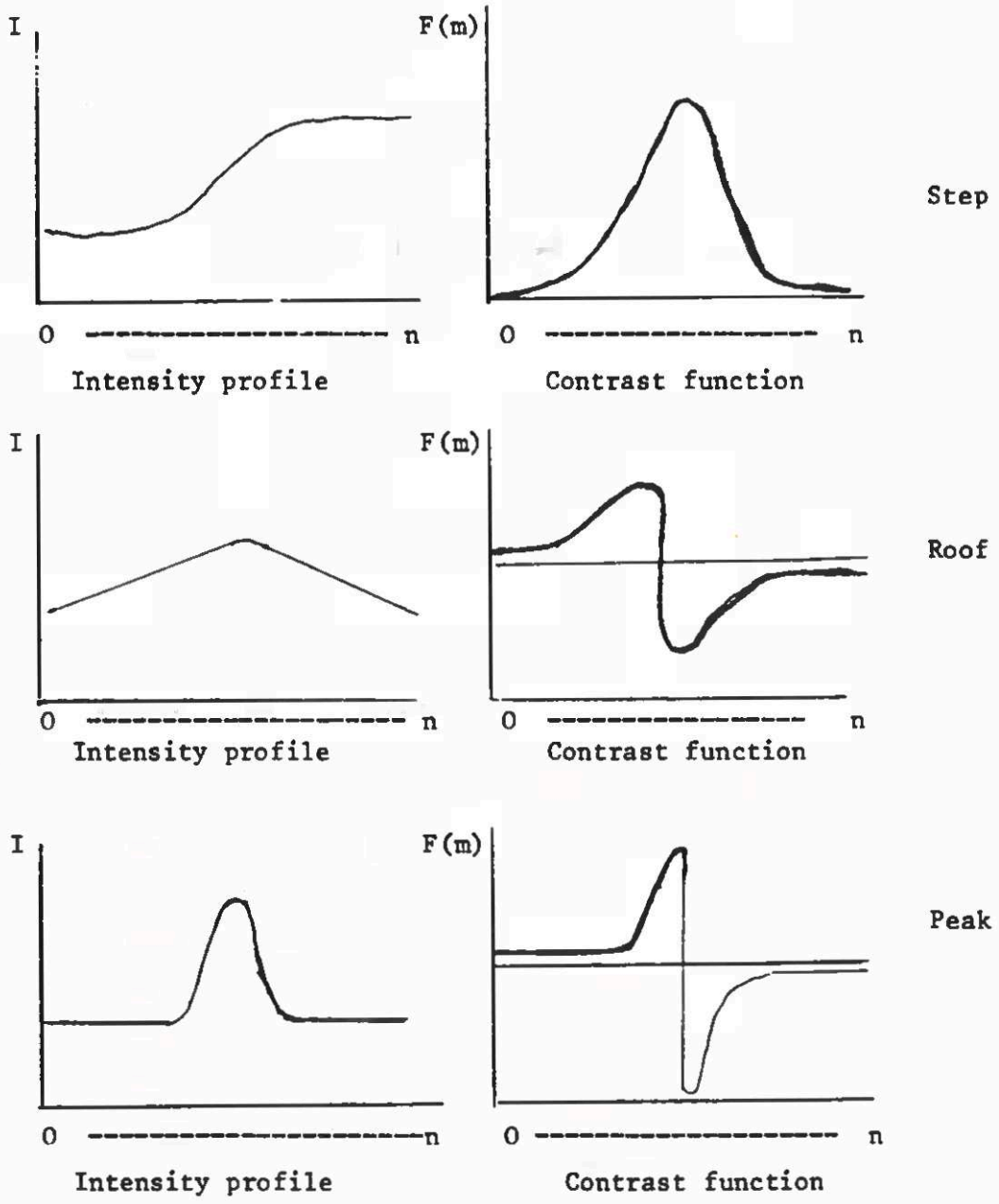
roof



peak (spike)

Intensity profiles

The step edge (first derivative), encountered previously, occurs at a high contrast boundary between regions of relatively homogeneous intensity. The peak (or spike) edge (second derivative) occurs at boundaries representing a sharp highlight, or representing a crack where one object rests against another one. The roof edge (integral of second derivative) occurs at boundaries between regions whose intensity profiles vary almost symmetrically across the boundary, for example, texture edges. The contrast functions for these profiles are:

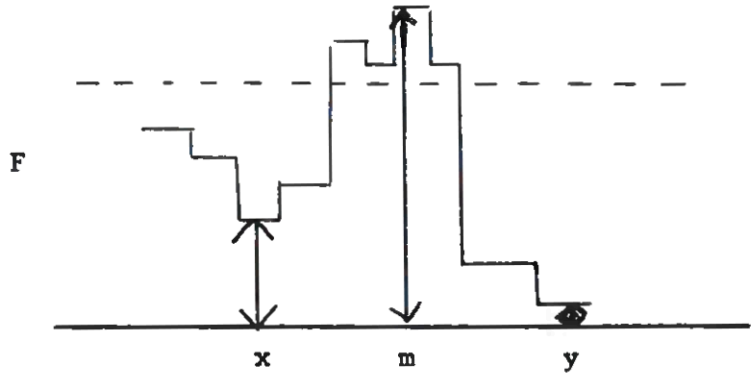


Returning now to the problem of detecting feature points, there are two questions still to be tackled:

a) Given an arbitrary contrast function, how do we decide whether or not it represents an edge?

b) Given that a contrast function does represent an edge, how do we decide what kind of edge it is?

The answer to the first question (a) is that an edge is represented in the contrast function as a good peak. A good peak is defined as a peak which is sufficiently high in an absolute sense, as well as being sufficiently high relative to nearby troughs.



F(m) > Ta where Ta is threshold on absolute height

F(m) - F(x) > Tr

F(m) - F(y) > Tr

where Tr is a threshold on relative height.

The answer to the second question is that the type of edge is determined both by the number of peaks in the contrast function and by the relationship between peaks. For example, in the case of single peak contrast functions, a positive peak represents a step edge where the intensity profile crosses from a region of relative brightness to a region of relative darkness, whereas a negative peak represents a step edge with a converse bright-dark relationship. When both positive and negative type peaks are detected in the contrast function, if the difference in the height of the two peaks is not greater than 75% of the height of the largest peak, the feature point represents a highlight in the case of a negative-positive pair, or a crack in the case of a positive-negative pair. A roof-type contrast function could be detected by examining the width of the peaks, but is usually ignored in blocks world analysis since texture information is usually less valuable than the edge, highlight and crack information.

Besides Shirai's program, this method of detecting candidate edge points has been used quite widely, to good effect. One aspect of its use which we must consider is the choice of thresholds. Up till now, we have accepted a crude approach to the problem of thresholding: the choice of a single threshold value a priori. But if boundary edges are of higher contrast than internal edges, the single threshold causes problems. If set high enough to exclude spurious boundary points, some of the internal edge data may be missed. If lowered to capture all the internal edge data, a great deal of spurious data in the neighbourhood of the boundary edges will be captured in the candidate edge point representation.

Clearly we want to use different threshold values for detecting object/background edges and surface/surface edges. The answer is to use dynamic thresholds which are automatically adjusted as processing proceeds. In other words, the threshold is set in accordance with local rather than global luminance values in the grey level representation. This practice was adopted by Shirai.

As we shall see, a difference between an edge-finding program and Shirai's edge proposing program is that the former examines each and every grey scale value in an attempt to determine if it is a candidate edge point whereas the latter examines a subset of grey scale values selected on the basis of the program's knowledge of the properties of bodies in its world: e.g. that edges are parallel. Being more specific, Shirai's program continually proposes the most plausible edges according to context, and actively searches for them by means of a set of edge seeking procedures.

Let's look at the program's main features. It analyses blocks world scenes, comprising evenly lit convex bodies with well defined edges. The most obvious intensity gradients in these scenes are the "contour" edges which separate the white body from its black background. The next obvious intensity gradients are the "boundary" edges which separate one body from another, and the least distinct are the "internal" edges which separate one face of a body from another face of the same body. The program has implicit knowledge of these differences since it is designed to detect contour edges before boundary edges, and boundary edges before internal edges. These edges are represented as lines on the program's graphical output so we will refer to them as "lines"

A typical scene, input to the computer by an image dissector device, comprises 100,000 grey scale values. As indicated above, the program's

initial task is to find the contour edges between object(s) and background. Rather than inspect every value in the grey scale representation, the program examines a sub-set of values. The grey level representation is divided into 8 x 8 subsets: one value is selected from each subset, making about 1500 in total. To find a contour edge, the program searches through this reduced data set until it finds a high contrast point. Using this point as starting value, it tries to locate the position of a contour edge segment, using a procedure called tracking. Briefly, the contour tracking routine uses the step-edge detector operator described earlier, with the threshold adjusted to the average value of the contrast function, to search for points along a hypothesised edge and to check that they are collinear, i.e. lie on a straight edge. The collinearity test checks

a) that the number of edge points exceeds a threshold number Tn

b) that the deviation E of the points in line fitting with the least square method must be less than a threshold T_E.

In this manner, a set of contour points is found. Then, the remainder of the data is scanned until a new contour point is found; tracking is repeated, and so on until the entire data subset has been examined and all sets of contour points are known.

Next, the program returns to the high-resolution grey scale representation and matches the sets of candidate contour points with particular values in the representation. Using them as starting points, it derives a refined set of contour points, by applying the contour tracking procedure once again. Next, the program forms a polygon by connecting the contour points one by one. The curvature of the polygon, i.e. the position of the vertices, is computed to yield a final contour which can be used for the next stage in the analysis, namely finding boundary lines. But a boundary line is a line on the border of an object. So contour lines are boundary lines, except where objects overlap. In that situation, there will be one or more boundary lines in the scene. If these boundary lines can be located, the program will be able to segment the scene into its constituent bodies.

Suppose the program's task is to analyse the scene showin in Figure 1 (f). As indicated above, it will begin by locating the contour lines AB, BC, CD, DE, EF, FG, GH, HI, IJ, JK and KA (Figure 1 (a)). But since the program's world excludes concave polyhedra, contour (boundary) lines which form a concave vertex can be interpreted as the boundary lines of two different bodies. The obvious strategy is to try to locate the rest of the boundary lines, using a peak edge

detector tracking operator. In fact, this is the first of a set of ten heuristic rules which embody knowledge about where lines are likely to be found. Details of these heuristics are as follows:

1. If two boundary lines make a concave point, try to find collinear extensions of them.

   * This heuristic is tried for the concave points G and J. However, the position of G is not precise enough to find the extension of FG. On the other hand, a line segment is found as an extension of the line KJ. KJ is extended by tracking, as far as L (see Figure 1b).

2. If no extensions of the two boundary lines are found, try to find another line starting from the concave point using a circular search technique. If only one is found, track along it.

   * This heuristic is invoked for point G. One line segment is found and extended until tracking terminates. Thus, line G' M' is obtained (see Figure 1b). This line is interpreted as an extension of FG. The positions of the points F,G,L are adjusted so that line G'M' becomes line F,G,L (as shown in Figure 1c).

   But notice that this means that two bodies, B1 and B2, have been identified by the creation of the boundary lines GL and JL. Consequently, the first heuristic can be applied again, at point L, provoking the extension of line FL as far as M (see Figure 1d). LM is interpreted as an extension of FL but the end point M is not connected to any other lines. Thus, the vertices F,G,L and end point M are adjusted to form the new line LM.

3. If both extensions (of the boundary lines) are found, try to find a third one and track along it.

   * This heuristics is not invoked.

4. If an end of a boundry line is left unconnected, try to find the line starting from the end point by circular search. If multiple lines are found, try to decide which line is the boundary. If a boundary line is found, track along it.

   * This yields three lines, as shown in Figure 1d. MN' is classified as a boundary line and extended by tracking. When it terminates, the line is connected to boundary line BC at N (as in Figure 1e). Now, body B1 splits into bodies, B1 and B3. At this stage, it is known that B1 is hidden by B3, and B2 is partly hidden by B3 and partly by B1.

5. If no boundary line is found by circular search, extend the unconnected boundary line by a certain length and test if it is connected to other lines. If not, apply circular search again, as in (4). If necessary, repeat the process until a solution is found.
   * Since heuristic 4 was applied successfully, the fifth one was not invoked.

   At this point in the analysis, all the boundary lines have been found. The task now is to find the internal lines.

6. Select the vertices of the boundary that might have internal lines starting from them. If a line is found, track along it using the step edge tracking procedure.
   * Notice that the selection of vertices is based on heuristics such as selecting the upper right vertex rather than the lower right vertex. Also, the system looks for internal lines that are nearly parallel to boundary lines (using its knowledge about blocks).

   This heuristic is invoked and is applied to bodies B3, B1 and B2 (starting with the most complete body since it is the easiest case to deal with). Internal lines CO and MO are found and connected at vertex O (see Figure 1e), as are AM and IP (see Figure 1f).

7. If no line is found (by 6), try to find one by circular search between adjacent boundary lines. When one is found, track along it.
   * Since the sixth heuristic was invoked, the seventh was ignored.

8. If two internal lines meet at a vertex, try to find another internal line starting at the vertex (using circular search, if necessary).
   * This is applied to vertex O and a line segment towards E is identified. This is extended by tacking as far as E' (see Figure 1f).

9. If an end of an internal line is not connected to any line, try to find lines starting from the end by circular search. If lines are found, track along them, one by one.
   * This heuristic fails.

10. If no line is found in (9), extend the line by a certain length (as in 5 above) and test if it is connected to other lines. If not, try circular search again. Repeat until successful.

* After a few trials, line OE' is extended to connect to vertex E, giving the final analysis shown in Figure 1f.

Strengths and Weaknesses

Notice that Shirai's ten heuristics are ordered with respect to their likelihood of success in finding useful cues in the scene. Also, their results are continually tested for consistency with previous results, so the program is less likely to be confused by small imperfections in the input. For example, when an unconnected line fragment is found, the program checks if it could be extended to intersect with another fragment already found, or whether it could be the continuation of an existing line fragment. In such a situation, the tracker can be made more sensitive by lowering its threshold. Similarly, circular search can be made more sensitive and more wide-ranging in the area covered. In this way, quite faint lines can be detected.

Notice,too, that Shirai's programs can ignore irrelevant lines provided they do not distort the contour because they do not make sense as representations of edges in the 3-D scene hypothesised by the interpretation as a whole.

Notice, finally, that the edge description of the scene is partitioned into groups of edges which represent bodies. Thus the program achieves a measure of segmentation. It is, however, possible for the program to miss bodies. For example, if it is presented with a tower of bricks, as shown in Figure 2a, it will not propose the cracks between them because there are not concave points to activate the boundary-line detecting heuristics. Also, when one object is supported within the face of another object, as in Figure 2b, so that no part of it touches the contour, it will not be found since the program does not include heuristics for searching for line segments inside a region.

REFERENCES

Shirai, Y. (1973) A context sensitive line finder for recognition of polyhedra. Artificial Intelligence Journal, 4 95-120.

Shirai, Y. (1975) Analyzing intensity arrays using knowledge about scenes. Psychology of Computer Vision (Ed. Winston). New York: McGraw-Hill.

Nevatia, R. (1982) Machine Perception. New Jersey: Prentice Hall. Chapter 7.

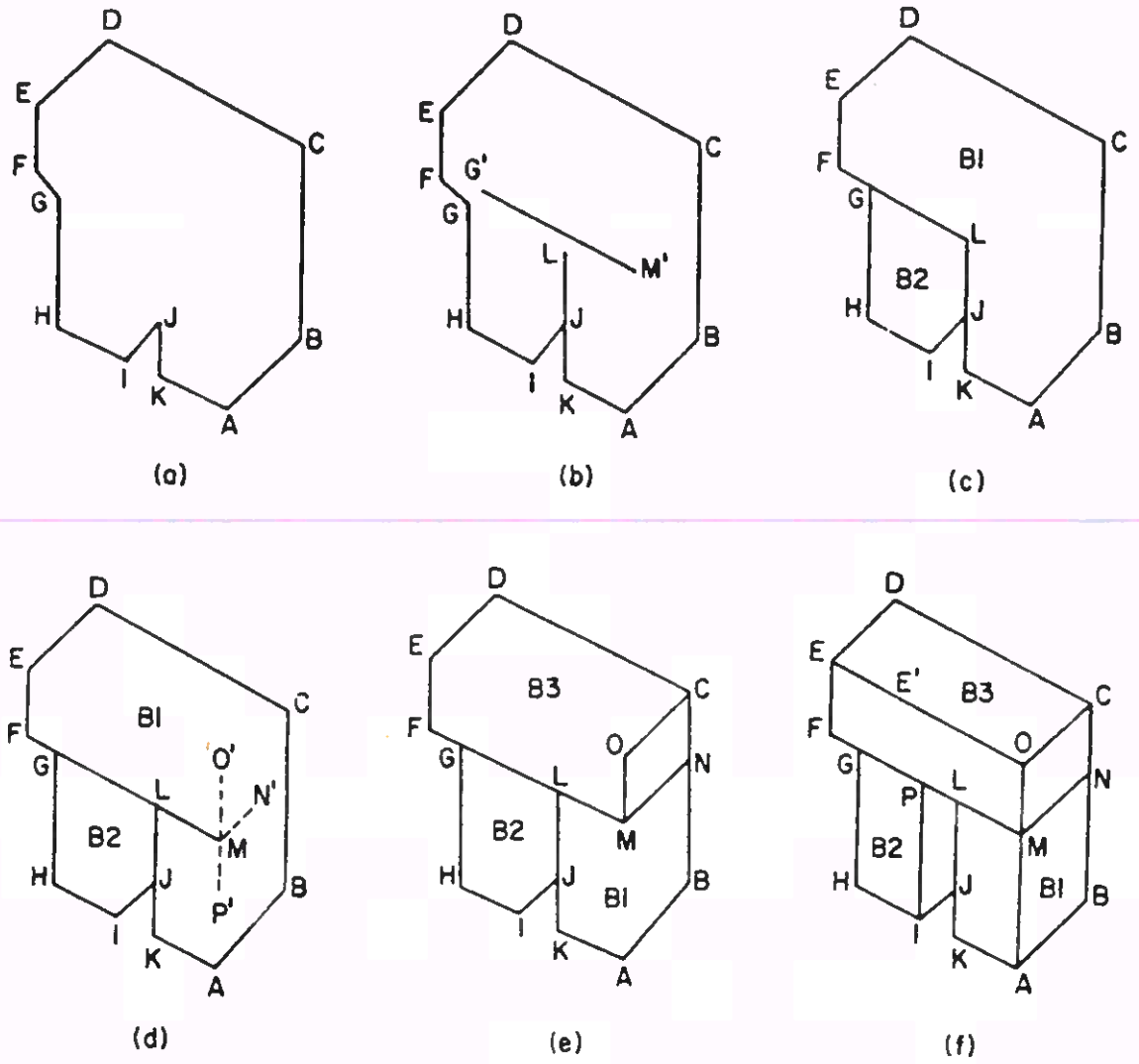


Fig. 1 Steps in analysis.

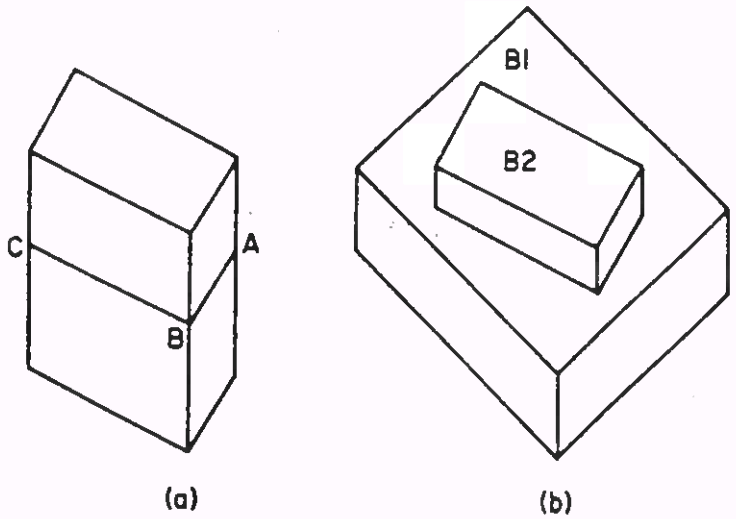


Fig. 2 Situations with a lack of cues.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

UNIVERSITY OF EDINBURGH

Knowledge guided segmentation

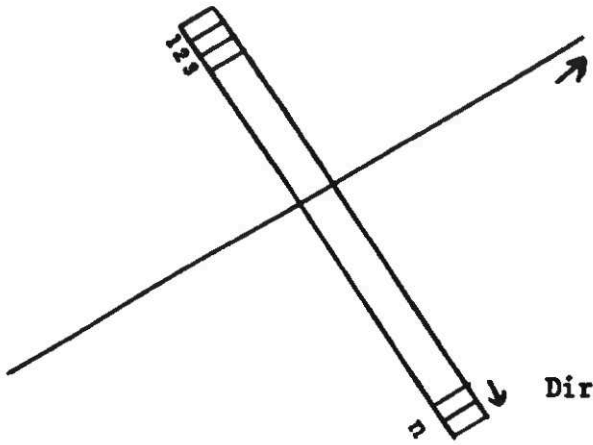
The programs which we have been reviewing so far have a hierarchical structure. First, they find feature points in an entire scene; next they make a complete edge representation using these feature points; finally, they segment the edge description to form separate bodies. This approach is susceptible to errors in the early stages (due to noise etc.); so later analysis based on the earlier results is likely to lead to serious mistakes.

What we will look at now is an attempt to overcome the limitations of the hierarchical structure, replacing it with a more flexible structure which abandons the rigid ordering in favour of a strategy of using knowledge to guide the processing. The particular research that we will be considering was carried out at M.I.T. by Shirai.

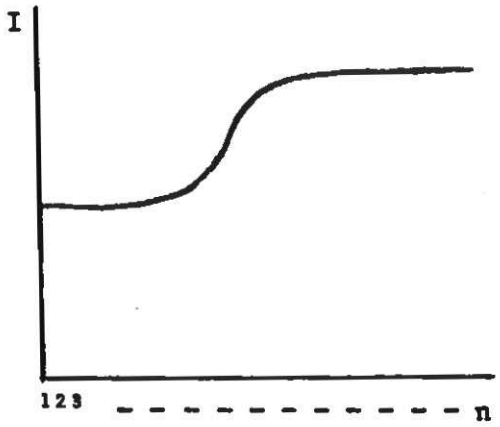
More about detecting edges

In the work considered previously, we assumed that the edges of bodies were represented as step (or near step) edges in the grey level representation. This was an over simplification. Typically a blocks world scene contains a variety of different kinds of edges. Besides the object background boundary edges, usually of high contrast (our step edge of before), there are lower contrast (more blurred) internal edges between adjacent surfaces of an object, and so on. The operators considered previously (e.g. Robert's cross, high pass filter) were designed to suit step-like edges: they perform relatively poorly on blurred edges which are characterised by luminance gradients which extend over a larger area of the grey scale representation. If we had a method that could factor out the different kinds of edges from the visual data, this information could be used by higher level processes to segment the scene into bodies, without invoking a labelling process. In fact, Shirai's program does exactly this. Let's examine how it does it.

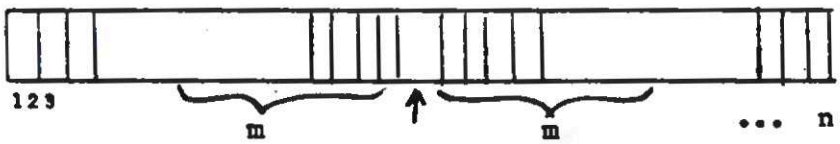
Consider an intensity profile of n points taken along a band perpendicular to the direction of a step edge:



The intensity profile is represented as follows



The contrast function is calculated as follows. We define the contrast function of the i^{th} profile point, to be the difference between the sum of the m subsequent points and the sum of the m preceding points, where m is a parameter. Graphically,

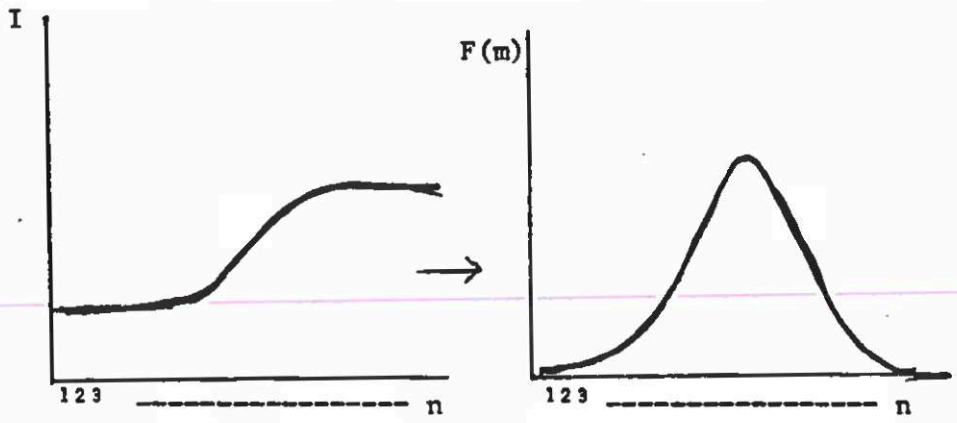


i^{th} profile point

$$S_1 = \sum_{r=i-m}^{i-1} \qquad S_2 = \sum_{r=i+1}^{i+m}$$

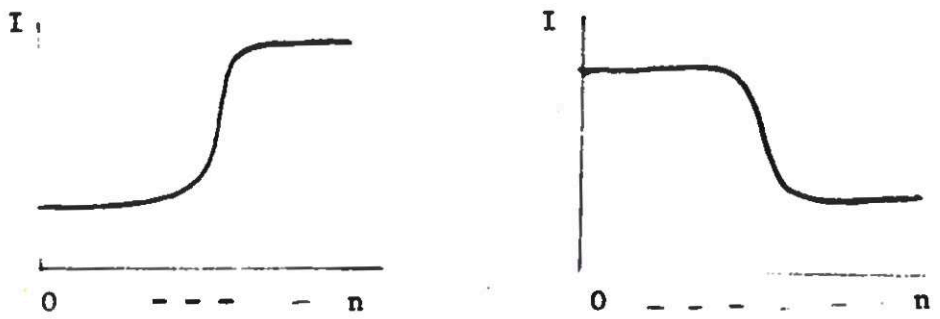
The contrast function at the i^{th} profile point is F(m) where $Fm(i) = S_1 - S_2$

But there are (n-2m) points for which values will be obtained, yeilding a contrast function of the edge.

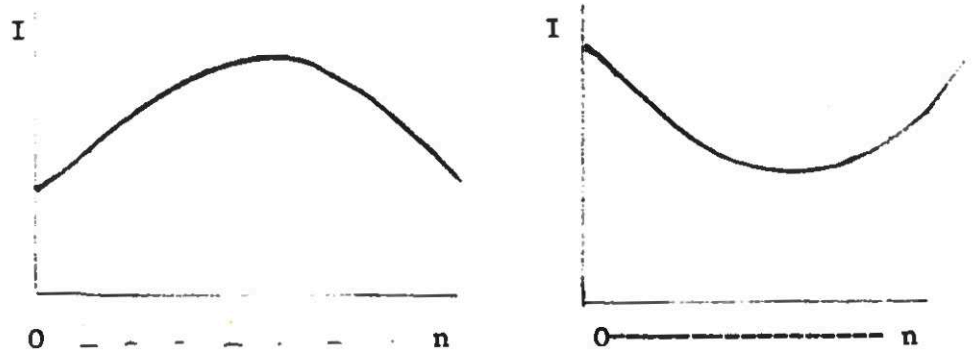


Intensity function Contrast function

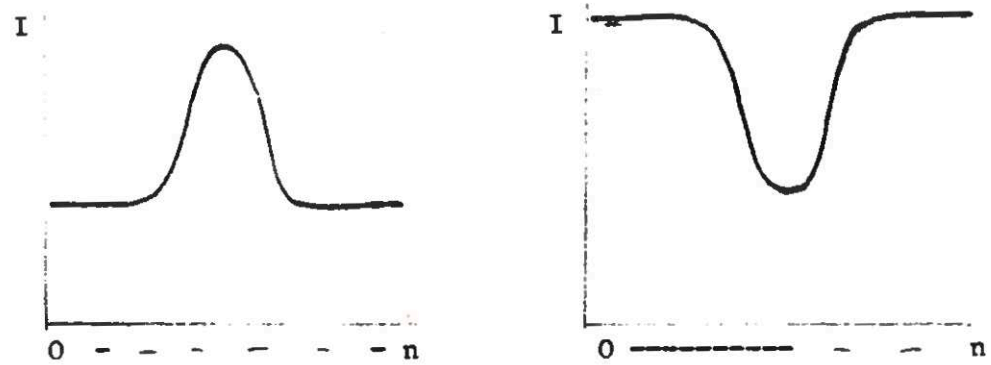
So much for the step edge, but as pointed out above not all real edges have similar cross-sectional intensity (luminance) profiles. Herskovits and Binford classified edges into three types, namely, step, roof, peak (or spike), according to the shape of their intensity profiles.



step



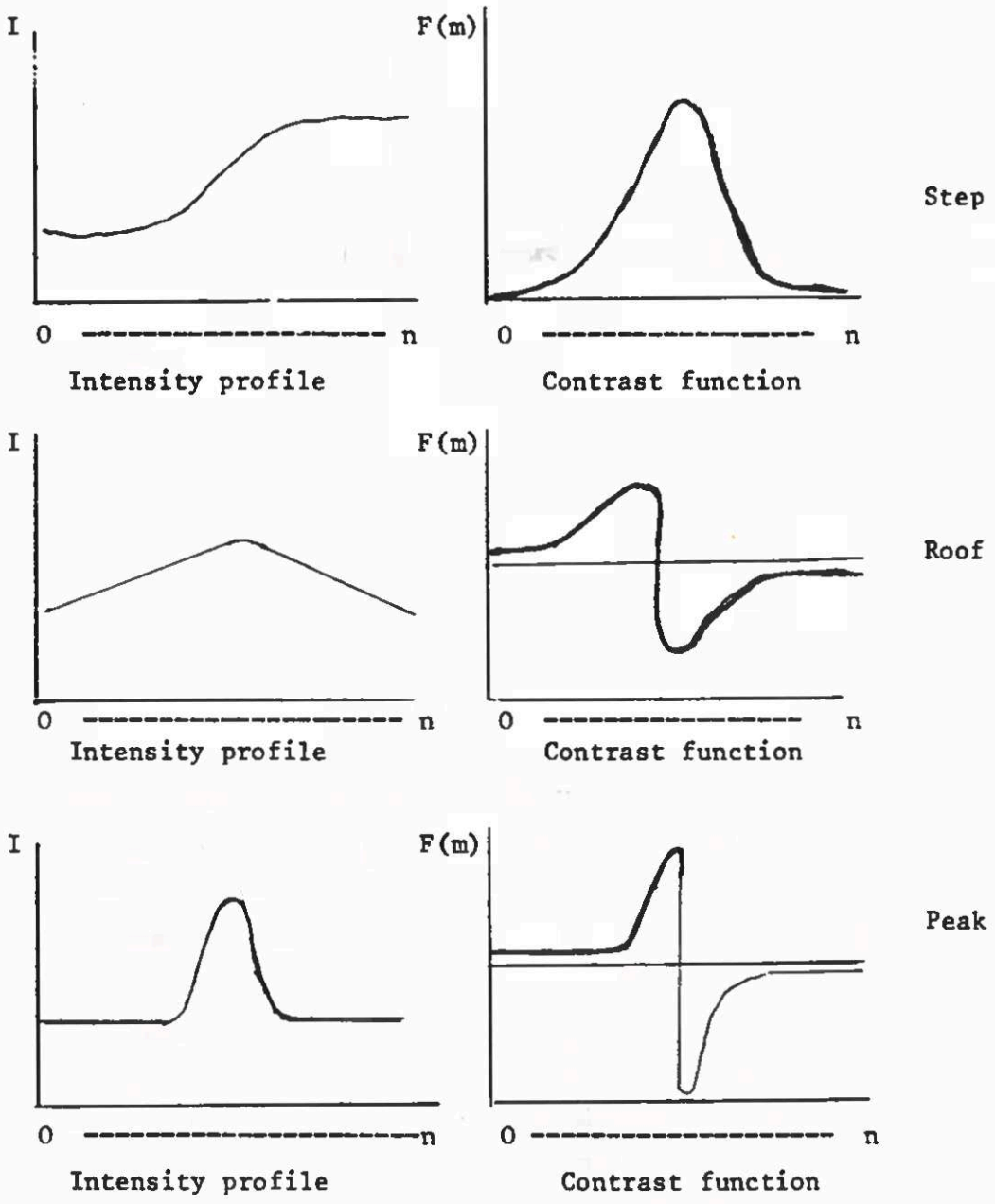
roof



peak (spike)

Intensity profiles

The step edge (first derivative), encountered previously, occurs at a high contrast boundary between regions of relatively homogeneous intensity. The peak (or spike) edge (second derivative) occurs at boundaries representing a sharp highlight, or representing a crack where one object rests against another one. The roof edge (integral of second derivative) occurs at boundaries between regions whose intensity profiles vary almost symmetrically across the boundary, for example, texture edges. The contrast functions for these profiles are:

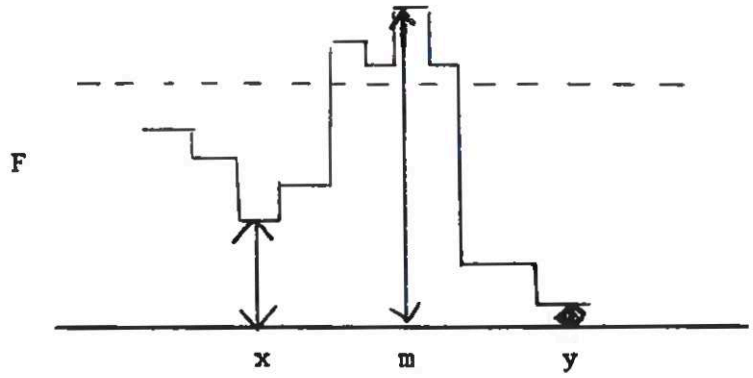


Returning now to the problem of detecting feature points, there are two questions still to be tackled:

a) Given an arbitrary contrast function, how do we decide whether or not it represents an edge?

b) Given that a contrast function does represent an edge, how do we decide what kind of edge it is?

The answer to the first question (a) is that an edge is represented in the contrast function as a good peak. A good peak is defined as a peak which is sufficiently high in an absolute sense, as well as being sufficiently high relative to nearby troughs.



$F(m) > Ta$ where Ta is threshold on absolute height

$F(m) - F(x) > Tr$

$F(m) - F(y) > Tr$

where Tr is a threshold on relative height.

The answer to the second question is that the type of edge is determined both by the number of peaks in the contrast function and by the relationship between peaks. For example, in the case of single peak contrast functions, a positive peak represents a step edge where the intensity profile crosses from a region of relative brightness to a region of relative darkness, whereas a negative peak represents a step edge with a converse bright-dark relationship. When both positive and negative type peaks are detected in the contrast function, if the difference in the height of the two peaks is not greater than 75% of the height of the largest peak, the feature point represents a highlight in the case of a negative-positive pair, or a crack in the case of a positive-negative pair. A roof-type contrast function could be detected by examining the width of the peaks, but is usually ignored in blocks world analysis since texture information is usually less valuable than the edge, highlight and crack information.

Besides Shirai's program, this method of detecting candidate edge points has been used quite widely, to good effect. One aspect of its use which we must consider is the choice of thresholds. Up till now, we have accepted a crude approach to the problem of thresholding: the choice of a single threshold value a priori. But if boundary edges are of higher contrast than internal edges, the single threshold causes problems. If set high enough to exclude spurious boundary points, some of the internal edge data may be missed. If lowered to capture all the internal edge data, a great deal of spurious data in the neighbourhood of the boundary edges will be captured in the candidate edge point representation.

Clearly we want to use different threshold values for detecting object/background edges and surface/surface edges. The answer is to use dynamic thresholds which are automatically adjusted as processing proceeds. In other words, the threshold is set in accordance with local rather than global luminance values in the grey level representation. This practice was adopted by Shirai.

As we shall see, a difference between an edge-finding program and Shirai's edge proposing program is that the former examines each and every grey scale value in an attempt to determine if it is a candidate edge point whereas the latter examines a subset of grey scale values selected on the basis of the program's knowledge of the properties of bodies in its world: e.g. that edges are parallel. Being more specific, Shirai's program continually proposes the most plausible edges according to context, and actively searches for them by means of a set of edge seeking procedures.

Let's look at the program's main features. It analyses blocks world scenes, comprising evenly lit convex bodies with well defined edges. The most obvious intensity gradients in these scenes are the "contour" edges which separate the white body from its black background. The next obvious intensity gradients are the "boundary" edges which separate one body from another, and the least distinct are the "internal" edges which separate one face of a body from another face of the same body. The program has implicit knowledge of these differences since it is designed to detect contour edges before boundary edges, and boundary edges before internal edges. These edges are represented as lines on the program's graphical output so we will refer to them as "lines"

A typical scene, input to the computer by an image dissector device, comprises 100,000 grey scale values. As indicated above, the program's

initial task is to find the contour edges between object(s) and background. Rather than inspect every value in the grey scale representation, the program examines a sub-set of values. The grey level representation is divided into 8 x 8 subsets: one value is selected from each subset, making about 1500 in total. To find a contour edge, the program searches through this reduced data set until it finds a high contrast point. Using this point as starting value, it tries to locate the position of a contour edge segment, using a procedure called tracking. Briefly, the contour tracking routine uses the step-edge detector operator described earlier, with the threshold adjusted to the average value of the contrast function, to search for points along a hypothesised edge and to check that they are collinear, i.e. lie on a straight edge. The collinearity test checks

a) that the number of edge points exceeds a threshold number Tn

b) that the deviation E of the points in line fitting with the least square method must be less than a threshold T_E.

In this manner, a set of contour points is found. Then, the remainder of the data is scanned until a new contour point is found; tracking is repeated, and so on until the entire data subset has been examined and all sets of contour points are known.

Next, the program returns to the high-resolution grey scale representation and matches the sets of candidate contour points with particular values in the representation. Using them as starting points, it derives a refined set of contour points, by applying the contour tracking procedure once again. Next, the program forms a polygon by connecting the contour points one by one. The curvature of the polygon, i.e. the position of the vertices, is computed to yield a final contour which can be used for the next stage in the analysis, namely finding boundary lines. But a boundary line is a line on the border of an object. So contour lines are boundary lines, except where objects overlap. In that situation, there will be one or more boundary lines in the scene. If these boundary lines can be located, the program will be able to segment the scene into its constituent bodies.

Suppose the program's task is to analyse the scene showin in Figure 1 (f). As indicated above, it will begin by locating the contour lines AB, BC, CD, DE, EF, FG, GH, HI, IJ, JK and KA (Figure 1 (a)). But since the program's world excludes concave polyhedra, contour (boundary) lines which form a concave vertex can be interpreted as the boundary lines of two different bodies. The obvious strategy is to try to locate the rest of the boundary lines, using a peak edge

detector tracking operator. In fact, this is the first of a set of ten heuristic rules which embody knowledge about where lines are likely to be found. Details of these heuristics are as follows:

1. If two boundary lines make a concave point, try to find collinear extensions of them.

   * This heuristic is tried for the concave points G and J. However, the position of G is not precise enough to find the extension of FG. On the other hand, a line segment is found as an extension of the line KJ. KJ is extended by tracking, as far as L (see Figure 1b).

2. If no extensions of the two boundary lines are found, try to find another line starting from the concave point using a circular search technique. If only one is found, track along it.

   * This heuristic is invoked for point G. One line segment is found and extended until tracking terminates. Thus, line G' M' is obtained (see Figure 1b). This line is interpreted as an extension of FG. The positions of the points F,G,L are adjusted so that line G'M' becomes line F,G,L (as shown in Figure 1c).

   But notice that this means that two bodies, B1 and B2, have been identified by the creation of the boundary lines GL and JL. Consequently, the first heuristic can be applied again, at point L, provoking the extension of line FL as far as M (see Figure 1d). LM is interpreted as an extension of FL but the end point M is not connected to any other lines. Thus, the vertices F,G,L and end point M are adjusted to form the new line LM.

3. If both extensions (of the boundary lines) are found, try to find a third one and track along it.

   * This heuristics is not invoked.

4. If an end of a boundry line is left unconnected, try to find the line starting from the end point by circular search. If multiple lines are found, try to decide which line is the boundary. If a boundary line is found, track along it.

   * This yields three lines, as shown in Figure 1d. MN' is classified as a boundary line and extended by tracking. When it terminates, the line is connected to boundary line BC at N (as in Figure 1e). Now, body B1 splits into bodies, B1 and B3. At this stage, it is known that B1 is hidden by B3, and B2 is partly hidden by B3 and partly by B1.

5. If no boundary line is found by circular search, extend the unconnected boundary line by a certain length and test if it is connected to other lines. If not, apply circular search again, as in (4). If necessary, repeat the process until a solution is found.
   * Since heuristic 4 was applied successfully, the fifth one was not invoked.

   At this point in the analysis, all the boundary lines have been found. The task now is to find the internal lines.

6. Select the vertices of the boundary that might have internal lines starting from them. If a line is found, track along it using the step edge tracking procedure.
   * Notice that the selection of vertices is based on heuristics such as selecting the upper right vertex rather than the lower right vertex. Also, the system looks for internal lines that are nearly parallel to boundary lines (using its knowledge about blocks).

   This heuristic is invoked and is applied to bodies B3, B1 and B2 (starting with the most complete body since it is the easiest case to deal with). Internal lines CO and MO are found and connected at vertex O (see Figure 1e), as are AM and IP (see Figure 1f).

7. If no line is found (by 6), try to find one by circular search between adjacent boundary lines. When one is found, track along it.
   * Since the sixth heuristic was invoked, the seventh was ignored.

8. If two internal lines meet at a vertex, try to find another internal line starting at the vertex (using circular search, if necessary).
   * This is applied to vertex O and a line segment towards E is identified. This is extended by tacking as far as E' (see Figure 1f).

9. If an end of an internal line is not connected to any line, try to find lines starting from the end by circular search. If lines are found, track along them, one by one.
   * This heuristic fails.

10. If no line is found in (9), extend the line by a certain length (as in 5 above) and test if it is connected to other lines. If not, try circular search again. Repeat until successful.

* After a few trials, line OE' is extended to connect to vertex E, giving the final analysis shown in Figure 1f.

Strengths and Weaknesses

Notice that Shirai's ten heuristics are ordered with respect to their likelihood of success in finding useful cues in the scene. Also, their results are continually tested for consistency with previous results, so the program is less likely to be confused by small imperfections in the input. For example, when an unconnected line fragment is found, the program checks if it could be extended to intersect with another fragment already found, or whether it could be the continuation of an existing line fragment. In such a situation, the tracker can be made more sensitive by lowering its threshold. Similarly, circular search can be made more sensitive and more wide-ranging in the area covered. In this way, quite faint lines can be detected.

Notice,too, that Shirai's programs can ignore irrelevant lines provided they do not distort the contour because they do not make sense as representations of edges in the 3-D scene hypothesised by the interpretation as a whole.

Notice, finally, that the edge description of the scene is partitioned into groups of edges which represent bodies. Thus the program achieves a measure of segmentation. It is, however, possible for the program to miss bodies. For example, if it is presented with a tower of bricks, as shown in Figure 2a, it will not propose the cracks between them because there are not concave points to activate the boundary-line detecting heuristics. Also, when one object is supported within the face of another object, as in Figure 2b, so that no part of it touches the contour, it will not be found since the program does not include heuristics for searching for line segments inside a region.

REFERENCES

Shirai, Y. (1973) A context sensitive line finder for recognition of polyhedra. Artificial Intelligence Journal, 4 95-120.

Shirai, Y. (1975) Analyzing intensity arrays using knowledge about scenes. Psychology of Computer Vision (Ed. Winston). New York: McGraw-Hill.

Nevatia, R. (1982) Machine Perception. New Jersey: Prentice Hall. Chapter 7.

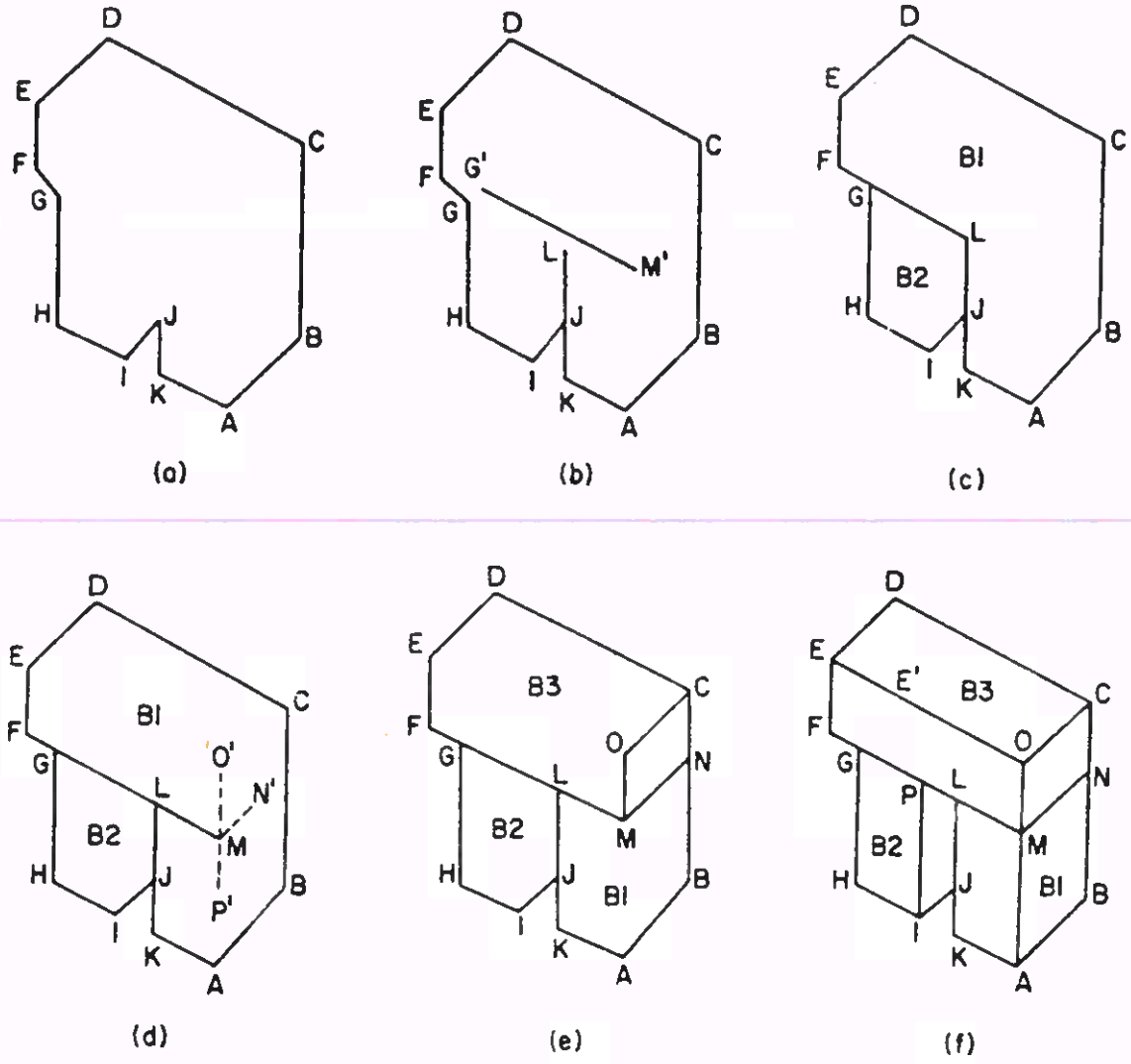


Fig. 1 Steps in analysis.

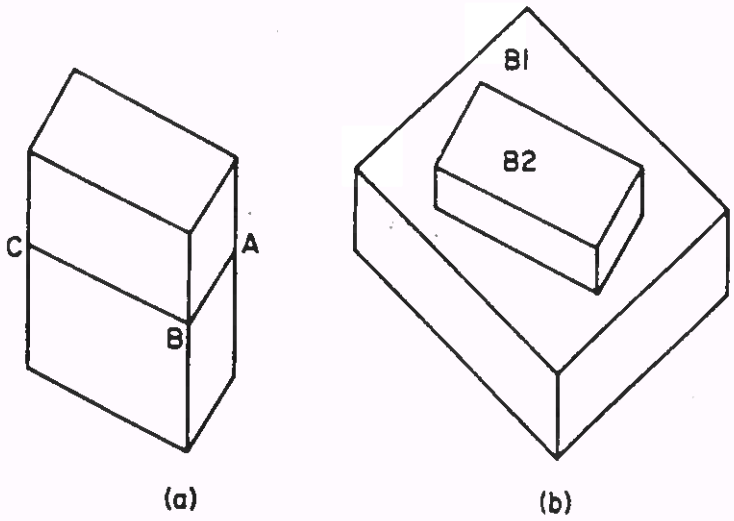


Fig. 2 Situations with a lack of cues.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

AI-2

JIM HOWE

Model-based segmentation

Today, we are going to follow a different route to segmentation, namely segmentation through recognition of the primitive bodies in a scene. This is the approach taken by Roberts.

Roberts' thinking was strongly influenced by the work of a psychologist called J J Gibson (1966). Previously, most psychologists considered vision as a problem of extracting invariant information about the physical world from a flat, static, ambiguous 2-D image. We have already seen that an 'edge' is not invariant information because it does not uniquely characterise its source in the environment. Instead, it might be caused by a change in illumination or colour, or perhaps by the edge of a shadow. So an 'edge' does not unequivocally specify any particular thing. Gibson suggested that the visual system extracts invariant information about the physical world from transformations of images over time. Typically, these transformations occur through movement of the observer, Information about surfaces such as their orientation, their extent and their inter-relationships is specific to the environment, and can be obtained by interpreting velocity signals generated by retinal transformations. This is still a research problem that we will not deal with here. Instead, Robert's captured invariance through the use of 3-D models stored in the machine. Each model represents an invariant percept, and each can be identified with any projection of itself (in 3 or 2-D). Such a system would therefore recognise what the objects in a scene are without the intervening activity of segmentation.

Like most other programs considered so far, Roberts' system is limited to handling three types of 'primitive'objects, namely cube, rectangular wedge and hexagonal prism (see Figure 1). Typical scenes analysed by the system include simple and/or compound objects. Simple objects are created by transformations which expand a 'primitive' object along each of its co-ordinate axes, rotate it and translate it. Compound (composite) objects are constructed by abutting two or more simple objects so that each pair share a common surface. Examples are shown in Figure 2.

Roberts' program is really a conglomerate of three separate programs. The first reduces the photograph of a scene (composed of simple/compound objects) to a line drawing using similar methods to those studied earlier. This program yields a set of lines, represented by their end point co-ordinates, and a set of regions bounded by these lines, where the regions are all polygons since the objects are all planar.

The second program, the interpretation program, handles the task of finding the models which best describe a scene. It is this process which interests us. The third program, which constructs a two-dimensional projection of the objects in a scene, will not be dealt with.

To understand the interpretation program, we have to understand the picture taking process. Details are as follows.

An image is a direct perspective transformation of a 3-D field of object points. This transformation is a projection of each object point on to a plane surface, through a lens. The basic model of the process is shown in Figure 3 where the camera is represented by a pinhole lens with an image plane lying a distance f behind the lens. The image of a point V in 3-space is determined by the intersection of the image plane with the projecting ray defined by V and the lens centre. V_p is the image point corresponding to the object point V.

A difficulty with this basic model of the picture-taking process is that the image is a reflection of the object, i.e. the image is inverted from top to bottom and left to right. To avoid reflections, the preferred focal plane is placed in front of the camera and intercepts the projecting ray as shown in Figure 4. Here the focal plane is really the plane of the print, not of the negative.

This process is called central projection. The z-axis is aligned with the optical axis, or principal ray, of the camera, the principal ray being the ray from the lens perpendicular to the image plane. The centre of co-ordinates (0,0,0) is the intersection of the principal ray with the image plane. Notice that the process of central projection is many-to-one: for each image point there is a line in space, defined by the image point and the lens centre, along which the corresponding object point must lie.

We are now in a position to understand the two questions associated with the image making process. First, given an arbitrary object point, the answer to the question, "What is the location of its image point?" is provided by a direct perspective transformation from object point to image point. Second, given an arbitrary image point, the answer to the question, "What is the straight line along which the corresponding object must lie?" is provided by the inverse perspective transformation from image point to object point.

Unfortunately, the equations from the direct perspective transformation of the object point V to the image point V_p are non-linear because they involve divisions. This difficulty is overcome by converting the non-linear transformations into linear transformations in a different co-ordinate system in which an object point (x,y,z) in three dimensions is represented by a vector of four numbers $[x_1, y_1, z_1, h_1]$. The four components of this vector are interpreted as co-ordinates

in four dimensional space. In order to transform a point in 3-D co-ordinates (x,y,z) into this 4-D representation, known as homogeneous co-ordinates, we merely choose some non-zero number w and form the vector [wx wy wz w]. This number w is the scale factor or the homogeneous co-ordinate. Since the choice of w is arbitrary, the homogeneous co-ordinates of a point are not unique. Of course, a homogeneous point [wx wy wz w] can be converted back into 3-D co-ordinates by dividing by the scale factor: the point is $[wx/_w \quad wy/_w \quad wz/_w]$.

With homogeneous co-ordinates, we can now express the perspective transformations as linear matrix operations. Let's consider a simple example, a unit cube centred at the origin (0,0,0). In 3-space, this can be described as a set of eight points, $V1(x_1, y_1, z_1,)$ to $V8(x_8, y_8, z_8)$, representing the eight vertices, as shown in Figure 5.

This set of eight points in 3-D space can be described by an 8 x 4 matrix of homogeneous co-ordinates. Co-ordinate transformations and projections, either with or without perspective, can be done, either individually or in combination, by multiplying this 8 x 4 matrix by an appropriate 4 x 4 transformation matrix to give a product matrix of the transformed homogeneous co-ordinates, as shown in Figure 6.

Returning to the central projection problem, the image co-ordinates are obtained by multiplying the object co-ordinates by a transformation matrix P. In similar fashion, the object co-ordinates can be obtained from the image co-ordinates by the inverse transformation, P^{-1}.

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-1}{f} & 1 \end{bmatrix} \qquad P^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{f} & 1 \end{bmatrix}$$

Note that the first column of the matrix affects only the x co-ordinate and so contains all the numbers that define the updated x co-ordinate. The same holds for y and z. A vertex is transformed as follows: To get a new x co-ordinate, the old x co-ordinate is multiplied by the top number in the first column; its result is added to the product of the old y co-ordinate and the second number in the first column. The sum is then added to the product of the old z co-ordinate and the third number in the first column. Finally, the total is added to the bottom number in the first column. The new y, z and h co-ordinates are obtained in the same way, using the second,third and fourth columns.

The perspective transformation applies when the object and image points are specified in the co-ordinate system aligned with the camera. It may be necessary to express objects in an independent co-ordinate system, the world co-ordinate system i.e. object co-ordinates have to be transformed from world co-ordinates to a system aligned with the camera before a perspective transformation can be applied.

The models in Roberts' system (cube, rectangular wedge and hexagonal prism) are also represented by homogeneous co-ordinates. This means that a model can be transformed to match an object in the scene by applying translation, scaling and rotation transformations. Indeed the advantage of matrix multiplication becomes evident when several consecutive transformations are required because a series of transformations can be done by using only a single 4 x 4 matrix which is the product of individual 4 x 4 transformation matrices. Note, however, that the individual matrices must be multiplied in the order in which the transformations are required. For example, a rotation followed by a translation does not give the same result as a translation followed by a rotation i.e. matrix multiplication is not commutative.

The transformation which translates a point (x y z) to the new point (x', y', z') is

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

where T_x, T_y and T_z are the components of the translation in the x, y and z directions respectively.

The transformation which scales dimensions in each co-ordinate direction separately is

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Different scalings along the three axes may be represented by non-unit terms in the diagonal of matrix S

Rotational transformations are more complex: we must determine an axis of rotation. The simplest form of transformation occurs when the axis passes through the origin and is aligned with a co-ordinate axis. To rotate about an arbitrary point, we must concatenate three transformations: translate the point to the origin, perform the rotation and translate back. To complicate matters, we must cope with axes of rotation that are not aligned with the co-ordinate axes: in these cases we have to concatenate several primitive rotation transformations to form a matrix that rotates about the desired axis.

The transformations for rotation about each of the co-ordinate axes, shown in Figure 7, are given below. Note that rotation is measured clockwise about the named axis of rotation, looking along the axis towards the origin.

Rotation by θ about the x axis, Rx:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation by θ about the y axis, Ry:

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation by θ about the z axis, Rz:

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now we are in a position to understand Roberts' program. Recognition is achieved by selecting a model, and by applying a transformation matrix R which will scale, rotate and translate it so that it matches a body in the scene.

The question to be solved is 'How is R calculated?'

Recollect that we encountered the matrix P which transforms object to image co-ordinates:

If we can find a transformation H which transforms model points into image points, we can calculate R as a concatenation of H and the inverse of P i.e. $R = HP^{-1}$. Having identified the object, its location can be specified completely, except for its actual distance from the camera. This distance is computed by assuming that the most downward facing surface of an object must lie on the ground plane.

We turn now to consider how H is found. The question now is how does the program select a model to account for the image data? Obviously, the space of the three models, juxtaposed and transformed in all possible ways and viewed from every direction is too large for a blind search. So instead of generating all possible images of all possible objects until one matches the input data, the search space must be intelligently structured. Roberts uses the facts that despite transformation the topology of an object remains invariant, and within a wide range of viewpoints the topology of the visible aspect of an object does not change, to reduce the search space to models viewed from a small number of typical viewpoints.

Let us consider the topology search in detail. It is based on the notion of an "approved polygon" which is simply one of the shapes of the model surfaces. For the three models used in the program, the set of possible polygons is restricted to convex polygons of 3, 4 or 6 sides. Notice that eachvertex on a cube has 3 quadrilaterals around it; each vertex on a wedge model has 2 quadrilaterals and 1 triangle around it, and each vertex on a prism model has 2 quadrilaterals and 1 hexagon around it. We can now identify the task of the interpretation program as that of matching regions (polygons) in the picture description with regions in the model description.

The first step is to select the appropriate regions from the scene description since not all regions will be convex polygons (1) because of the effects of occlusion of one object on another, and (2) the presence of compound objects. To find the largest picture fragment corresponding to a model, the program applies an ordered sequence of tests, each one involving a smaller fragment than the previous one until one succeeds.

Test 1. Find a picture point surrounded by three approved polygons. See Figure 8a. (7 picture points required). Notice that this test corresponds to Guzman's FORK rule.

Test 2. Find a line with an approved polygon on each side. See Figure 8b. (6 picture points required). Notice that this test corresponds to Guzman's ARROW rule.

Test 3. Find an approved polygon with an extra line coming from one vertex. See Figure 8c. (5 picture points required).

Test 4. As a last resort, find a point with 3 lines coming from it. See Figure 8d. (4 picture points required).

When a fragment has been identified the program searches the models in sequence, cube followed by wedge followed by prism, to find a model point surrounded by the same polygon structure as the fragment's image point. Next, it constructs a list of topologically equivalent image-model point pairs; it transforms the model fragment and calculates the mean square error of the fit between model fragment and image fragment, using a threshold to eliminate any model which matches the image topologically but would require deformation to fit it. A model with a small error can now be completely transformed to produce lines and points not part of the fitted area. These points are checked against the image to make sure they do not fall outside the object's external boundary. A fit confirms the selection of the correct model and the correctness of the transformation, H.

If some of the model-generated points fall outside the external boundary of the image, the wrong model has been selected and another must be tried. If, however, they fall within the boundary, but do not account for all the image lines, the image is probably of a compound object. The points corresponding to the model are then stripped from the image, and the remaining points are examined as hitherto.

In Figure 9, we see a typical compound object considered by Roberts. The topology search finds no fragment from applying Step 1, but two from applying Step 2, namely lines 2 and 3 have approved polygons on each side of them. The cube has approved polygons on both sides of an edge so the geometry matcher tries A and B as surfaces of a transformed cube as shown in Figure 10, but discovers that the residual error of the least squares fit of the corresponding object-model point pairs is too large and rejects it. Similarly for line 3. The topology search then finds a fragment using Step 3, namely polygon A with line 9 attached. The five points defined by that fragment match a transformed cube exactly,as in (b). This is removed from the original image and the process continues by finding the parts shown in (c) and (d) with the final compound object shown in (e).

This example also emphasises the importance of perspective. Without it, lines 1, 2 and 3 would be parallel, as would lines 5 and 6, and 7 and 8. Under these circumstances, the transformed cube would fit exactly as shown in Figure 10 with disastrous effects on the subsequent analysis. This imposes severe demands on the accuracy of Roberts' line-finder system whose adequacy is

questionable. Just as with other systems we have looked at, it is relatively easy to find counter-examples. Mackworth provides one, shown in Figure 11. This object is simply a wedge on top of a cuboid. But it seems that whenever topology tests succeed, the model suggested will not pass the geometric transformation test, and so the program fails completely. Details are as follows:

The topology test finds the two quadrilaterals flanking line 4 but if one face of the cube is fitted to region A the rest of the cube will fall outside the complete figure as shown in Figure 12(a). Next, it finds a polygon with a line from one corner, and attempts to fit cubes or wedges. Again it fails. In particular, Figure 12(b) shows a wedge that might be thought to fit but it is incorrect as only rectangular wedges are allowed. Finally, even withdrawing to the weakest test, a point with three lines coming from it, will not succeed. Looking at lines 1, 2 and 3 of figure 12(c) we can see that they are the three significant edges of a cube model that could be made to fit but the program cannot find that context as it only looks for contexts concentrated at vertices.

Before moving on, a few words of comparison between Guzman's and Roberts' program might be useful.

1. Guzman's program would segment the scene called 'BRIDGE', shown in Figure 13, into 8 separate bodies, namely:

```
Body 1 : (R24 R9  R21  R27 R12 R25)
Body 2 : (R22 R26 R23)
Body 3 : (R17 R3  R20)
Body 4 : (R1  R2)
Body 5 : (R13 R14 R15)
Body 6 : (R19 R18 R16)
Body 7 : (R29 R28)
Body 8 : (R8  R11 R5   R6  R4  R10  R7)
```

The question is how would Roberts' program cope with this scene? We might expect it to arrive at the following conclusions:

(R24 R9 R21 R27 R12 R25) is instance of *cube* cf. Body 1 above
(R22 R23 R26) is instance of *cube* cf. Body 2 above
(R17 R3 R20) is instance of *cube* cf. Body 3 above
(R1 R2) is instance of *wedge* cf. Body 4 above
(R3 R14 R15) is instance of *cube* cf. Body 5 above
(R16 R18 R19) is instance of *cube* (or *wedge*) cf. Body 6 above
(R28 R29) is instance of *cube* (or *wedge*) cf. Body 7 above

So far, Roberts' program has made the same segmentation of the scene. However, at this point its analysis differs. Guzman's "Body 8" is not an instance of one of Roberts' prototypes. Instead Roberts' program would decompose it into its primitive parts, as shown in Figure 14 yielding:

(R10 R32) is instance of *cube* cf. Body 8 above
(R33 R34) is instance of *cube* cf. Body 8 above
(R4 R11) is instance of *cube* cf. Body 8 above
(R6 R5 R31) is instance of *cube* cf. Body 8 above

So Roberts' program finds three more *components* than Guzman's program, i.e. it doesn't matter whether or not all the edge information is present in the picture description, *provided that the outer boundary is intact.* In contrast, Guzman's program is highly susceptible to missing internal edge information. The reason for this difference is that Roberts' prototypes carry with them information about 3-D structure whereas Guzman's corner models are derived from the 2-D appearance of a 3-D scene, and do not carry information about 3-D structure.

2. Notice that Roberts' first test, namely finding a point surrounded by three approved polygons, corresponds to Guzman's FORK heuristic. Notice also that his second test, namely find a line flanked by two approved polygons, is Guzman's ARROW rule. Finally, notice also Roberts' use of T-joints to provide evidence of interposition of one body in front of another one.

3. Although we discussed Guzman's program before dealing with Roberts' program, in fact Roberts' progrzm was written about 4 years *before* Guzman's program. Although it doesn't identify objects, it does identify all the primitive component bodies, e.g. cubes, wedges and hexagonal prisms, and can name them if required. Because of this, it is referred to as a recognition program, and is cited by many as an important example of the

seeing paradigm which is based on the notion of a stimulus fragment invoking a prototype model. But in reality, Roberts' program is another *special case* segmentation program because it decomposes a scene into its constituent bodies, i.e. blocks, wedges and prisms. It does not recognise objects made from these components, such as arches, bridges, tables and so on.

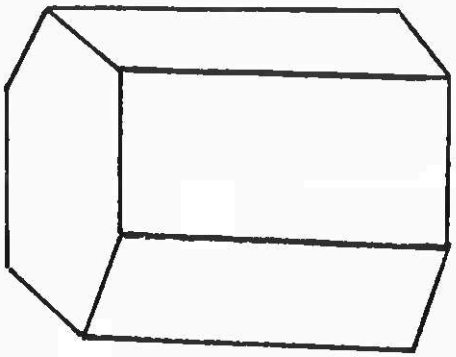
References

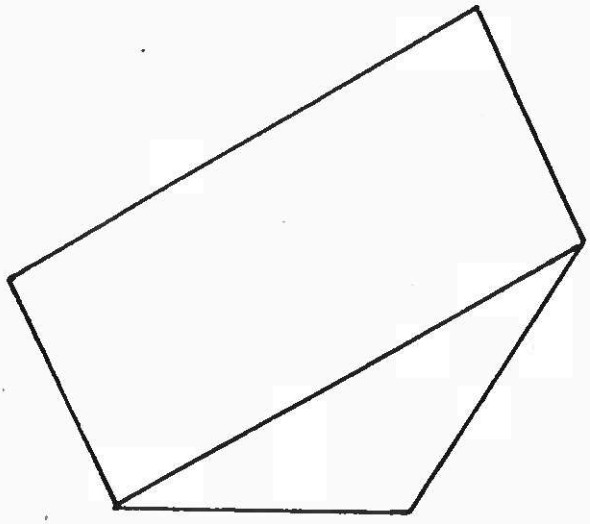
Gibson, J.J. (1966) *The Senses Considered on Perceptual Systems.* Cornel Press.

Newman, W.M. & Sproull, R.F. (1979) *Principles of Interactive Computer Graphics.* McGraw Hill.

Roberts, L.G. (1965) Machine Perception of Three Dimensional Solids. In *Optical and Electro-optical Information Processing, Chapter 9,* M.I.T.

Nevatia, R. (1982) Machine Perception. New Jersey: Prentice Hall. Chapt. 3.





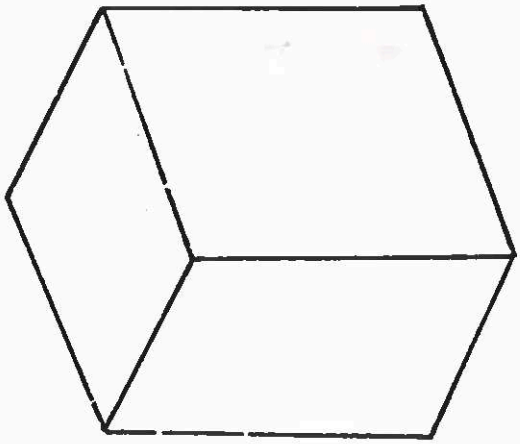
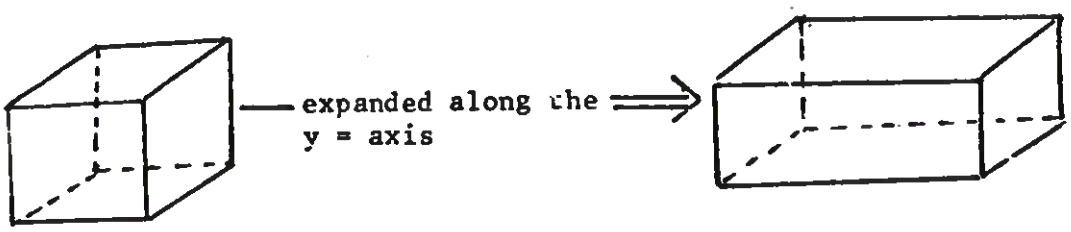


Figure 1.

MODEL TRANSFORMATION OBJECT

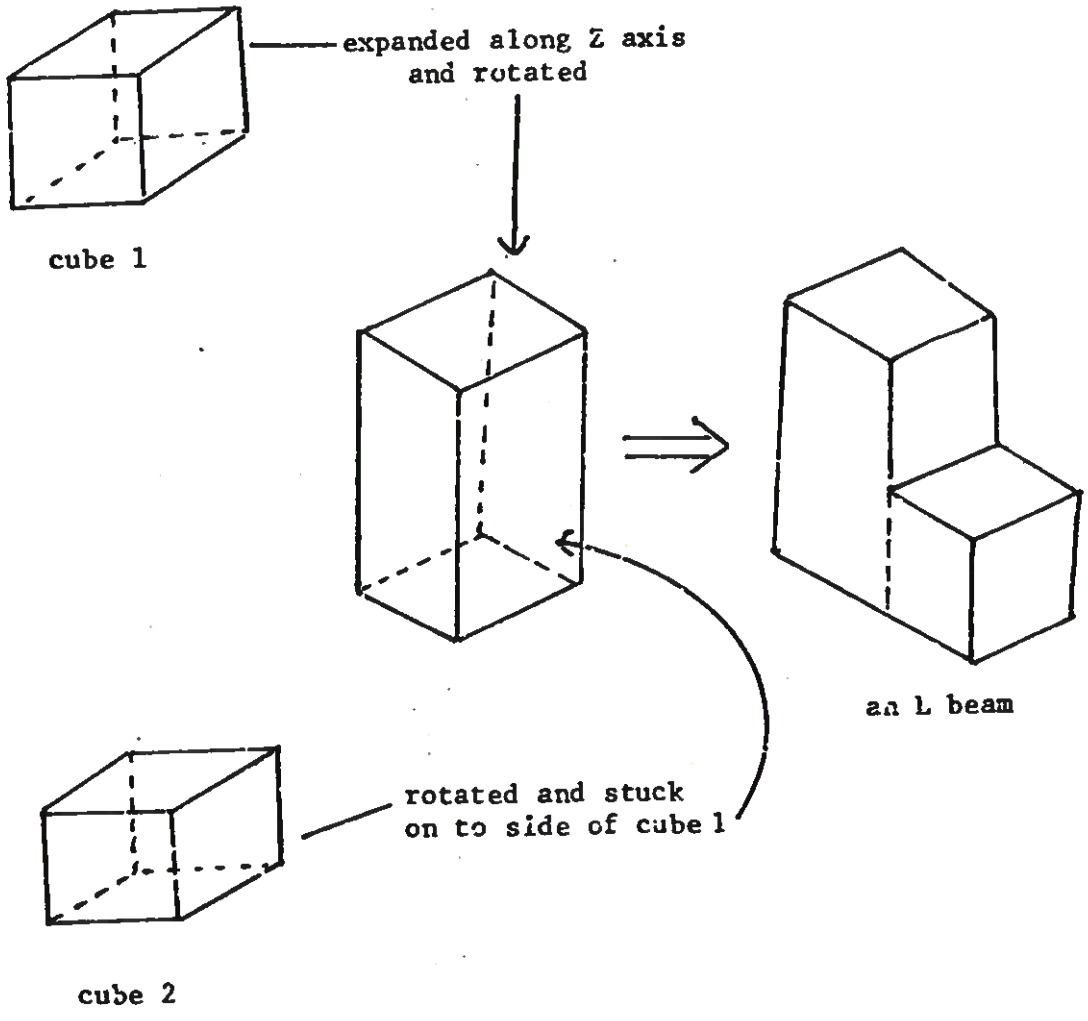
(1)



Cube

2 x 1 cuboid

(2)



cube 1

an L beam

cube 2

Figure 2

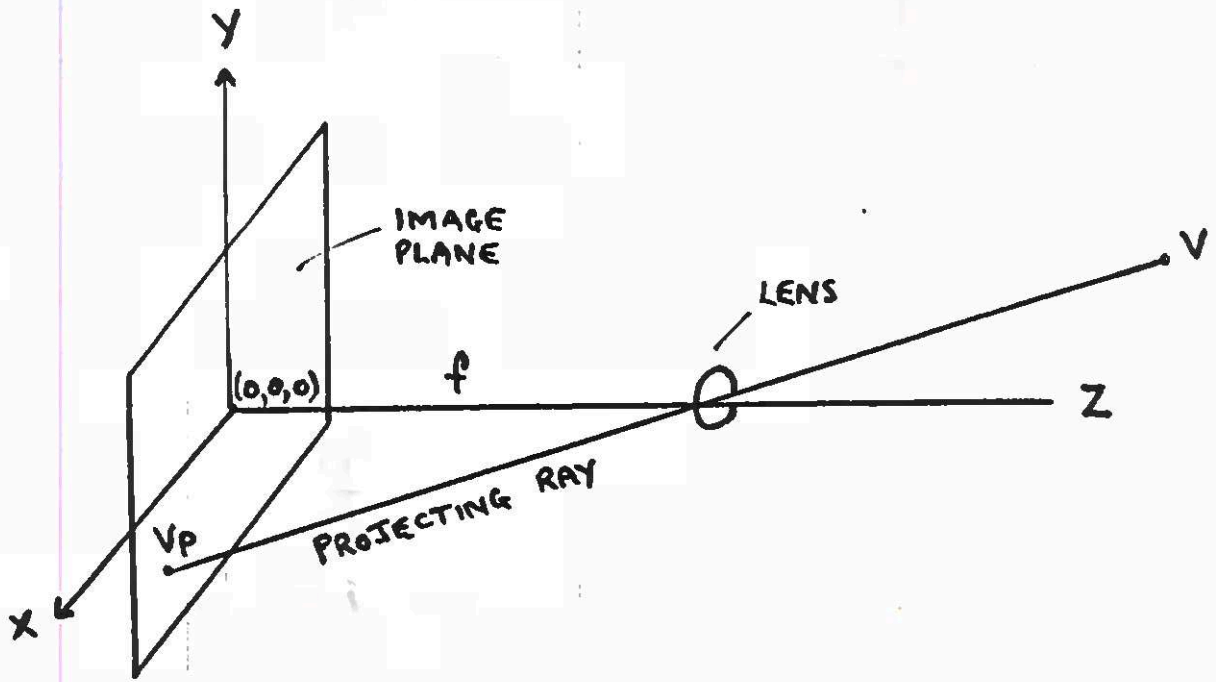


Figure 3

MODEL OF PICTURE TAKING PROCESS.

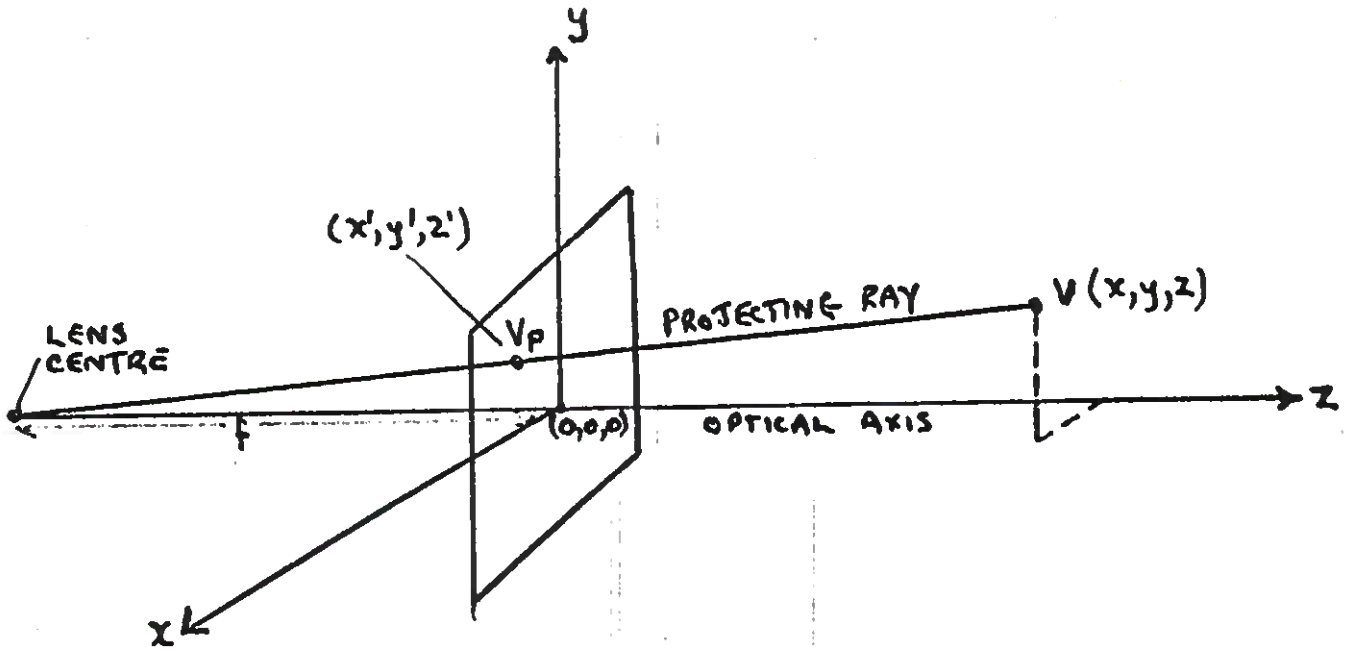


Figure 4 CENTRAL PROJECTION

THE IMAGE V_p OF POINT V AT LOCATION (x,y,z) IS GIVEN BY

$$x' = \frac{fx}{f+z} \qquad y' = \frac{fy}{f+z} \qquad z' = 0$$

THIS TRANSFORMATION IS NON-LINEAR SINCE IT INVOLVES A DIVISION.

SO, TOO, IS THE INVERSE TRANSFORMATION

$$x = \frac{x'(f-z)}{f} \qquad y = \frac{y'(f-z)}{f}$$

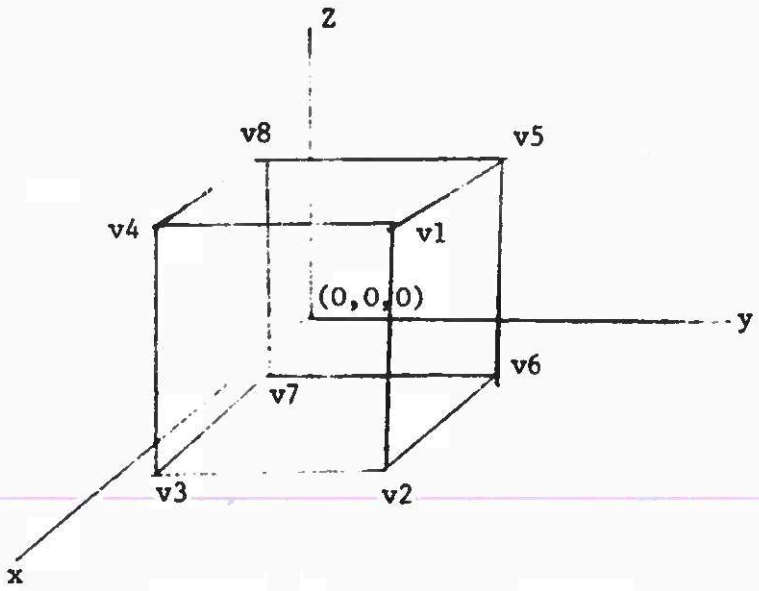


Figure 5

$$\begin{bmatrix} x_1 & y_1 & z_1 & h_1 \\ x_2 & y_2 & z_2 & h_2 \\ & & & \\ x_8 & y_8 & z_8 & h_8 \end{bmatrix} \begin{bmatrix} \\ 4 \times 4 \\ \text{Transformation} \\ \text{Matrix} \\ \\ \end{bmatrix} = \begin{bmatrix} x_1' & y_1' & z_1' & h_1' \\ x_2' & y_2' & z_2' & h_2' \\ & & & \\ x_8' & y_8' & z_8' & h_8' \end{bmatrix}$$

8 x 4 matrix of original coordinates

8 x 4 matrix of transformed coordinates

Figure 6

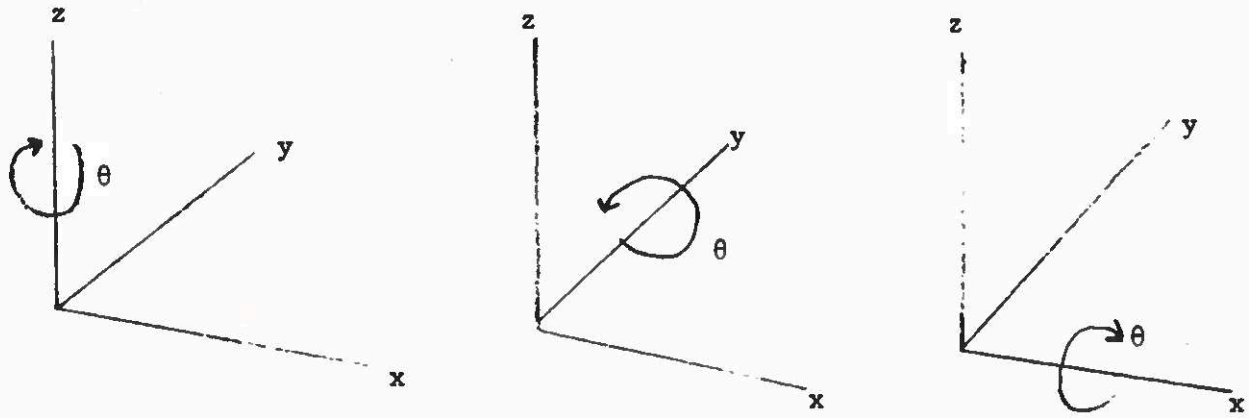
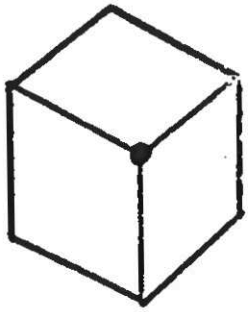


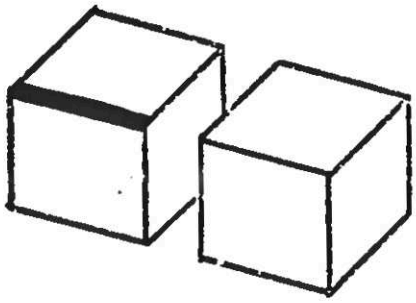
Figure 7

1. Find a vertex surrounded by 3 approved polygons.



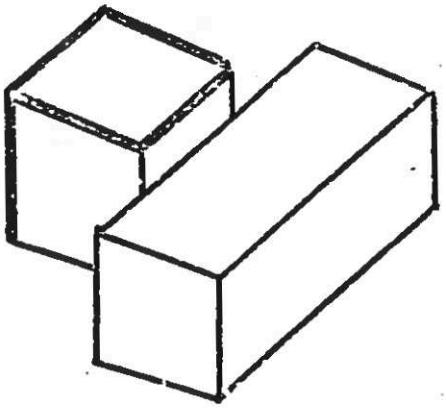
(a)

2. Find a line with an approved polygon on each side.



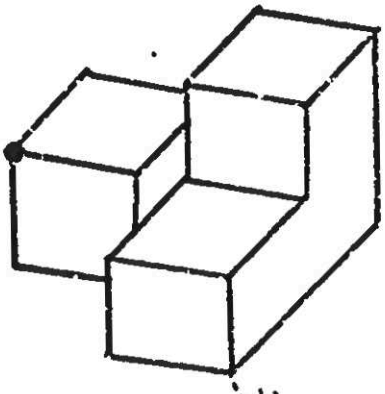
(b)

3. Find an approved polygon with an extra line coming from one vertex.



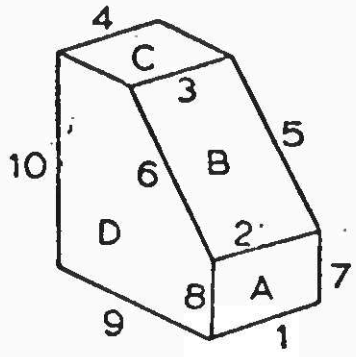
(c)

4. Find a vertex with 3 lines coming from it.

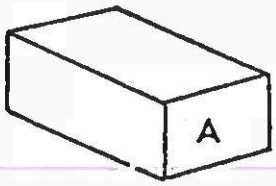


(d)

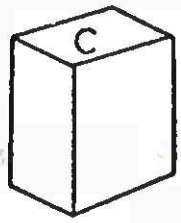
(a)



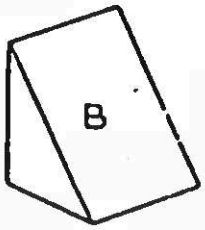
(b)



(c)



(d)



(e)

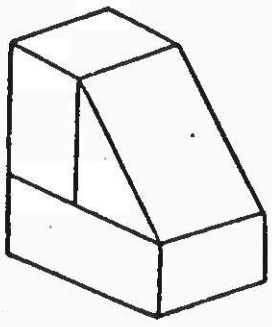


Figure 9

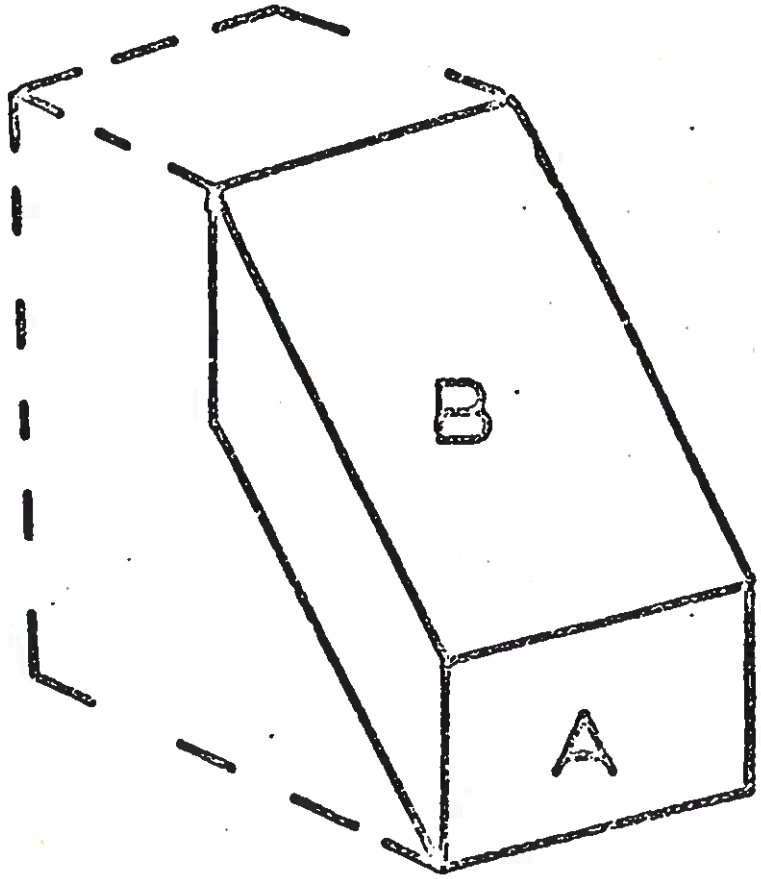


Figure 10

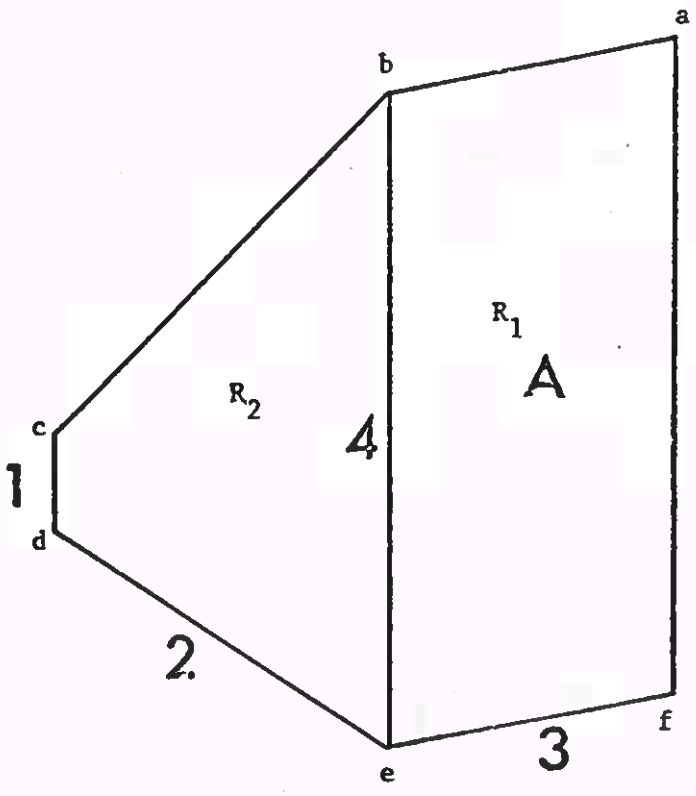
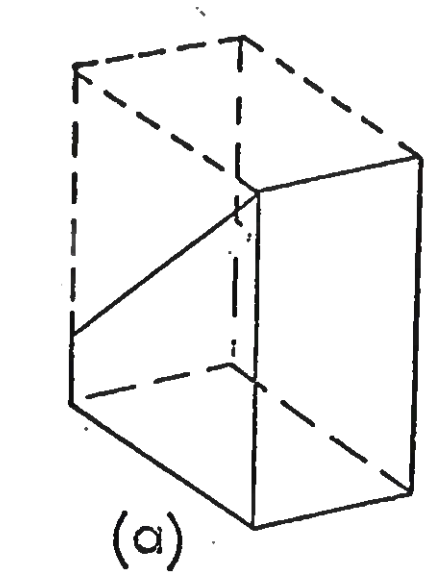
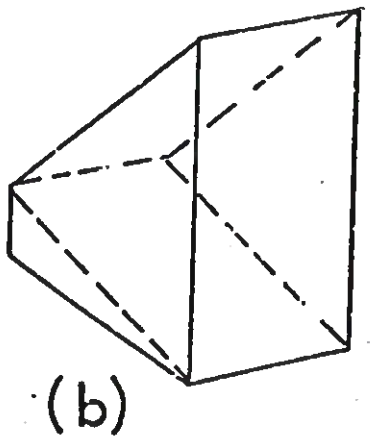


Figure 11





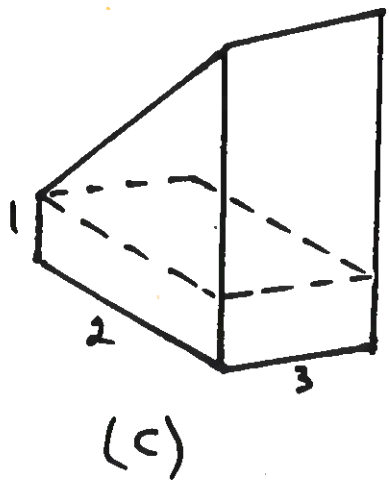


Figure 12

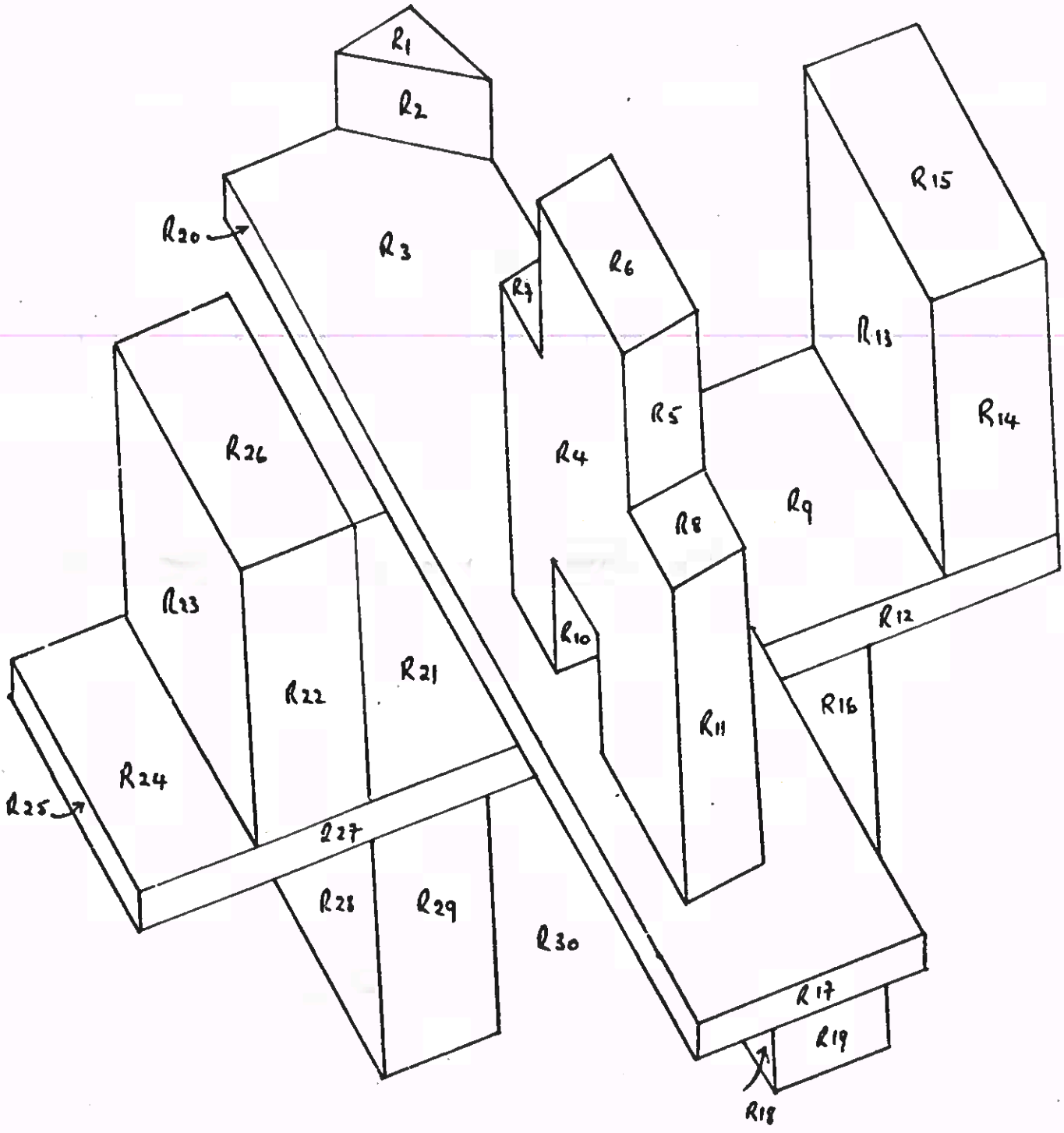

R1
R2
R3
R20
R7
R6
R15
R13
R5
R14
R4
R26
R8
R9
R23
R10
R12
R22
R21
R11
R16
R24
R25
R27
R28
R29
R30
R17
R19
R18

Figure 13

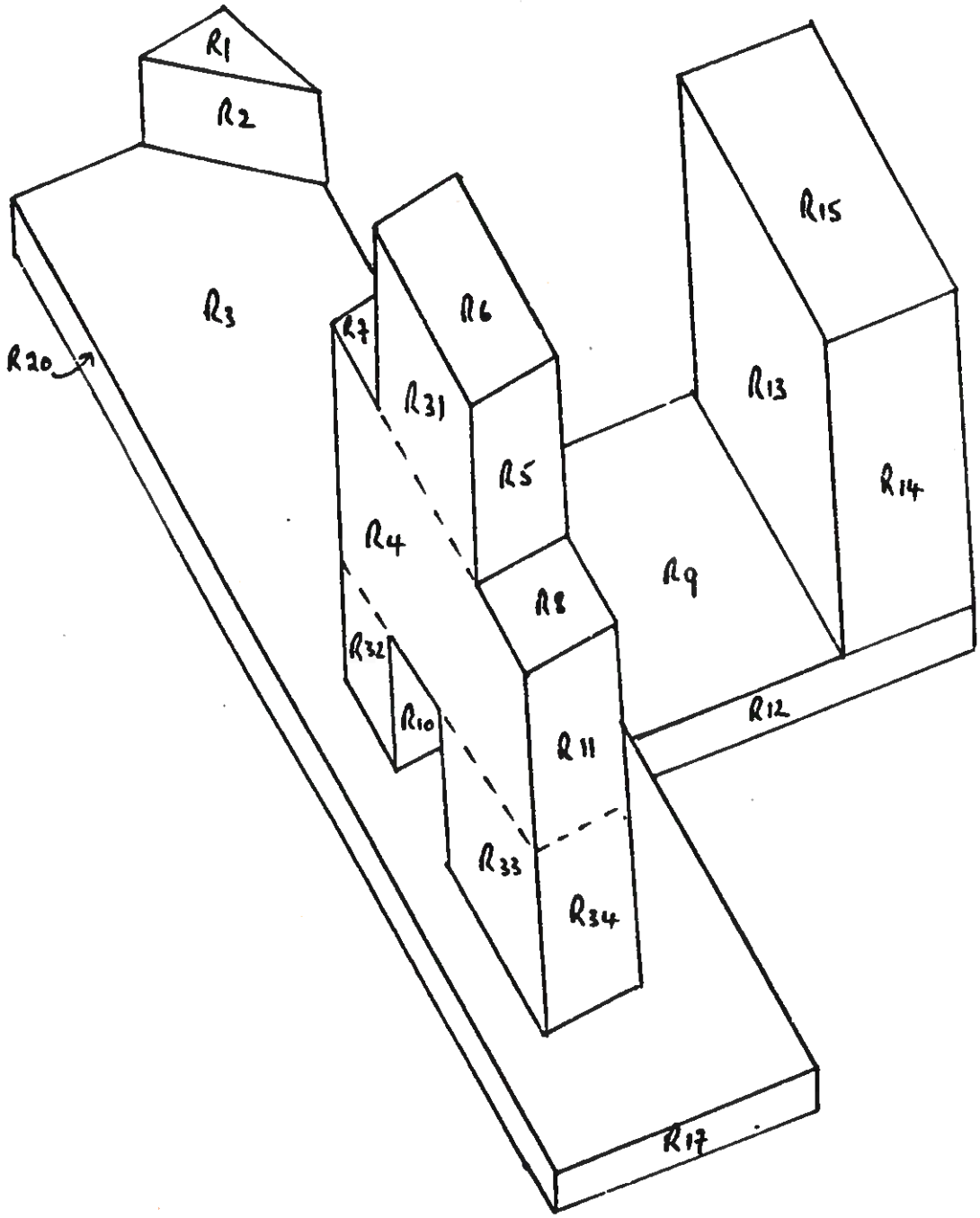

R1
R2
R3
R20
R7
R6
R31
R5
R4
R8
R32
R10
R11
R33
R34
R9
R15
R13
R14
R12
R17

Figure 14

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

AI-2 JIM HOWE

Recognising 3-D objects

Although Roberts claimed that his program recognised objects, it seems preferable to think of it as a model driven segmentation program since it decomposes bodies into their constituent parts. But recognition of objects also implies the existence of models of these objects which can be invoked, and matched with a scene description to achieve recognition. The question is what form might these models take, bearing in mind that they have to represent object invariances?

Dealing with polyhedral objects first of all, consider a simple arch, built from three blocks. We know that Roberts' program would recognise three transformed cubes, but it would not "see" an arch. So how is an arch defined? The answer is that an arch is defined in terms of the invariant relationships between the constituent bodies. A convenient way of representing scenes is to use a directed graph, where objects and object parts are represented by nodes, and relationships between parts are characterised by arcs between nodes. In the blocks world, typical objects, and their properties, include:

wedge	brick	small
cube	object	medium-size
rectangle	standing	large
triangle	lying	

and typical types of relationships include:

one-part-is	a-kind-of	left-of
in-front-of	group-of	right-of
has-property-of	supported-by	married-to
abut		

Let's apply this technique to the HOUSE scene shown in Figure 1(a). Basically, it is just one wedge on top of one brick. Its graph description is given in Figure 1(b). The top node represents the scene, while its daughter nodes (A and B) represent the two parts. The arcs are labelled ONE-PART-IS to denote the relationship between scene and parts. In turn, each daughter node has its own daughter node which describes the type of part. Part 'A' is a wedge and part 'B' is a brick: these are represented by the relationship A-KIND-OF. Finally, there is one more relationship between parts 'A' and 'B', the support relationship SUPPORTED-BY.

Let's turn now to the scene shown in Figure 2(a). Its graph structure shown in Figure 2(b), differs in one respect: the absence of the SUPPORTED-BY relationship. If we are prepared to see Figure 1(a) as a HOUSE, but reject Figure 2(a) as a HOUSE, this suggests that the SUPPORTED-BY relationship between the parts is an essential feature of a HOUSE. This can be captured in the graph by modifying the SUPPORTED-BY description to a MUST-BE-SUPPORTED-BY description, as shown in Figure 3. Again, if we wish to reject the scenes shown in Figure 4 as examples of HOUSE, the A-KIND-OF description attached to to the arcs from parts to part types must be altered to become a MUST-BE-A-KIND-OF description (Figure 5).

Construction of a pedestal model proceeds in much the same way (Figure 6):

i) the top object MUST-BE-A-KIND-OF brick

ii) the upper object MUST-NOT-MARRY the lower

The scene in Figure 7(a) forces the top object to be a brick while the scene in Figure 7(b) forces the lower one to be a brick, too. Figure 7(c) forces support and Figure 7(d) forbids the MARRYS relation.

The arch scene in Figure 8(a) introduces a wider variety of differences and produces a more complex graph, as shown in Figure 9. The scene given in Figure 8(b) indicates that the MARRY relationship is not necessary; the scene in Figure 8(c) shows that the support relationships are crucial i.e. MUST-BE-SUPPORTED pointers are required; by placing the standing bricks in contact, scene 8(d) highlights the key feature of an arch, namely, the space between the supports, handled by the pointer MUST-NOT-MARRY, and finally scene 8(e) suggests that anything in the class OBJECT will do for the top brick.

When a scene involves groups of objects, this feature is handled by specifying relationships between typical members. Figure 10(a) shows a composite column. Figure 10(b) introduces the MUST-MARRY pointer; Figure 10(c) relaxes the object type from BRICK; Figure 10(d) emphasises SUPPORTED-BY by changing it to MUST-BE-SUPPORTED-BY; through Figure 10(e) the notion of a group (a group implies at least three members) is emphasised by changing the pointer on the arc to GROUP from ONE-PART-IS to ONE-PART-MUST-BE, and Figure 10(f) generalises the number of objects by introducing the node INTEGER and the relationship NUMBER-OF-MEMBERS, allowing a column to have any number of objects greater than 2.

Another example of the use of the GROUP node is the table shown in Figure 11(a). "Near miss" scenes are shown as Figures 11(b) to Figure 11(e).

What we have been looking at are examples of structural descriptions generated by a program developed in the early 1970's by Pat Winston at M.I.T.

Each description was constructed by feeding the program with examples and near misses. We can view his program as one method of building prototypes for use in a visual recognition system.

Matching

We turn now to look at another problem, that of matching a structural description of a scene with a stored structural description of prototype objects to achieve recognition.

Since structural descriptions can be viewed as graphs, our initial inclination is to compare pairs of graphs (or subgraphs) either by discovering that they are isomorphic or by generating a maximal clique of the match graph. Unfortunately, both methods are computationally expensive. Also, the graphs being compared may differ due to imperfect edge evidence, missing parts, extra parts, distortion due to scene perspective, and so on.

We turn, instead to consider methods based upon decision tree matching. Decision trees are useful when it is known that certain features are more reliably extracted than others, and that certain relations are easier to establish or are more germane to the success of a match.

1. Matching structural descriptions

We begin by considering how Winston's program 'recognised' structural descriptions derived from typical blocks world scenes containing single objects. Basically, the program compares the unknown description with each stored description (ordered by degree of similarity), and generates a set of 'difference descriptions' to characterize the discrepancies. Recollect that Winston's structural descriptions contain arcs such as MUST-NOT and MUST-BE expressing forbidden or mandatory relations. If one of the these essential relationships is violated, a match is not permitted. These outright rejections are easy cases. In practise, the harder cases are those where the unknown description matches more than one stored description. This is dealt with by associating numerical values with elements in the descriptions and computing a weighted sum of differences to express the degree of match in each case.

Scenes containing more than one object are harder to handle since evidence may be missing from the input description due to occlusion. Winston's program makes use of two heuristics:

1. Objects that seem to have been stacked and could be of the same type are judged to be the same.

2. A match is not rejected if an essential property is missing from the input description since its absence may be due to occlusion.

One result of the latter is that the program sometimes misses the best match in favour of the first possible match.

2. Backtrack search

Next, we consider the approach taken by Nevatia in a program that recognizes instances of biological shapes (horses, dolls, snakes etc). Prototypes are segmented into parts, each with a central axis and a cross section. They also record information about the connectivity of the sub-parts, and descriptions of the shape and joint type of the parts. The prototypes (i.e. models) are generated by the machine by the same process that later extracts descriptions of the image for recognition purposes.

The matching process is essentially a depth first tree search, with initial choices being constrained by "distinguished pieces" of the image. Nevatia uses the example of the doll, shown in Figure 12, to illustrate the process. In its case, the head and trunk are the "distinguished pieces" because they are the widest parts. Before the input description is matched to a prototype, a connection graph of its pieces is formed, as shown in Figure 13. Due to noise in the input data, two of the joints are missing.

On the basis of the distinguished pieces in the input instance, which match those in the prototype, the program decides that the instance could be a doll or a horse. Both possibilities are carefully evaluated. A schematic view of the match between the input description and the doll prototype is shown as Figure 14. The final choice between candidates takes account of both the connectivity relations, and the quality of the matches between individual pieces. When the input description was matched with the horse prototype, some parts were missing from the input description. However, this did not rule out the horse since the missing part might have been hidden from the camera's view. When compared with the doll prototype, the piece matching was superior so this was the 'preferred' solution.

The examples that we have considered above both involved matching the input description to the full set of prototypes. However, if the number of stored models in memory is large, this approach is impractical. Instead, there is a need for an invocation mechanism that will select a candidate subset of the stored models. Typically, object features, such as the connectivity of distinguished components and their type (remember Robert's approach), are used to compute an index (cue) into the model database. Each stored model is equipped with an index. So, given an index computed from an object in the image, a list of models with the same index is immediately available. Also, several indices may be computed for a single model, to ease the invocation task. In practice, this invocation problem is harder than it might seem, and is a subject for much needed research.

REFERENCES

Winston, P. (1975) Learning Structural Descriptions from Examples. In Psychology of Computer Vision. New York: McGraw-Hill.

Nevatia, R. (1974) Structured Descriptions of Complex Curved Objects for Recognition and Visual Memory. AIM-250, Stanford AI Laboratory.

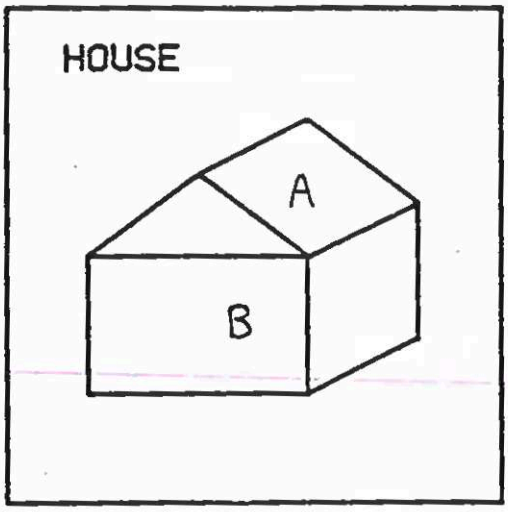


Figure 1(a)

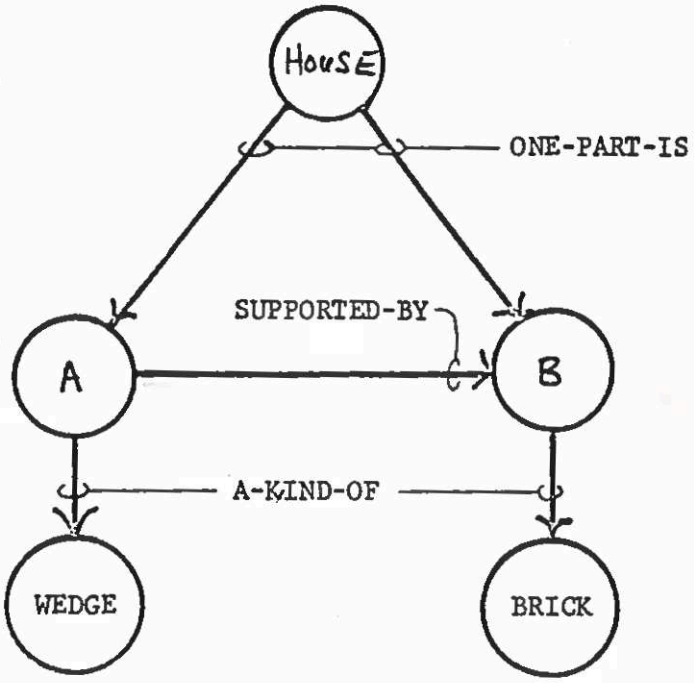


Figure 1(b)

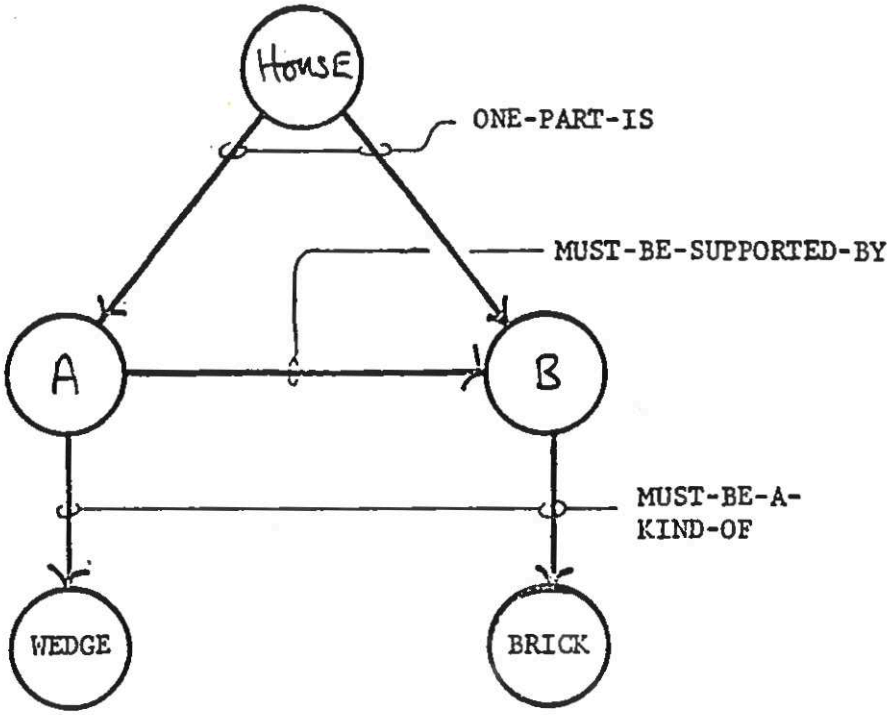


Figure 5.

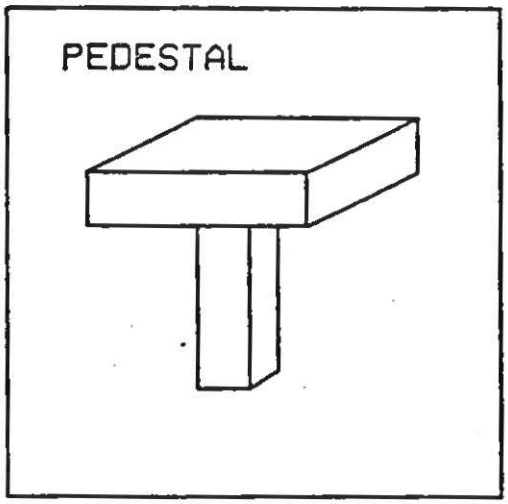


Figure 6

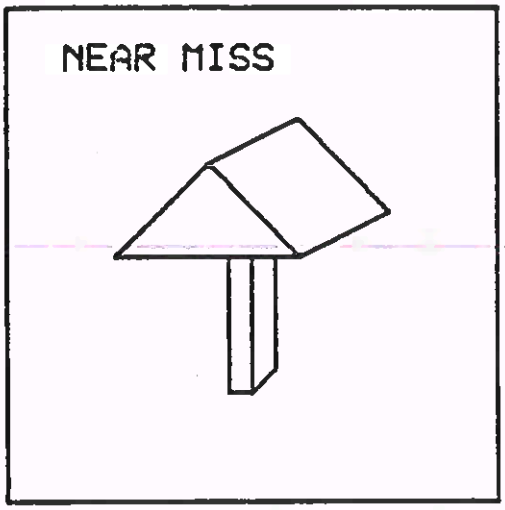


Figure 7(a)

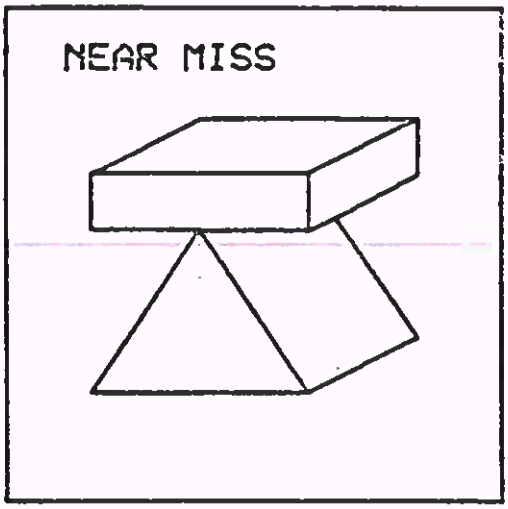


Figure 7(b)

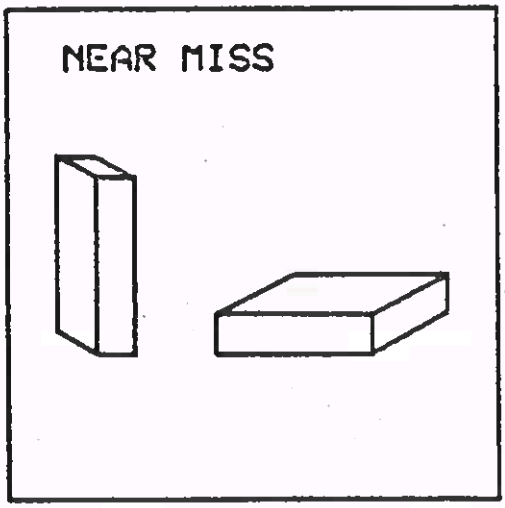


Figure 7(c)

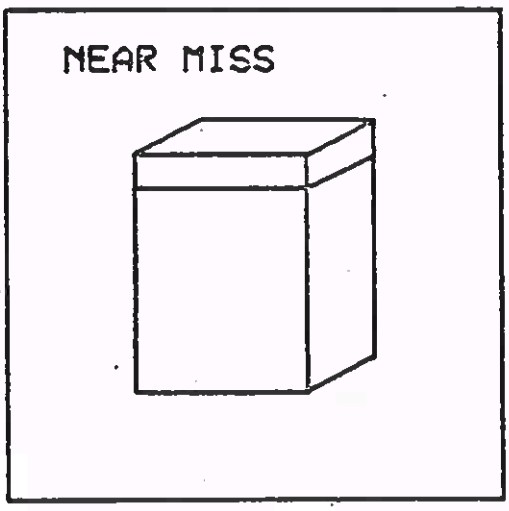


Figure 7(d)

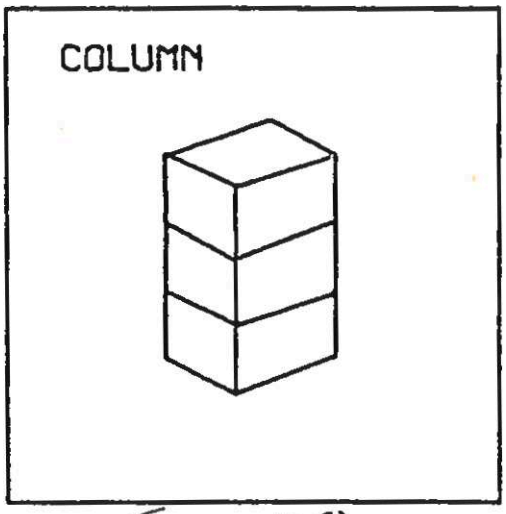


Figure 10(a)

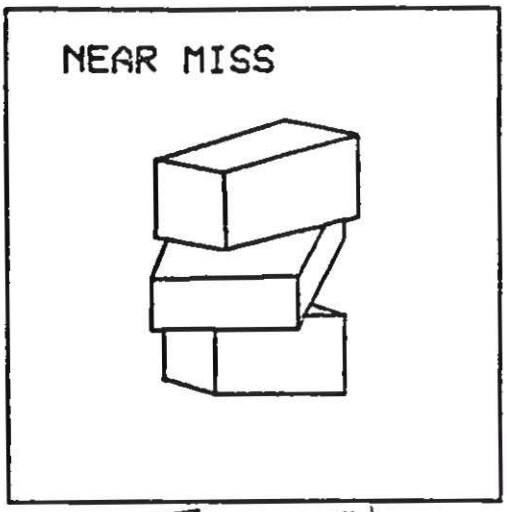


Figure 10(b)

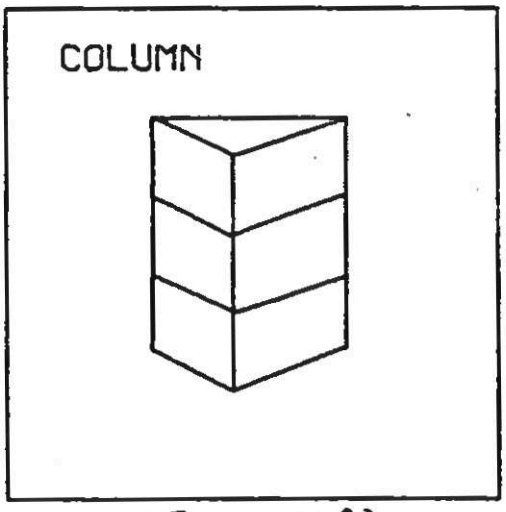


Figure 10(c)

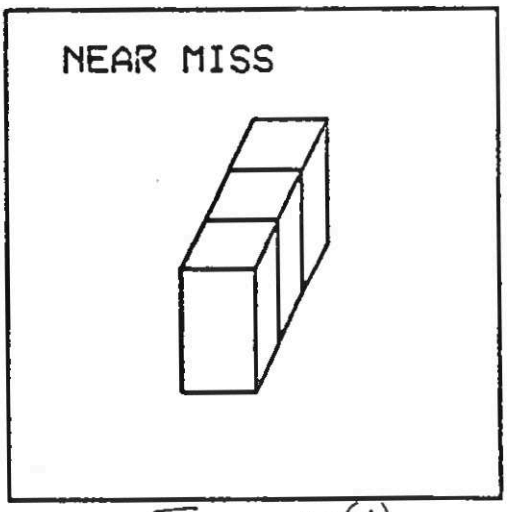


Figure 10(d)

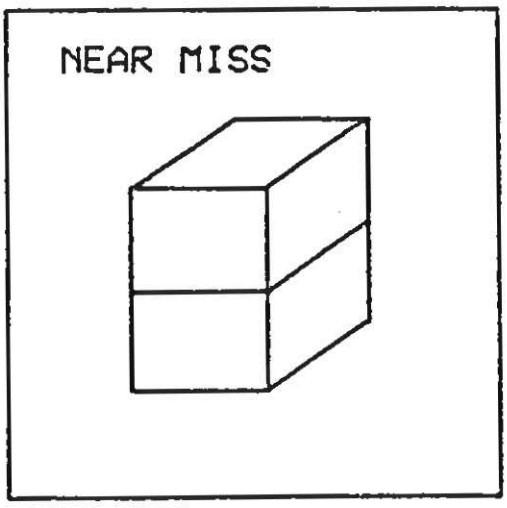


Figure 10(e)

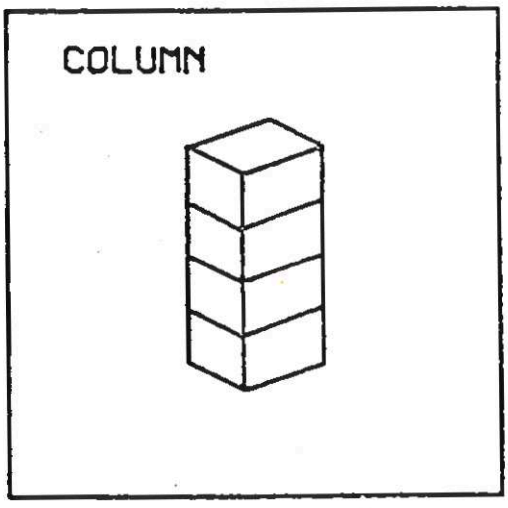


Figure 10(f)

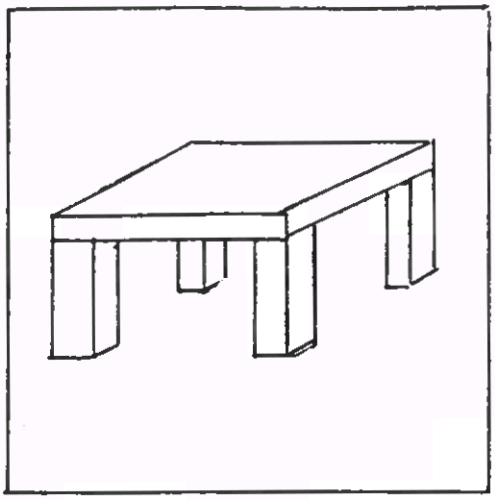


Figure 11 (a)

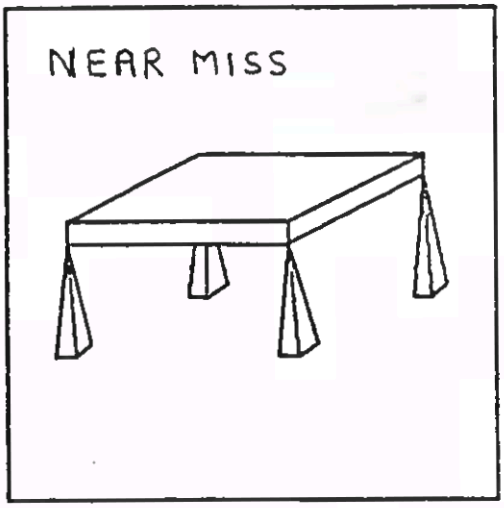


Figure 11 (b)

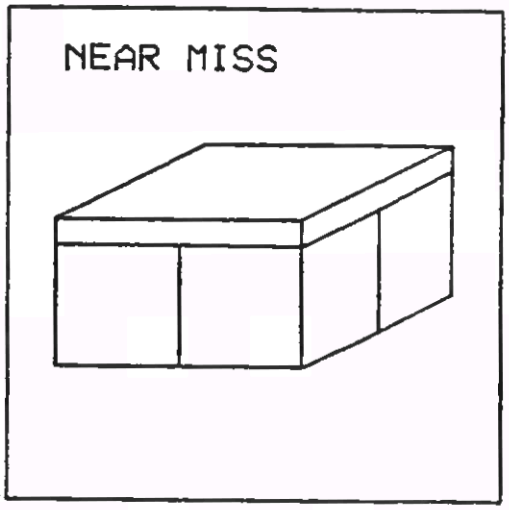


Figure 11 (c)

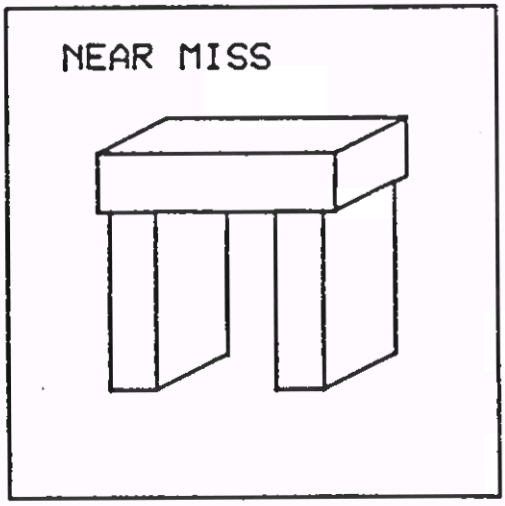


Figure 11 (d)

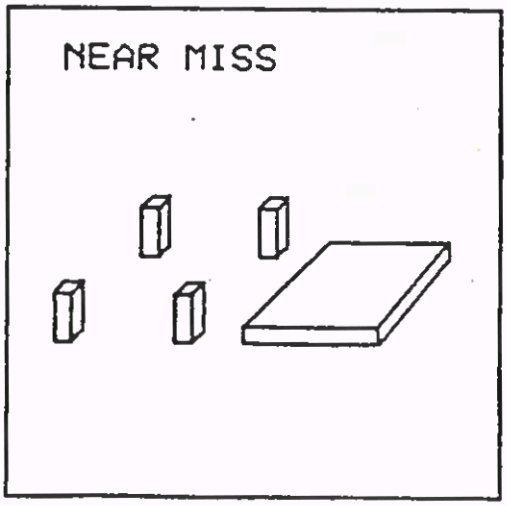


Figure 11 (e)

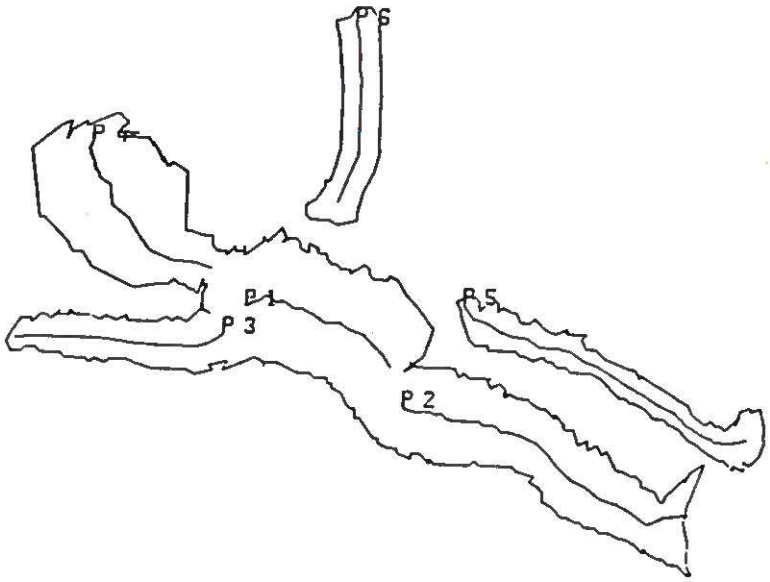


Fig. 12 A view of a doll, with derived structure.

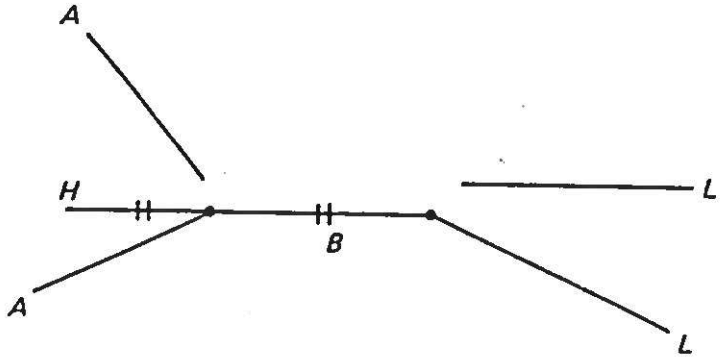


FIGURE 13 CONNECTION GRAPH OF DOLL.

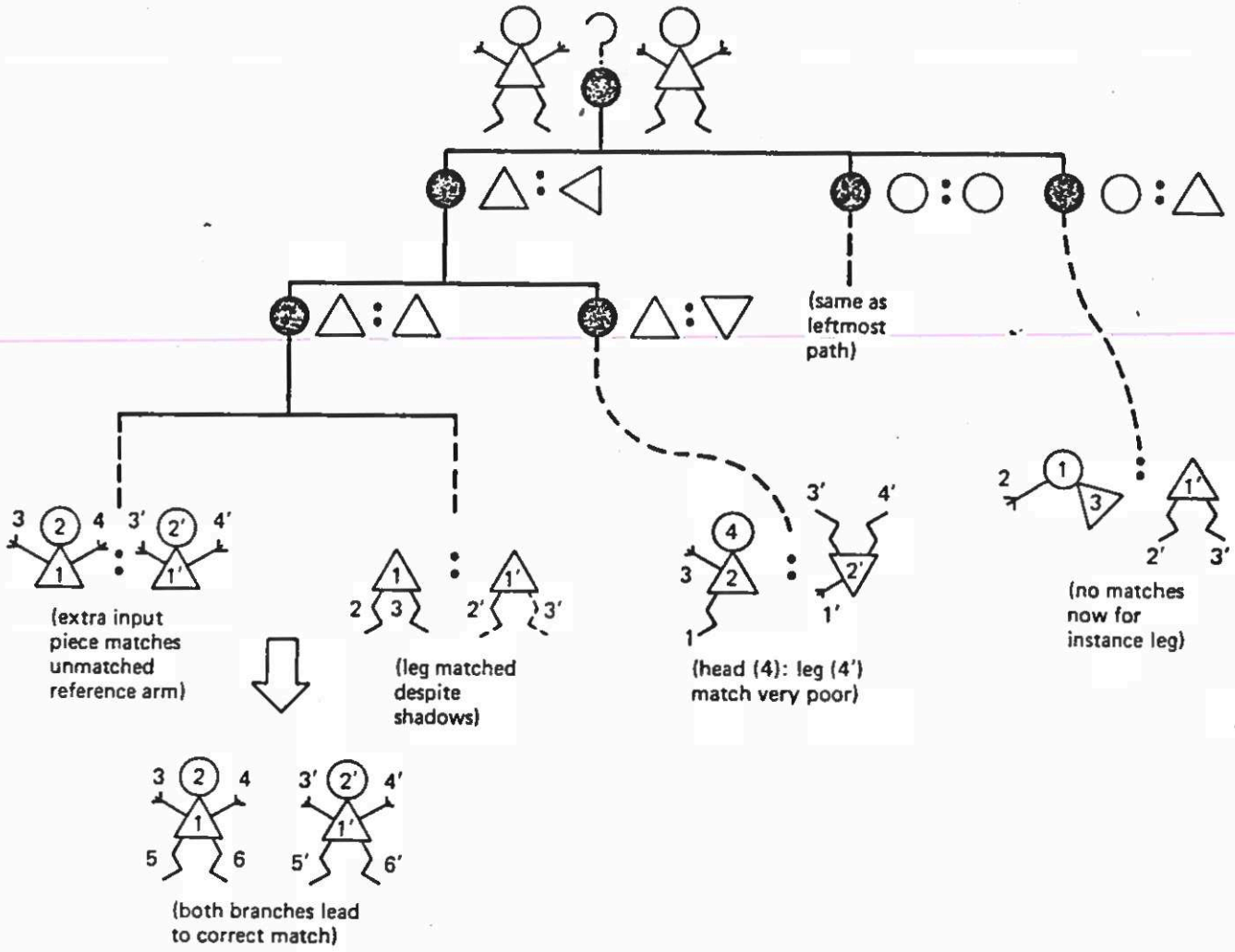


Fig. 14 A pictorial guide to the combinations tried by the matcher establishing the best correspondence of the doll input with the doll reference. The graphic shapes are purely pedagogical: the program deals with symbolic connectivity information and geometric measurements. Inferences and discoveries made by the program while matching are given in the diagram. A:B means that structure A is matched with structure B, with the numbered substructures of A matching their primed counterparts in B. (from Ballard & Brown, 1982)