Computer Vision for Line Drawings

Cullen Jennings

Modern maps and engineering diagrams are usually constructed and stored using GIS or CAD systems. A large number of drawings, however, exists only in a paper form. This thesis examines the problem of automatically converting such drawings and maps from raster image to high quality vector GIS or CAD forms.

This thesis begins with a review of previous work in the area and then proposes a new method based on findings about how human vision works and domain specific knowledge. Another system based on the classical work in this area is presented, to which the new system is compared. This comparison shows that the method proposed here obtains substantially better results than classical methods. The time a human operator could expect to spend correcting the errors created by this system would be less than one tenth of the time required to correct the errors created by a classical vectorization system.

Most new engineering drawings and maps are produced with CAD or geographic information systems (GIS). These electronic formats provide substantially more flexibility than paper does: updates to a document can easily be reproduced at all the sites that use it; documents can be combined or have additional information incorporated into them; and documents can by analyzed by computer. For example, from a GIS system with maps a user might identify the location of the house nearest to a river where chemicals had been spilled. From CAD electrical diagrams, a computer simulation of a circuit might be created. In addition, ease of archiving and updating make attributed electronic forms for document information very valuable.

Although new documents are often drafted in electronic form, many older documents exist only on paper. Concerted efforts are being made worldwide to get this information into electronic databases. Maps are being transformed from paper form into GIS so that, correlated with satellite images, they can aid in the evaluation and optimization of crop growth. Well logs – graphs of data from instruments that have been lowered down oil and gas wells – are being digitized so that computer simulations can be used to predict the size of the reserves in the reservoir. Telephone companies are transforming wiring diagrams of connections into GIS, so that when wiring changes are made in one location, all the other locations can be informed of the change.

When a drawing is scanned into a computer, it is represented by a raster, a two dimensional array of black or white picture elements. The computer can do almost nothing with a raster except display it. To make the information usable by CAD or GIS, it must be converted to a vector format, a format in which each line in the drawing is represented in the computer by a record that describes the characteristics of that line. The characteristics include things like start point, end point, and width.

This thesis concentrates on the problems of automatic computer conversion from raster to vector representations of drawings. Perfect raster to vector conversion is impossible because different vector drawings can result in identical raster images. If two straight lines have one end point in common and the lines are collinear, they are indistinguishable from one straight line.

During the early 1980s commercial enterprises began to seek ways to scan these documents and automatically translate them into vector formats. One of the first systems was Laser Scan's FASTRACK system in 1978 [Fulf:81]. Some of the more important programs have been Audre's system, AutoDesk's CAD/Camera, DataSpan's RVCS [RVCS:90], GTX's GTXRaster CAD [Macl:91] [Bhas:89], Hitachi's CADCore [Geor:90] [Saka:89], Intergraph's I/VEC [IVEC:92] and I/GEOVEC [IGEO:92], Scorpion Technologies' SRV [SVR:90], QC Data's MEWS [QC:87], Universal Systems' SAMI [SAMI:92], and Winchester Data Products' VECTRESS [VECT:92]. Many of these systems claimed to be considerably faster than manual conversion. Audre claimed their system was seven times faster and cost one tenth as much [Bono:88]. A 1986 Anderson Report suggested that the available automatic conversion systems ran three to four times faster than most manual systems and cost about half as much [Ande:86].

Lately many companies, sometimes receiving public funding, have been working on this problem, but they have not been trumpeting their success. QC Data's MEWS project, the result of a $700,000 joint venture between the Alberta Research Council and QC Data's Research and Development Department [QC:87], is no longer in use – manual redrafting with digitizing boards has proven faster. DataSpan, another company that has produced such a product, is no longer in business. Intergraph, which has done a great deal of research in this area, suggests that manual conversion be used for most projects. Companies that claim to have created systems faster than manual redrafting include Winchester Data Products, GTX (a leader in the area [Macl:91]), and Hitachi. Currently the most popular low–end system is Adobe's Streamline [Powl:92]. All of these systems generally fail to provide a definitive solution to the conversion problem, and some are even slower than redrafting a document on a digitizing board. Despite the considerable amounts of time and money spent, much of the conversion done today still involves the use of a manual digitizing board to redraft the document into a

CAD or a GIS.

Much academic effort has also been devoted to conversion. Two of the first systems were described by Ramachandran [Rama:80] and Clement [Clem:81]. A more recent system that incorporates much previous research is described by Jennings and Flanagan [Jenn:93]. The bibliography for this thesis identifies a couple of hundred researchers in the field.

One of the important types of line drawings requiring conversion from raster to vector formats is engineering diagrams. The many problems these pose for conversion systems are characteristic of the problems that arise in converting most types of line drawings. Engineering diagrams consist mainly of lines, arcs, and symbols, including text. One of the more complex problems in converting these diagrams to digital form is that the lines and arcs are often dashed. A recognition system must take all the segments of a dashed line and represent them as one line with a particular dash pattern, not as a bunch of short, solid lines. Complicating matters further is the fact that lines often intersect at exact angles like 30^o; having them converted to 30.1^o is often unacceptable. As well, lines running through symbols make them harder to detect. Typical commercial problems involve converting tens of thousands of large engineering diagrams into digital form [Naga:88]. For example, the Saint John Shipbuilding project is converting 22,000 drawings in the first phase [Sea:90].

This thesis examines the use of computer vision systems in recognizing two dimensional line drawings, such as engineering diagrams, and proposes a new conversion method. Much research has been done on this problem, but automatic conversion that takes an order of magnitude less human operator time than manual redrafting is currently impossible for most drawings. It seems unlikely that any automatic system will ever perform 100% correctly on complex documents, but even a system that was 90% correct and helped the user fix the remaining 10% would considerably reduce conversion time. Compared to three dimensional vision in an real world environment, conversion is likely a simple vision problem, but it is still far from solved.

The next section of this thesis describes previous work in the area. Then comes a discussion of the scanners and preprocessing methods available, followed by an examination of state of the art conversion techniques. The conversion technique proposed in this thesis is described next. Finally it is compared to the state of the art techniques.

Chapter 1

Previous Work

A thorough review of the literature reveals pertinent information about most of the relevant aspects of the vectorization process. This chapter describes and briefly evaluates the significant contributions to the field of conversion, suggesting how the many techniques have been combined in the past.

1.0 Classical Conversion

The steps taken in the classical method for converting images used by Musavi *et al* [Musa:88] are:

1. Scanning

   Scan the paper image to produce a grayscale raster image;

2. Thresholding

   Form a binary image from the grayscale image;

3. Salt and Pepper Filtering

   Remove the isolated black and white pixels that are noise;

4. Thinning

   Thin the lines on the image to one pixel wide;

5. Chain Coding

   Follow the thinned lines and produce chain codes representing them;

6. Vector Reduction

   Reduce straight segments of the chain code into long lines that represent the chain codes.

Most commercial and research systems use schemes much like Musavi's. Such systems are described by Suzuki and Yamada [Suzu:90], Kikkawa *et al* [Kikk:84], Johnson and Bird [John:90], Hoshino *et al* [Hosh:86], Bixler and Sanford [Bixl:85], and Suetens *et al* [Suet:81]. Kasturi *et al*give an example of a similar complete system [Kast:90]. One of the earliest noncommercial systems of this type is described by Clement [Clem:81]. Nagasamy and Langrana present a similar system that does vector reduction to conic sections [Naga:90]. The rest of this chapter describes in detail the steps used in these methods.

2.0 Scanning

Almost all current scanners are based on CCD camera technology and produce 8–bit grayscale images. A resolution of 300 dpi is standard for the low end scanners, while 800 dpi is common in commercial-grade scanners. A more detailed discussion of scanners and the problems encountered in scanning is given in chapter 3. Vectorization with colour scanners has been studied by Kakumoto *et al* [Kaku:83] but did not result in appreciably better results.

3.0 Thresholding

The problem of thresholding and segmenting document images has been carefully studied [Taxt:89]. Since scanners now produce high quality results with little noise, thresholding is usually easy. Often using a fixed threshold for the whole image works fine. Complex thresholding is required only when the original paper quality is poor or when the document being scanned has faint lines produced by a photocopier low on toner or by spilled water or coffee. Dirty images are thresholded well by Parker's algorithm [Park:91]. Good surveys that provide an overview of the method used in this thesis are given by Fu and Mui [Fu:81] and Sahoo *et al* [Saho:88].

4.0 Salt and Pepper Filter

CCD cameras produce an effect called blooming, in which pixels become over or under exposed so that the scan looks as if it has been lightly sprinkled with salt and pepper. The thresholding scheme can correct for this effect, however, if a salt and pepper filter is incorporated into the thresholding. Such a filter simply finds each pixel whose colour is the opposite of all the pixels around it and changes its pixel value so that it matches its surroundings. This scheme is demonstrated well by Bury [Bury:89].

5.0 Thinning

Thinning is the process of taking an image with thick lines and producing an image that is basically the same but contains only thin lines. Diagrams in chapter 4 show the effect of thinning.

Most thinning algorithms approximate, or compute exactly, the Medial Axis Transform (MAT), which was originally defined by Montanari [Mont:69]. The most commonly used algorithm of this type is presented by Zhang and Suen [Zhan:84], but others have been given by Arcelli and Baja [Arce:85], Deutsch [Deut:72], Kwok [Kwok:88], Naccache and Shinghal [Nacc:84], O'Gorman [OGor:90], Pavlidis [Pavl:82], and many others. The first algorithm of this type was likely Hilditch's [Hild:69]. All MAT techniques result in artifacts highly undesirable for the vectorization of engineering diagrams. One such artifact is erosion hairs, tiny lines extending laterally from the skeleton of the image (see figure 4.22). Vectorizers interpret these erosion hairs as significant, making the final product a fuzzy rendition of the original.

Several thinning methods that do not generate MAT type images have been developed, one of which is described by Baruch [Baru:88]. These techniques often produce other artifacts just as undesirable as those produced through MAT techniques. One method that tries to eliminate some of the artifacts that happen at intersections is described by Govindan and Shivaprasad [Govi:87]. This method fails to prevent artifacts in some other cases, however. Li and Suen describe another method for thinning [Li:91] which uses knowledge about where traditional thinning fails but which still fails in many cases. A final method works directly from the grayscale image and produces a binary skeleton [Yu:90]. It does an excellent job, but it works only on lines running at angles that are multiples of 45^o. Partial thinning methods for vectorization are used by Espelid and Eileng [Espe:89]. Paler and Kittler do thinning on grayscale images [Pale:83]. An interesting thinning technique based on examining the edges is given by Sinha [Sinh:87]. It does not rely on the medial axis transform but still gets poor results.

A substantial body of literature about thinning exists. Pavlidis presents one good survey of the topic [Pavl:82b], as do Kwok [Kwok:88], Naccache and Shinghal [Nacc:84], and Smith [Smit:87]. An excellent overall survey is by Lam *et al* [Lam:92].

A study of metrics for measuring how “good” a skeleton is, a definition of thinning, and another thinning algorithm are presented by Parker and Jennings [Park:92]. This paper shows that there is currently no widely accepted definition of thinning and that it is therefore not surprising that there should be no exceptionally good thinning algorithms. This paper presents sample drawings so diabolical that it is not even intuitively clear to a human user what the correct skeleton should be. For an simple example, what should be the skeleton of a solid disk? If the answer is a single point in the middle, then should a solid ellipse have the same skeleton? What about a solid ellipse elongated to the point that it is almost a line segment? The metrics described in this paper can be used in a quantitative comparison of various thinning methods.

6.0 Chain Coding

Chain coding is the process of tracing the lines made by the pixels on the thinned image. Freeman originally proposed it [Free:61] and later presented algorithms for manipulating these chain codes [Free:74] [Free:77], collaborating with Davis on the last paper. All the chain codes used here will conform to Freeman's conventions, and many of the algorithms used to manipulate chain codes come from one of these papers. A detailed discussion of chain coding is given in chapter 5.

A dedicated hardware system that would chain code an image in about the same amount of time it took to scan the image was developed by Shimotsuji *et al*at Toshiba [Shim:88]. This hardware clearly showed that high speed chain coding was possible.

7.0 Vector Reduction

Vector reduction is the process of reducing the chain code to meaningful geometric figures, such as lines and circles. Several systems are described by Bixler *et al* [Bixl:88] and Jain [Jain:89].

An excellent, simple algorithm is described by Jain:

> *Algorithm.* Approximate the curve by the line segment joining its end points (A,B). If the distance from the farthest curve point (C) to the segment is greater than a predetermined quantity, join AC and BC. Repeat the procedure for new segments AC and BC, and continue until the desired accuracy is reached [Jain:89, p. 364].

This quick algorithm guarantees that the lines approximate the chain code within a specifiable error. It seems to produce very close to the minimum possible number of line segments needed to approximate the chain code. It is much like the one Lowe uses [Lowe:87].

One of the first vector reduction methods is described by Ramer [Rame:72]. Other vectorization methods have been developed, but none produces results any better than Jain's. Leu and Chen describe a method that approximates lines to chain codes with a maximum error criterion [Leu:88]. Rosin and West make useful suggestions about reduction to lines [Rosi:89]. Linear programming was used to solve this problem in an "optimal" way by Montanari [Mont:70]. A method based on the "optimal" definition is given by Sklansky and Gonzalez [Skla:80], but it does not work as well as Jain's.

Work has been done on reducing the chain codes to geometric objects rather than lines. Pavlidis [Pavl:82b, p. 230–288] describes methods that reduce the chain code to lines, arcs, or splines, rather than polygons. Reduction to conic segments is addressed by Albano [Alba:74]. Reduction to digital circles is described by Nakamura and Aizawa [Naka:84]. Landau gives some methods for approximating an arc from a set of points [Land:87], and Thomas and Chan give a faster method [Thom:89]. Finally, Jain [Jain:89] and Bixler *et al* [Bixl:88] provide methods for reducing the chain code to splines.

A final stage, closing gaps introduced by noise in the image and error in the previous stages, has been developed by Scher *et al* [Sche:82] and Princen *et al* [Prin:90]. Higher level processing using syntactic methods is the subject of research by Joseph and Pridmore [Jose:92].

8.0 Other Vectorization Work

Most of the problems with these system can be blamed on artifacts created in the thinning process. Several researchers have developed methods for solving this problem, with mixed success. A very useful survey paper for vectorization is by Nadler [Nadl:84].

One often proposed vectorization method that needs to be eliminated from consideration right away is to move a large square across the black section of the image and to follow the centre of the square. Wakayama [Waka:82] and Shih and Kasturi [Shih:89] show clearly that this technique does not work.

The artifacts caused by most current thinning algorithms have motivated a search for techniques that can recognize lines without thinning. Chen and Hsu [Chen:89] associate a direction with every pixel by finding the direction of the shape that the pixel is part of and then segmenting the image into regions containing smoothly changing directions. This paper gives the definition of an orientation map, it and shows how they are used. The technique Chen and Hsu propose does an excellent job of separating two intersecting lines into separate ones.

Methods for following the centre of lines without thinning them are presented by Black *et al* [Blac:81].

8.1 Outline Collapsing

Instead of starting by thinning the image, Intergraph's I/VEC finds its outline and then uses chain coding and vector reduction steps like Musavi's to get the vectors that represent this outline of the image [IVEC:92][IGEO:92]. At the end of this procedure, the algorithm isolates single lines. It finds long, parallel lines that constitute the edges of individual lines and then isolates a single line that runs between them. This should be the centre of the original line. A similar system is described by Boatto *et al* [Boat:92].

8.2 Grayscale Work

Watson *et al* [Wats:84], Arvind [Arvi:84], and Stevens [Stev:87] take yet another approach to the thinning problem. They devise systems based on using a grayscale scan and blurring the image with a Gaussian. The darkest region of the blurred line is the centre of the line. Other grayscale methods have been examined by Joseph [Jose:89] and Nishimura and Fujimoto [Nish:89].

8.3 Related Work

Methods based on laying a grid over an image and recording where lines cross from one grid section to another are used by Ejiri *et al* [Ejir:84] and Vaxiviere and Tombre [Vaxi:92]. Hitachi has built a scanner specifically for this purpose. This method is based on the graph–based vectorization method proposed by Pavlidis [Pavl:86]. Methods employing hexagonal grids have been used by Gibson and Lucas [Gibs:82] with notable success. Although this method looks promising, there has not yet been much research into it.

The idea of taking a sketch of an engineering diagram and converting it to the engineering diagram it is meant to represent is being explored by Yoshino *et al* [Yosh:83], Jansen and Krause [Jans:84], and Donnelly and Martin [Donn:87]. Finding the dimension in engineering drawings has been pursued by Dori [Dori:89] [Dori:92].

Chapter 2

Scanning

The first stage in the conversion of a document is to obtain a cleanly scanned binary image that accurately represents it. In this chapter several aspects of scanners are considered: the transport mechanism, light sources, camera alignment, linearity, focus, pixel resolution, point spread function and speed.

A wide variety of scanning devices is available, and their various noise characteristics affect the quality of any image that is eventually vectorized. The majority of scanners today are based on CCD camera technology that scans one line at a time. A discussion of the different problems that arise in the scanning process is followed by a comparison of several scanners and their effects on vectorization. This discussion concentrates completely on gray scale scanners and ignores colour scanners because most engineering drawings have only one colour.

Currently most scanners for paper documents have either a flatbed or a roller design. With a flatbed scanner, the document sits on a large piece of glass above the lens of the camera. The paper remains stationary while the lens moves under the glass to scan the document. The lens moves only in the Y direction, and the CCD camera captures a complete scan line of information in the X direction for each Y position. The light source usually moves with the camera, resulting in fairly even lighting conditions; however, vibrations generated as the camera moves may create problems. A variation of this scanning setup is to have a moving mirror that reflects the light into a camera that remains in a fixed spot. Flatbed scanners are therefore much like common photocopiers in their construction. One problem with such scanners is that a scanner that can deal with E–size (26x48 inch) drawings is very large and hard to transport.

The other common scanner design, the roller scanner, features a camera mounted in a fixed position and a pinch roller that moves the paper across one or more lenses. Fax machines are set up in much the same way. One problem with this design is that occasionally the paper in the roller slips and causes severe distortions in the scan. Distortions can also occur as the paper slowly skews as it feeds through the scanner: a straight line on the paper gets scanned as a curve. Often several cameras are used in a wide roller scanner – the ANA Tech Eagle 4080ET scanner has seven [ANAT:92]. In most scanners there is one light source for each camera.

Two other configurations for scanners are drum scanners and vacuum scanners. In a drum scanner, the document is attached to a drum that is rotated past a single point optical sensor in a way analogous to how a lathe would turn a drum. Because drum scanners contain single point optic detectors, they are very slow and are no longer in common use.

A type of scanner whose popularity is increasing is the vacuum scanner. It is much like the roller scanner, except that instead of pinching the document between rollers, its vacuum mechanism sucks the document down against moving belts. Because of the large area of contact between the the belt and the document, a document in a vacuum scanner can be moved much faster without slipping than can one in a roller scanner. Speeds in the range of 18 inches per second are possible.

To get consistent scans, a stable, even, bright light source is required. Alternating current (AC) fluorescent and direct current (DC) halogen lights are most commonly used. The spectra of both of these kinds of lights have narrow peaks that cause strange effects when coloured documents are scanned. AC lights also flicker slightly at the AC frequency of 60 Hz. If a document is scanned quickly, the value calculated by the CCD sensor does not have a chance to become integrated over several cycles, and 60 Hz aliasing noise can appear on the image. Some scanners give excellent results using fluorescent AC light sources with a high frequency AC source, so that the CCD can integrate the results of the scans over several cycles for each pixel.

The alternative to AC fluorescent is DC halogen lights, used in many high end scanners. They tend to be point light sources that illuminate the document unevenly. Often, to remedy this problem, calibration documents are scanned and corrective tables implemented in software. The ANA Tech Eagle 4080ET and the Mekel scanners use this technique [ANAT:92], [Meke:92]. Another problem that arises with DC halogen lights is that they get very hot and can cause a document to stretch as it is scanned. Many high quality maps are produced on mylar film that is not as prone to stretching when heated.

Dirty light sources, whether AC or DC, are almost as much of a problem as dirty lenses. A dirty light source must be carefully cleaned and the whole system recalibrated.

Wide, high resolution roller scanners like the ANA Tech Eagle 4080ET have several cameras. This feature makes possible the scanning of wide documents without the problems introduced by moving cameras, but it complicates the translation of the camera images into one large raster image. The assembly of images is made easier by arranging the cameras so that their fields of view overlap slightly and configuring the scanner such that its software selects which camera each pixel in the scan comes from.

A square that is scanned on one part of the scanner and then scanned again elsewhere may not come out the same size twice – it may even be contorted into a rectangle, trapezoid, or worse. As well, slight stretching of the image is not uncommon. This usually happens because either the optics are poor and distort the image or the actual CCD array is not mounted squarely in relation to the image. If the CCD array is twisted relative to the image, the scanned image becomes skewed.

Current CCD technology projects as much as eight inches of paper onto strips less than $1/8$ of an inch wide that contain 3000 to 5000 CCD elements. The high level of magnification and short focal length implied in this setup result in a very small depth of field. On the ANA Tech Eagle 4080ET scanner, a change in focal length of $2/1000$ of an inch can blur the image noticeably. The focus is markedly affected if a transparent piece of mylar film with data drawn on one side of it is scanned upside down instead of right side up. Because of the sensitivity of such scanners, the focus must always be sharp and refocusing is difficult.

1.0 Information Per Pixel

Most grayscale scanners provide eight bits of data per pixel but fail to provide eight bits of information. The number of bits of information being taken from a paper scan can be calculated by scanning a page that is half white and half black. V_b and V_w will represent the average values of black and white pixels, respectively. The maximum deviation of a black pixel value from this average will be called N_b, and the corresponding value for a white pixel will be N_w. Assuming that the amount of noise is linear in gray level, the noise at the gray pixel x becomes

$$n(x) = N_b + \frac{x - V_b}{V_w - V_b}(N_w - N_b). \tag{2.1}$$

Since the number of possible distinct gray level values in a range $[x_0, x_1]$ is the size of the range divided by the amount of noise in the range, the number of different gray values that could possibly come off a scanner is

$$\int_{V_b}^{V_w} \frac{dx}{n(x)}. \tag{2.2}$$

Every possible gray value is assumed to be equally likely to appear, so the number of bits of information provided by the scanner about a pixel is

$$I = \log_2 \int_{V_b}^{V_w} \frac{dx}{n(x)}, \tag{2.3}$$

which can be written as

$$I = \left(\frac{\ln \left(-\frac{(\ln(V_b - V_w) - \ln(N_w\,V_b - N_w\,V_w))(V_b - V_w)V_b}{N_b - N_w} \qquad -\frac{(\ln(N_b\,V_b - N_b\,V_w) - \ln(V_b - V_w))(V_b - V_w)V_b}{N_b - N_w} \right)}{\ln(2)} \right). \tag{2.4}$$

The equation provides a simple method for approximating how many bits of useful information each pixel contains by estimating the amount of noise in each pixel.

2.0 Point Spread Evaluation

The point spread function (PSF) provides a good quantitative way to describe how blurry a given scanner is. It also provides a way to correct the blurring as shown in Jain [Jain:89] and many other image processing texts.

Mathematically modelling the process of scanning requires first that the optical density of a point on the paper being scanned be represented by the function $f(x, y)$. The scanning process distorts this function f and provides a raster scan represented by the function $g(x, y)$. Added to the image are the distortion caused by the imaging system, designated by the function H, and some noise, $\eta(x, y)$. This set of equations is represented by the block diagram in figure 2.1.

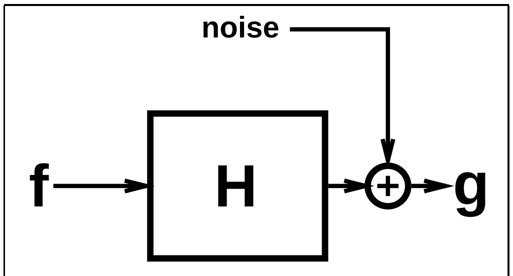


Fig 2.1: Scanning distortion model

This is a common image degradation model (Gonzalez and Wintz describe it well [Gonz:87]) that is described mathematically as

$$g(x, y) = Hf(x, y) + \eta(x, y) \tag{2.5}$$

where H is an undetermined function operator that represents distortions caused by the imaging system. So H operates on a function that describes one image and returns a different function that describes the distorted image.

Given that the image is captured using a CCD camera in controlled lighting conditions, it can be assumed that

$$\eta(x, y) = 0 \tag{2.6}$$

and that the distortion is linear, so that for any functions f_1 and f_2 it holds that

$$H[k_1 f_1(x, y) + k_2 f_2(x, y)] = k_1 H f_1(x, y) + k_2 H f_2(x, y). \tag{2.7}$$

Assuming as well that the distortion is the same in all areas of the image, H is shift invariant. This means that given

$$g(x,y) = Hf(x,y) \tag{2.8}$$

then

$$Hf(x-\alpha, y-\beta) = g(x-\alpha, y-\beta). \tag{2.9}$$

That this distortion is the same in all areas of the image may not always be a fair assumption, but it is generally reasonably accurate.

From the equations and assumptions described above, a method like Gonzalez and Wintz's [Gonz:87, p. 207] for approximating H for a given scanner can be derived. Express $f(x,y)$ in the form

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)\delta(x-\alpha, y-\beta)d\alpha d\beta \tag{2.10}$$

where $\delta(x,y)$ is the two dimensional Dirac delta function. Therefore

$$g(x,y) = H\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)\delta(x-\alpha, y-\beta)d\alpha d\beta. \tag{2.11}$$

Assuming that the additive property is valid for this integral (the proof that this is the case is given by Niemann [Niem:90, p. 53]), we find that

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} Hf(\alpha,\beta)\delta(x-\alpha, y-\beta)d\alpha d\beta, \tag{2.12}$$

but with the linearity property of H and the fact that $f(\alpha,\beta)$ is shift invariant, it follows that

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)H\delta(x-\alpha, y-\beta)d\alpha d\beta. \tag{2.13}$$

Now define $h(x,y)$ to be the impulse response of H, so that

$$h(x,y) = H\delta(x,y). \tag{2.14}$$

But since H is shift invariant, meaning that

$$h(x-\alpha, y-\beta) = H\delta(x-\alpha, y-\beta), \tag{2.15}$$

and

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)h(x-\alpha, y-\beta)d\alpha d\beta, \tag{2.16}$$

g is the convolution of f and h. Here h is the point spread function (PSF). It determines how much the original f is smeared when the image g is produced: the larger the PSF, the greater the smearing.

Given the above definition of the PSF, an approximate measure of the PSF for a given scanner can be made from a suitable test image. A few assumptions about the PSF are made:

- The PSF is symmetrical

  This assumption only means that the distortion would remain unchanged if the image being scanned were mirrored right to left. This assumption is reasonable because the construction of scanners is symmetrical; there is no reason for a right to left scan to differ from a left to right one.

- The volume of the PSF is 1, so that

  $$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x)dxdy = 1. \tag{2.17}$$

  This assumption ensures that the grayscale value of a constant image is not changed by the distortion except near the edges. It really just amounts to having a gray scaling factor.

The following work derives a method for approximating the PSF from the values of scanning a particular test image. First it is derived for a continuous space. Then an analogous derivation is done for a discrete space.

Consider a test image whose left half is completely black and whose right half is completely white, and consider also that where the regions meet there is a sharp transition from black to white. The gray values of the scanned test image are scaled so that black is -1, white is 1, and the line occurs at $x = 0$. Because the image is vertically uniform, the convolution in the vertical direction is irrelevant. The problem is reduced to a one dimensional one.

The image distortion model is

$$g(x) = \int_{-\infty}^{\infty} f(\alpha)h(\alpha - x)d\alpha, \tag{2.18}$$

where $g(x)$ is the values scanned from the test image. From the construction of the test image,

$$f(x) = \left\{ \begin{array}{ccc} 1 & ; & x > 0 \\ -1 & ; & x < 0 \end{array} \right\}. \tag{2.19}$$

Now $g(x)$ can be rewritten as

$$g(x) = \int_{-\infty}^{0} f(\alpha)h(\alpha - x)d\alpha + \int_{0}^{\infty} f(\alpha)h(\alpha - x)d\alpha. \tag{2.20}$$

By the definition of $f(x)$ it becomes

$$g(x) = -\int_{-\infty}^{0} h(\alpha - x)d\alpha + \int_{0}^{\infty} f(\alpha)h(\alpha - x)d\alpha. \tag{2.21}$$

Using a change of variable to shift by x we get

$$g(x) = -\int_{-\infty}^{-x} h(\alpha)d\alpha + \int_{-x}^{\infty} f(\alpha)h(\alpha)d\alpha. \tag{2.22}$$

Since h is symmetrical (i.e. since $h(x) = h(-x)$),

$$g(x) = -\int_{\infty}^{x} h(\alpha)d\alpha + \int_{-x}^{\infty} f(\alpha)h(\alpha)d\alpha. \tag{2.23}$$

This can be rewritten as

$$g(x) = \int_{-x}^{x} h(\alpha)d\alpha, \tag{2.24}$$

and, as h is symmetrical,

$$g(x) = 2\int_{0}^{x} h(\alpha)d\alpha. \tag{2.25}$$

So, by the fundamental theorem of calculus

$$h(x) = \frac{d}{dx}\frac{g(x)}{2}. \tag{2.26}$$

This last equation provides a method for approximating the PSF from the scan of the test document.

An analogous argument can be made for discrete spaces, in which g, f, and h are raster images instead of continuous functions. The distortion model becomes

$$g(x) = \sum_{i=-\infty}^{\infty} h(i-x)f(i). \tag{2.27}$$

These lines are rewritten using the definition of f, yielding

$$g(x) = -\sum_{i=-\infty}^{0} h(i-x) + \sum_{i=0}^{\infty} h(i-x). \tag{2.28}$$

This is shifting to get that

$$g(x) = -\sum_{i=-\infty}^{-x} h(i) + \sum_{i=-x}^{\infty} h(i). \tag{2.29}$$

Since h is symmetrical it holds that

$$g(x) = -\sum_{i=x}^{\infty} h(i) + \sum_{i=-x}^{\infty} h(i), \tag{2.30}$$

thus

$$g(x) = \sum_{i=-x}^{x} h(i). \tag{2.31}$$

Since h is symmetrical,

$$g(x) = 2\sum_{i=0}^{x} h(i). \tag{2.32}$$

The difference between two consecutive points can be written as

$$g(x+1) - g(x) = 2(\sum_{i=0}^{x+1} h(i) - \sum_{i=0}^{x} h(i)). \tag{2.33}$$

Combining the ranges of the summations gives

$$g(x+1) - g(x) = 2(\sum_{i=x+1}^{x+1} h(i)), \tag{2.34}$$

which simplifies to

$$g(x+1) - g(x) = 2h(x+1). \tag{2.35}$$

This is rewritten as

$$\frac{g(x+1) - g(x)}{2} = h(x+1), \tag{2.36}$$

which simplifies to

$$h(x) = \frac{g(x) - g(x-1)}{2}. \tag{2.37}$$

This final equation provides a way to approximate h from the scan data g.

2.1 Measurements

To estimate the PSF of a scanner, first a large black and white image was photographed onto high contrast 35mm film to make the test image. The edge of this film was examined under a microscope at a magnification of 30X and found to be very sharp. Because of the film's high contrast, the black and white regions had uniform consistencies.

This test image was scanned on two scanners, and the pixel values near the edge were recorded. The image was then rotated 90^o and the scanning performed again to get an approximation for h in the other axis.

2.2 Results

Table 2.1 describes the pixel values near the edge of the scan on one scanner. From these values, the values of the PSF function for positive x were calculated and recorded in table 2.2.

Scanner	Pixel Values										
Microtek 300zs	4	5	5	15	42	53	59	60	61	61	62

Table 2.1: Edge values from scanner

Using the scan data as $g(x)$ and equation 2.37 values for $h(x)$ are computed and shown in table 2.2.

The plot of the point spread function data in table 2.2 is shown in figure 2.2. The standard deviation of this PSF is 1.37 pixels or $5/1000$ of an inch.

Scanner	PSF Function Values						
Microtek 300zs	27	11	6	1	1	0	0

Table 2.2: Point spread function

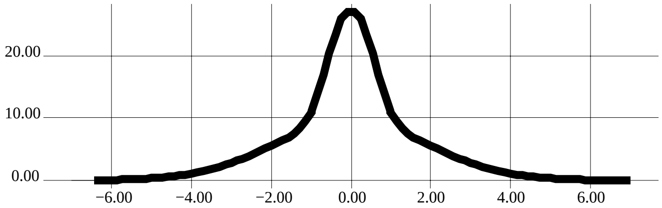


Figure 2.2: Point spread function

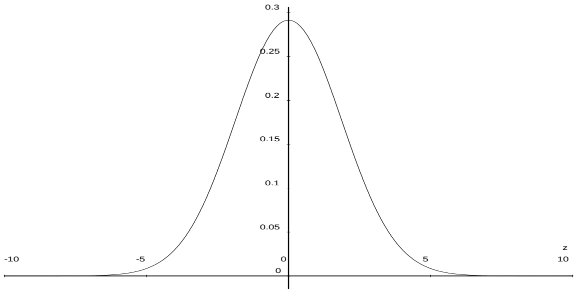


Figure 2.3: Gaussian approximation of PSF

The Gaussian that approximates this curve (same mean and standard deviation) is shown in figure 2.3. From this graph it becomes apparent that the PSF for this scanner is not terribly close to a Gaussian but has a shape reminiscent of one. Gaussians are often used to approximate the PSF function of a scanner, but a Moffet function may be a better model for the PSF. The Moffet function is

$$I(x) = I_0 \left(1 + \left(\frac{x}{\rho} \right)^2 \right)^{-\beta} . \tag{2.38}$$

The parameters I_0, ρ and β give more degrees of freedom than a Gaussian so that the model can more accurately describe the measured PSF data.

If the PSF is narrow (has a small standard deviation), an image will not be blurred as much by the scanner.

3.0 Speed

A scanner's speed in scanning a file and saving it to disk is most often lost at a bottleneck either as the data is transferred from the scanner to the workstation controlling it or as the workstation transfers the data to disk. A useful measure of speed is the bits of useful information acquired per second.

This measurement is defined as the area scanned (in square inches), divided by the variance of the point spread function (also in square inches), multiplied by the number of useful bits of information obtained per pixel per unit of time. This measurement of speed makes it easier to compare scanners that produce data of different qualities.

4.0 Comparison of Scanners

The information in table 2.3 provides a way to evaluate the probable effectiveness of these scanners for different purposes. The scanners are described in terms of PSF, pixel resolution, speed, maximum document size, transport type, light source type, and price. A scanner for engineering diagrams must be able to handle E size drawings and have a PSF of less than 10/1000 of an inch, so that lines close together can be easily distinguished. Such a scanner must also have information per pixel greater than four bits for a very clean document and greater than six bits for blueprints or dirty paper.

In table 2.3, Size is the largest size of document that can be scanned, in inches. The ANA Tech scanner has an 81 inch length limit, even though it is a roller scanner, because it has a 65534 scan line limit. The units for PSF are 1/1000 of an inch. Pixel resolution is given in bits. Speed is in kbps (kilobits per second) of information. The speed given for the ANA Tech scanner was not measured but was taken from the manufacturer's specifications. Prices are constantly dropping. The prices quoted are the approximate list price in the summer of 1992; actual prices are likely lower.

Scanner	Microtek 300ZS	Ana Tech 4080
PSF	5.00	1.26
Pixel Res	6.13	6.43
Speed	160	2750
Size	8.5x14	40x81
Transport	Flatbed	Roller
Light	Neon	Halogen
Price	$1900	$45,100

Table 2.3: Comparison of two scanners

Chapter 3

Preprocessing

Preprocessing describes steps that can be taken to clean up the scan after it comes off the scanner but before it is passed to the vectorization software.

1.0 Thresholding

The thresholding step converts the grayscale image into a black and white one. The simplest thresholding method simply picks a gray level and calls every pixel darker than this level black and every lighter pixel white. This procedure works well for many drawings, and most scanners can do it through their hardware. Another thresholding technique, in which the image is dithered to yield a grayscale–like appearance, are unsuitable when vectorization is the goal. This is because gray lines get turned into black and white dots, which defeats the whole purpose of thresholding as far as vectorization is concerned: thresholding must clean up the image and get rid of as many unaccountable dots as possible.

Excellent coverage of techniques for thresholding line drawing type documents such as engineering diagrams are provided by Taxt *et al*, Sahoo *et al*, and Fu and Mui. Taxt *et al*describe several algorithms that produce excellent results but require interactive input [Taxt:89]. Fu and Mui [Fu:81] take a very theoretical statistical approach to the problem, while Sahoo *et al* [Saho:88] take a statistical approach to derive very practical algorithms.

1.1 Histogram Evaluation

One of the simplest methods for thresholding an image interactively is to have the user look at the histogram of its pixel values to pick a fixed thresholding level for the whole image. Histograms are traditionally displayed on linear graphs, but histograms of line drawings are better displayed on log–linear graphs. The reason is that line drawings are usually predominantly white with a small percentage of black pixels that form the lines. Their histograms are bimodal and skewed so heavily that they can barely be seen on linear–linear graphs.

The histogram for a typical engineering diagram is shown in figure 3.1.

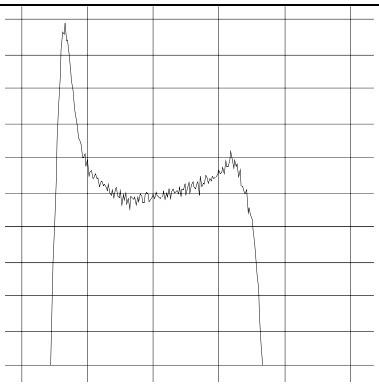


Fig 3.1: Histogram of engineering diagram

A good thresholding value for the scan of this diagram would be the value at the bottom of the valley between the two peaks – about 75 for this image. The histogram evaluation method is an interactive but simple way to select thresholds for an image. It works well for scans with sharp lines taken off clean paper.

The major problem with the histogram evaluation thresholding method is that it thresholds the whole image with the same value, which is fine as long as the contrast, the cleanness of the background, and so on are uniform throughout the image. If, however, the image has a region in which the lines are very faint as well as a large coffee stain that darkens the background in a certain area, histogram evaluation thresholding becomes inadequate. If the lines in the faint section are not as dark as the background in the stained section, it will be impossible to find a single value that will result in a good thresholding. The stained section requires a thresholding value different from the one used in the light section. Using different thresholding values for different parts of the image is called adaptive thresholding.

1.2 Adaptive Thresholding

Adaptive methods generally build a local histogram based on nearby pixels for each individual pixel and then threshold the pixel. Such adaptive methods can be flexible and sophisticated enough to incorporate other information known about the diagram into their calculation of the thresholding level. For example, the fact that engineering drawings are mostly white with a few black lines but no large black areas can easily be incorporated into these adaptive schemes and used as *a posteriori* probabilities for constructing Bayesian classifiers to separate black from white. Taxt *et al*present some of these adaptive thresholding methods [Taxt:89]. They provide an excellent way to use knowledge about classes of images to assist in thresholding.

A method based on edge detection and modelling the illumination in the image is presented by Parker, Jennings, and Salkauskas [Park:93]. First, a Shen–Castan edge detector [Shen:92] is run over the image to find edge pixels. These edge pixels represent a desired threshold for the part of the image near each edge pixel. All the edge pixels are used as data points to fit a surface across the image that represents the thresholding value at each location. The surface is fit using a moving least squares method. The surface ends up modelling the lighting gradient across the image, and the image is finally thresholded by looking to see if the value of any given pixel is above or below the illumination surface at that location. Although good results can often be obtained from poor quality input, computation times are very long. Parker's algorithm [Park:91] describes a method that computes faster and provides thresholding that is not quite as good.

2.0 Noise Removal

Images scanned by CCD scanners are often noisy, having little black and white speckles that make an image look as if salt and pepper had been sprinkled on it. This sort of noise is often seen on photocopies made by older copiers. The cause of this phenomenon, called blooming, has yet to be convincingly set forth. The possibility that blooming noise is thresholded Poisson noise from the CCD does not quite fit the data. One quantum related theory has it that blooming occurs because although the CCD sensors are usually accurate they are occasionally way off, causing the occasional pixel value in a scan to differ significantly from the pixel values around it. The other theory is that when monochromatic light is scattered from a surface whose roughness is of the order of the wavelength of the light, interference among the light waves produces nodal patterns that leave bright and dark spots here and there. Jain presents this explanation [Jain:89, p. 313].

Some of the most sophisticated algorithms for speckle removal have been developed for Synthetic Aperture Radar (SAR). Complex methods for speckle removal are presented by Jain and Christensen [Jain:80], Niemann [Niem:90], and Lim and Nawab [Lim:80]. A simple method for speckle removal that I developed involves a modification of a common flood fill algorithm [Roge:85, p. 88], [Fole:82, p. 450]. It may be summarized as follows:

1. Start with any white pixel;
2. Set this pixel to black;
3. Recursively set all this pixel's white neighbouring pixels to black.

The algorithm proceeds until it either runs out of white neighbours, in which case it proceeds to the next white speckle to be removed, or until the region exceeds the user-specified maximum area for speckles – i.e., until the region is determined to be too big to be noise. When this happens, the algorithm backtracks, converting the pixels back to white. This algorithm removes small white specks (salt noise) but an inverse algorithm that removes black specks (pepper noise) can be made by swapping white and black in the algorithm. Speckle removal methods that use up less stack space exist, but since these areas are small, the amount of stack space used is not usually of consequence.

The image in figure 3.2 is the thresholded image. It contains small black areas. In figure 3.3 these small black areas have been removed by the speckle

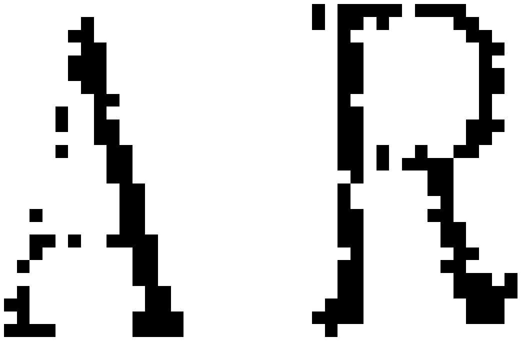


Figure 3.2: Image with speckle noise

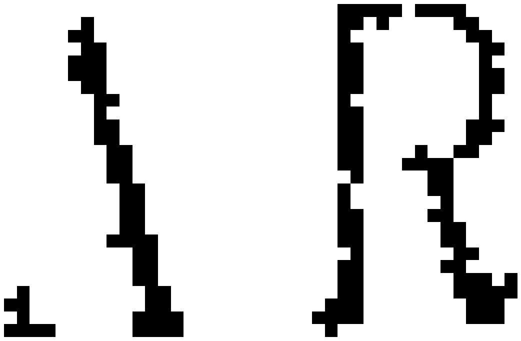


Figure 3.3: Image after speckle noise removed

filter.

Chapter 4

Erosion Vectorization

Many current vectorization systems utilize erosion methods (such systems are described by Bixler and Sanford [Bixl:85], Bhaskaran and Flandrena [Bhas:89], Jennings and Flanagan [Jenn:93], Johnson and Bird [John:90], Hoshino *et al* [Hosh:86], Kikkawa *et al* [Kikk:84], Lee *et al* [Lee:90], Musavi *et al* [Musa:88], and Suzuki and Yamada [Suzu:90]). These systems work by thinning the image until all the lines are one pixel wide and then chain coding the thinned image, taking the chain codes to represent the centres of the lines. The chain codes are then reduced to straight line segments. This chapter describes the implementation of this type of vectorization method.

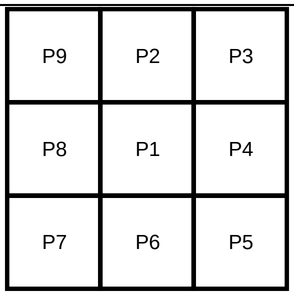
1.0 Thinning

Many different thinning algorithms exist. Thinning takes a raster scan with lines several pixels wide and produces a skeleton in which the same lines are only one pixel wide. Some of the more innovative or unusual thinning methods are described by Arcelli and Baja [Arce:85]; Baruch [Baru:88]; Govindan and Shivaprasad [Govi:87]; Hilditch [Hild:69]; Jiminez and Navalon [Jimi:82]; Kwok [Kwok:88]; Li and Suen [Li:91]; Montanari [Mont:69]; Naccache and Shinghal [Nacc:84]; O'Gorman [OGor:90]; Pavel [Pave:79]; Pavlidis [Pavl:82b], [Pavl:82]; Sinha [Sinh:87]; Udupa and Murthy [Udup:75]; Wakayama [Waka:82]; Yu and Tsai [Yu:90]; and Zhang and Suen [Zhan:84]. Kwok [Kwok:88] and Smith [Smit:87] present surveys of thinning methods. Parker and Jennings give a quantitative method for evaluating thinning algorithms and a definition of *skeleton* [Park:92]. A thinning method by Jennings *et al*provides a skeleton with subpixel level accuracy. The vectors produced by this method have greater precision than vectors produced from a skeleton thinned to the width of one pixel and the skeletons are less affected by noise [Jenn:93a].

The thinned products of the various methods differ considerably, and each method has a significant impact on the quality of the final vectorization. This section describes one of the most commonly used thinning techniques and some of the problems that arise with them.

1.1 Implementation

Zhang and Suen's [Zhan:84] thinning method is used here, because it is more commonly used than any other method in raster–to–vector conversion systems and because its strengths and weaknesses tend to characterize most thinning algorithms. Gonzalez and Wintz provide an excellent overview of this thinning technique – generally referred to as an erosion algorithm – and supply helpful details about implementation [Gonz:87, pp. 398–404].



P9	P2	P3
P8	P1	P4
P7	P6	P5

Fig 4.1: Labelling of neighbours

This erosion algorithm for thinning requires the definition of some terminology at the outset. The pixels that are neighbours of a particular pixel, $p1$, are labelled $p2, p3, ...p9$, as shown in figure 4.1. Each pixel is either black or white. Black pixels have a pixel value of 1; white pixels have a pixel value of 0. The goal at this stage is to erode the black regions so that all that remains is a black skeleton. Two functions of a pixel are defined: $N(p_1) = p_2 + p_3 + \cdots + p_8 + p_9$ and $S(p_1) =$ Number of 0 to 1 transitions in the sequence $p_2, p_3, \ldots, p_9, p_2$. From these two functions, two conditions are defined for a given pixel. Condition one of a pixel $C_1(p_1)$ is true only if all of the following are true:

$$2 \leq N(p_1) \leq 6, \tag{4.1}$$

$$S(p_1) = 1, \tag{4.2}$$

$$p_2 p_4 p_6 = 0, \tag{4.3}$$

and

$$p_4 p_6 p_8 = 0. \tag{4.4}$$

Condition two of a pixel $C_2(p_1)$ is true only if all of the following are true:

$$2 \leq N(p_1) \leq 6, \tag{4.5}$$

$$S(p_1) = 1, \tag{4.6}$$

$$p_2 p_4 p_8 = 0, \tag{4.7}$$

and

$$p_2 p_6 p_8 = 0. \tag{4.8}$$

The algorithm now has two steps. Step one involves checking all the pixels in the image and marking each pixel for which C_1 is true. At the end of step one all the marked pixels are deleted by having their pixel values set to 0. Step two is the same as step one, except that C_2 is checked instead of C_1. Steps one and two are repeated in order until there is an iteration in either step one or two in which no pixels are marked.

When the algorithm terminates, a thinned skeleton is all that remains of the image. The image of the letter H in figure 4.2 has been thinned using this algorithm, and the result is shown in figure 4.3.

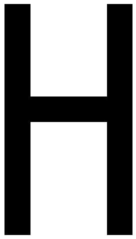


Figure 4.2: Image of an H

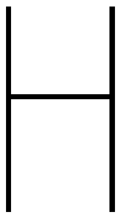


Fig 4.3: Image of a thinned H

A small problem with this thinning algorithm, like many others, is that it does not always give unitary skeletons. The number of 8–connected black neighbours a pixel has determines what kind of a pixel it is. On a unitary skeleton, one black neighbour means the pixel is an end point, two makes it part of a line, and more than two makes it an intersection. A nonunitary skeleton is one in which some points that are not intersections have $N(p_1) = 3$ (see figure 4.7). This confuses the part of the chain coding that determines when a pixel is part of a line, when it is an end point, and when it is an intersection. Abdulla *et al*propose an algorithm to make the image unitary [Abdu:88]. It is run as the final stage of thinning, after the Zhang and Suen algorithm. Pavlidis supplies a similar algorithm to solve this problem [Pavl:82b, p. 210–212] that does not work quite as well but does not require an extra stage of processing.

An image more complex than the letter H appears in figure 4.4. The thinned version of it is shown in figure 4.5. To make a comparison of the images easier, they are overlaid in figure 4.6.

A small section along one of the thinned lines is magnified in figure 4.7 to show that several of the pixels along the centre line are not unitary. Although they are connected to three other pixels they are not intersection points.

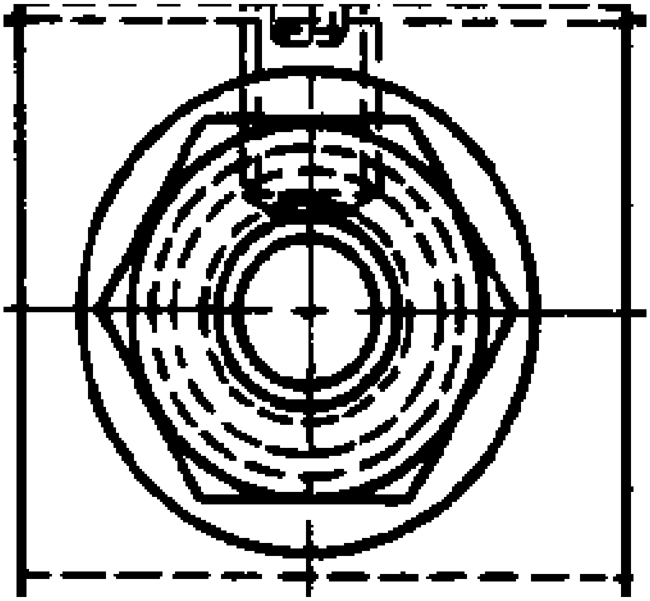


Figure 4.4: Image to be thinned

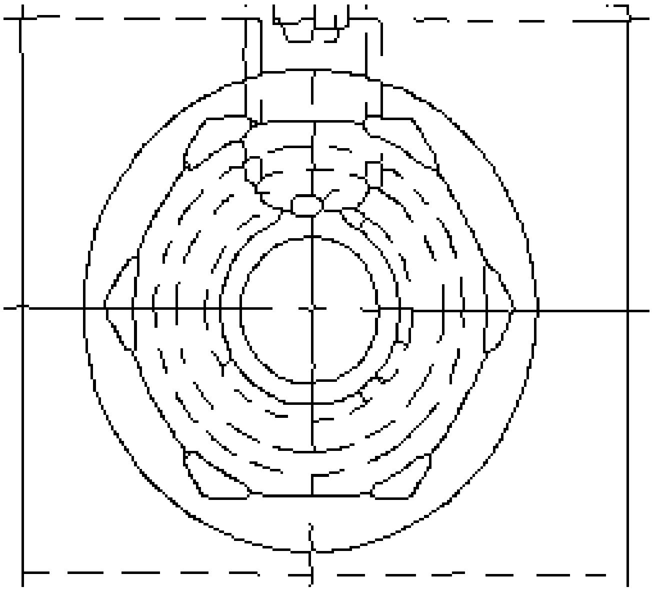


Figure 4.5: Thinned image

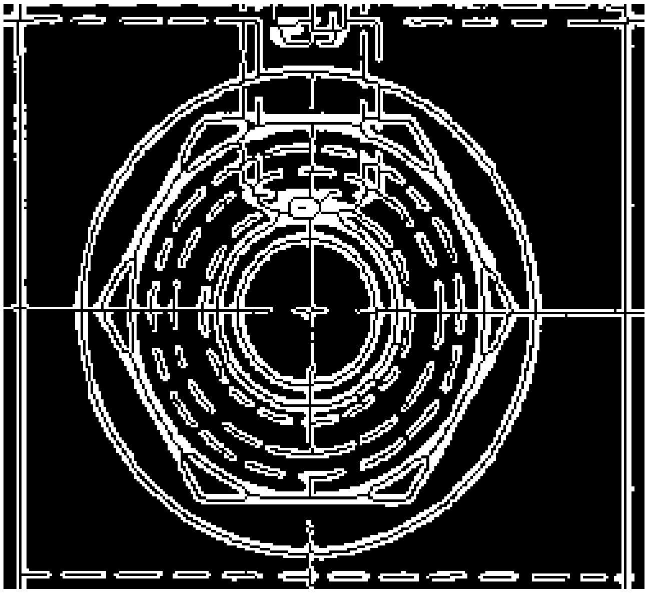


Figure 4.6: Original with thinned image overlaid

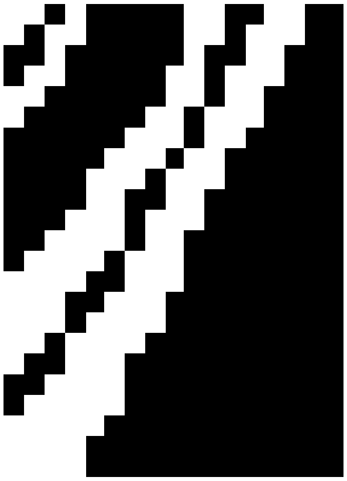


Figure 4.7: Some nonunitary pixels

2.0 Chain Coding

Once a line has been thinned to a width of one pixel it can be chain coded. The next step in vectorization, chain coding is the process of tracing a line one pixel wide by starting at an end pixel, finding the next pixel from directional information, and continuing in this way until the last pixel in the line is found. Representing a line by chain coding was first described by Freeman [Free:61] [Free:74] and later by Freeman and Davis [Free:77]. Others, including Gonzalez and Wintz [Gonz:87, pp. 392–398], have since expanded upon this idea.

As mentioned in section 4, a black pixel has a pixel value of 1, while a white pixel has a value of 0 and the $N(p_1)$ is defined as

$$N(p_1) = p_2 + p_3 + \cdots + p_8 + p_9 \tag{4.9}$$

where p_1, p_2, and so on are either 1 or 0. A pixel in a line has $N(p_1) = 2$ (i.e., it has two neighbours). Pixels that represent end or intersection points have $N(p_1) \neq 2$. If $N(p_1) < 2$ the pixel is an intersection point, and if $N(p_1) = 1$ it is an end point. If $N(p_1) = 0$, it is an isolated bit of noise that should be deleted.

Chain coding first finds an end point (where $N(p_1) = 1$) and then finds the black pixel next to the end. The algorithm uses a single number to describe the direction in which it moved. The eight possible directions are specified using the numbers shown in figure 4.8. The string of these numbers that describes the line is called the chain code.

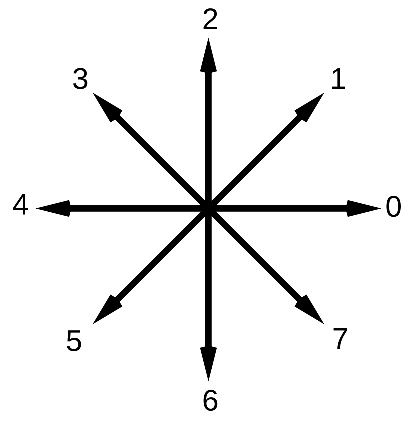


Fig 4.8: Chain code directions

When the algorithm finds an end point or an intersection for this line, the chain code for the line is output and all the pixels that are part of it are deleted. The chain coding process is repeated until all the black pixels on the image are deleted. A problem with this system is that it never chain codes figures, like the letter O, that have been thinned down to one contour and have neither a starting point where the chain coding can begin nor an intersection or an end point where the system can stop. A solution to this problem lies in first chain coding the whole image in the steps described above and then doing an extra pass over the whole image and finding any black pixels that have not yet been deleted, which are assumed to be part of closed loops. Chain coding can begin anywhere on the loop.

An image that has been eroded and then chain coded is shown in figure 4.9. The chain codes for this image are:

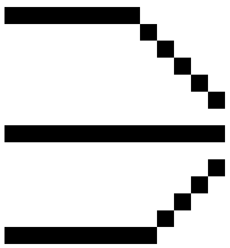


Figure 4.9: Simple image

```
2 2 000000011111
2 9 000000000000
14 11 555544444444
```

3.0 Chain Reduction

This stage seeks to identify vectors in the chain codes. It is the final step in the erosion vectorization method of raster–to–vector conversion. In it, the chain codes are examined and long vectors that closely represent the chain codes are formed. Rosin and West describe this part of the erosion vectorization technique [Rosi:89]. Algorithms almost identical to the one described here are used by Lowe [Lowe:87] and Jain [Jain:89].

The algorithm presented here (Jain's) allows the user to specify the maximum permissible deviation of the vectors from the chain codes they represent. It provides results that use of close to the minimum possible number of vectors.

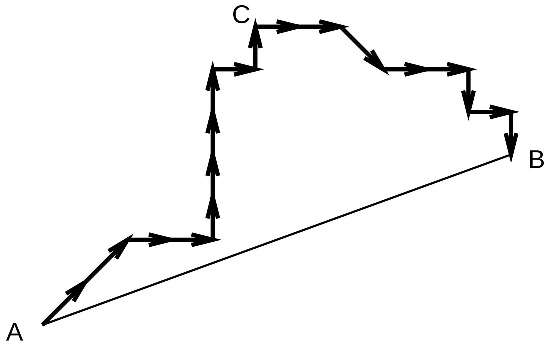


Fig 4.10: Vector approximation

This algorithm first takes a chain code and approximates it as a single line running between the two end points. This line is designated AB in figure 4.10. The point on the chain code farthest from line AB is found and labelled C. If the distance from line AB to point C is less than the maximum deviation the user has specified, the vector is a valid approximation of the chain code. If the distance is greater than the maximum allowable deviation, the algorithm tries to approximate the chain codes from A to C with the line AC and the chain codes from C to B with the line CB. This approximation continues recursively, comparing the the maximum allowable deviation with the distance between each point and line until the distances between these points and lines are less than the largest deviation allowed. All distance measurements in this algorithm use Euclidean distances. The second approximation is shown in figure 4.11.

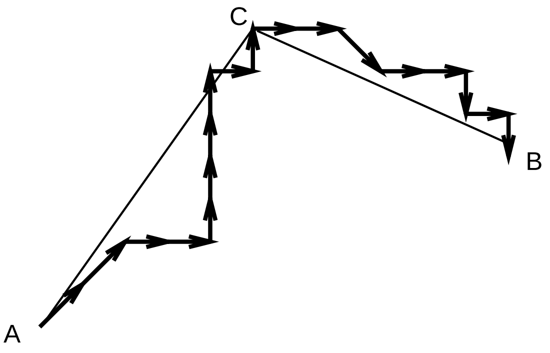


Figure 4.11: Second vector approximation

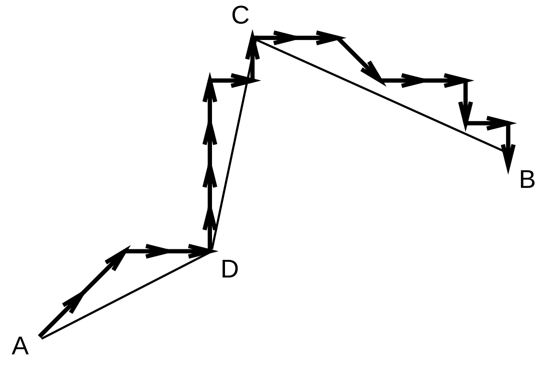


Fig 4.12: Third vector approximation

If the user had specified a maximum deviation of 2 pixels, the line CB would be close enough but AC would need to be further subdivided into the the configuration shown in figure 4.12. At this point all the vectors lie within the specified deviation, so the algorithm terminates, having identified three vectors to represent the chain code. This method was compared to the classic split and merge algorithm, which does something similar in the split step but then tries to merge adjacent segments into one segment [Grim:90, p. 105]. The split and merge produced minutely different, very marginally better results, but it took longer to run.

3.1 Distance Point to Line Segment

Since this algorithm spends most of its time computing the distance from a point to a line, it is worthwhile to optimize the speed of this calculation. One method is given by Morrison [Morr:91]. A reasonably efficient algorithm is given by Bowyer and Woodwark [Bowy:83, p. 47], but it requires that floating point numbers be used, which slows it down considerably. In this section a slightly faster integer algorithm I developed is described.

This algorithm finds the distance squared, d_s, from a point P to a line segment defined by the end points A and B (see figure 4.13).

$$V_p = A - P \tag{4.10}$$

$$V_l = B - A \tag{4.11}$$

$$l = V_l \bullet V_l \tag{4.12}$$

$$t = -V_p \bullet V_l \tag{4.13}$$

$$t_d = \min(\max(t, 0), l) \tag{4.14}$$

$$F = V_p + (t_d V_l)(\frac{1}{l}) \tag{4.15}$$

$$d_s = F \bullet F \tag{4.16}$$

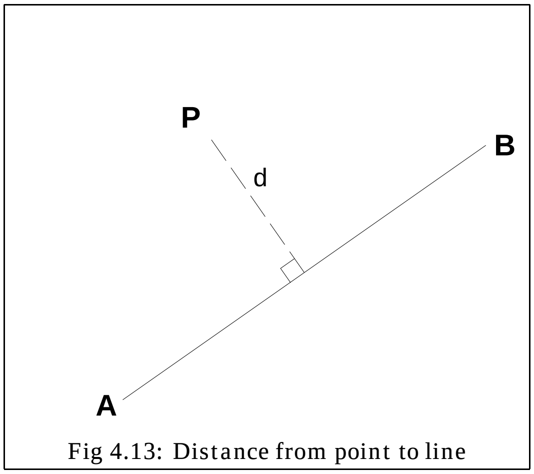


Fig 4.13: Distance from point to line

It is acceptable that this algorithm return the square of the distance (thus allowing the avoidance of the square root function), because the reason for obtaining this distance is to compare it to an error distance. An effective comparison can be made between the square of the error distance and the square of the distance between the point and the line segment. If the magnitude of V_p and V_l in the algorithm can be represented in b bits, the algorithm needs vector elements to be represented in $3b + 1$ bits. An implication of this statement is that if the points A, B and P are within about 1000 units of one another, the algorithm can run using 32 bit integers, which, on current workstations, provides a dramatic speed gain over using floating point numbers. If $V_p > 1000$ or $V_l > 1000$ the algorithm must switch to floats.

3.2 Analysis

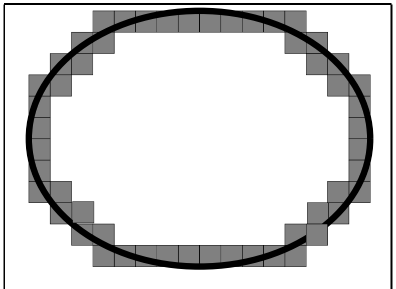


Fig 4.14: Digital ellipse

Chain reduction does introduce some error. In figure 4.14 an ellipse is turned into a digital ellipse. The gray square represents the pixels in the digital ellipse. The original ellipse is black. The digital ellipse is chain coded as shown in figure 4.15. In figure 4.16 the chain codes are represented by eight vectors. In figure 4.17 the original ellipse and the vectors that represent it are drawn together. The vectors do not coincide exactly with the ellipse. The fewer the vectors used to represent the ellipse, the greater the deviation distance.

This algorithm for reducing chain codes into vector lists is not optimal, in that it fails to reduce the chain code to the minimum number of vectors required to represent the chain code within the user determined error limit. In the optimal algorithm, most of the lines would be near their maximum error level, and the minimum number of lines would be supplied. Such a system is

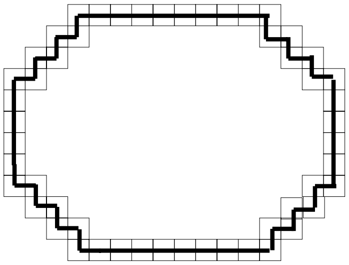


Figure 4.15: Chain coded

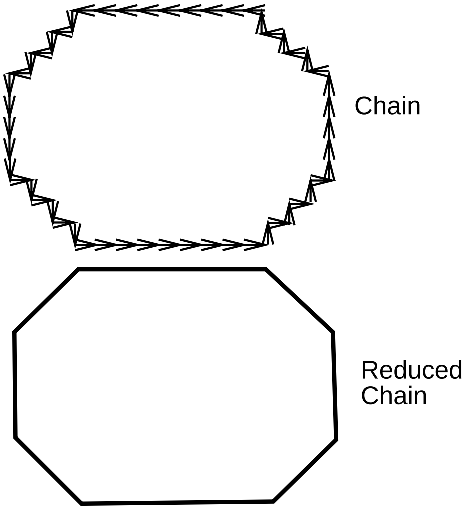


Figure 4.16: Vectors and chain code

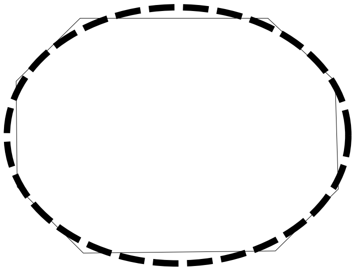


Figure 4.17: Vectors and original

presented by Wall and Danielsson [Wall:84], Sklansky and Gonzalez [Skla:80], and Montanari [Mont:70]. The optimal algorithm tends to make the vectors bend a short distance from the actual corners. The chain reduction algorithm presented here, although not optimal, tends to bend the vectors at the corners, producing nicer results. The vectors bend at corners because corners are usually the points farthest from other corners.

The majority of the computation time in a system such as this goes into the thinning stage. In the few test cases tried, this stage often took over 95 percent of the run time. Many thinning algorithms, such as Kwok's [Kwok:88], were developed to produce results similar to Zhang and Suen's but to run much faster. A different approach was taken by Molaro, Jennings, and Parker in their distributed thinning algorithm [Mola:93]. It runs on several networked UNIX workstations at once, to achieve better performance.

4.0 Results

A section of an image was taken through all the steps described in this chapter (first thinned, then chain coded, then vectorized) to produce a vectorized image. The original image appears in figure 4.18. Vectorization and reduction were done with error values of both 3 and 10 for the sake of comparison. The results are shown in figures 4.19 and 4.20.

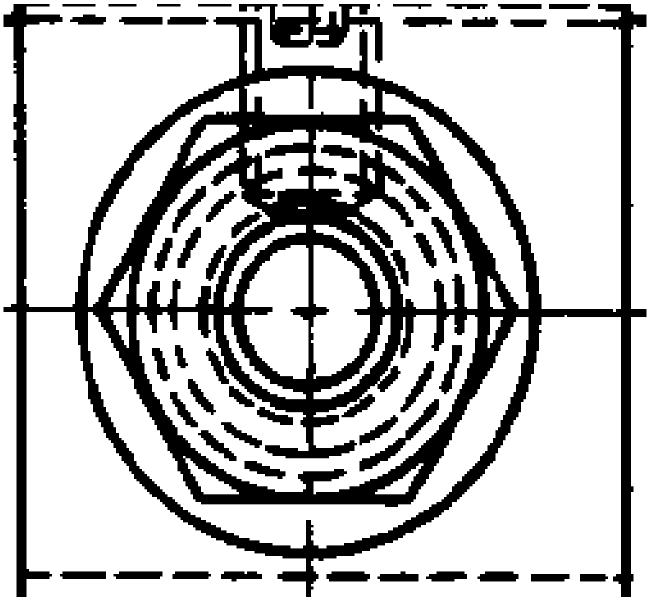


Figure 4.18: Binary image

The reduction error of 10 is too large and misrepresents the image. Much better vectorization occurs when the reduction error is 3 pixels. The long curves of the outside circle come out well. Most of the long, isolated lines emerge properly, but the lines near intersections are often messed up. The circle inside the hexagon is almost completely destroyed.

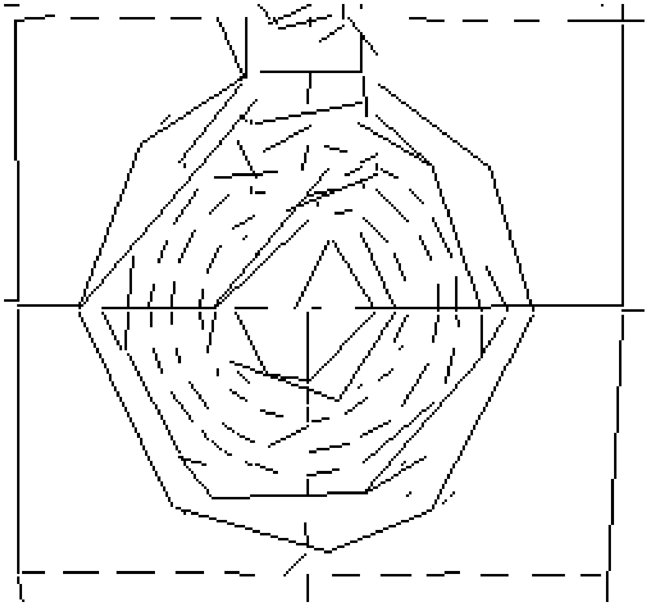


Figure 4.19: Reduction with error = 10

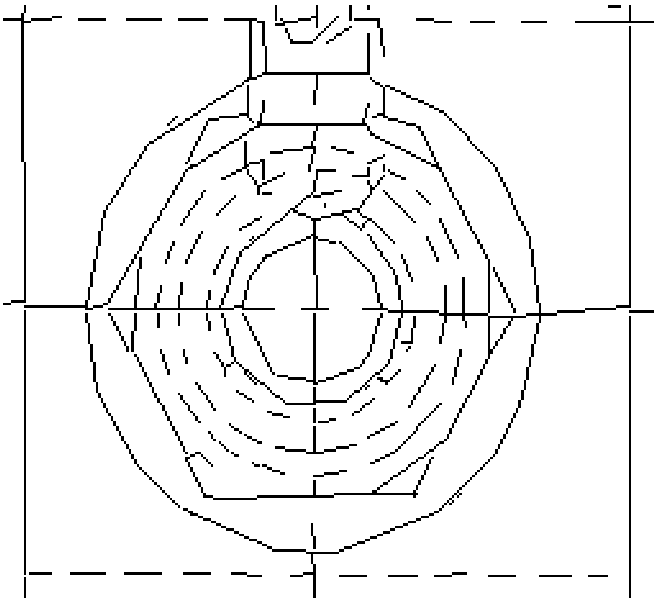


Figure 4.20: Reduction with error = 3

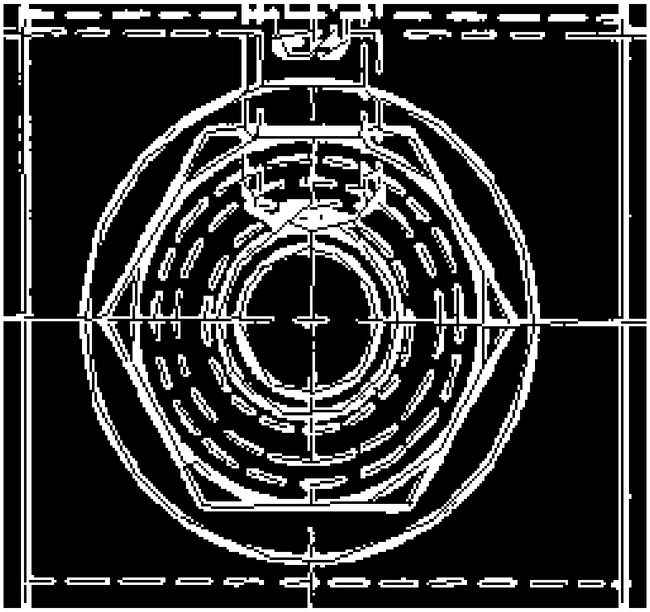


Figure 4.21: Reduction with error = 3 overlaid on original

5.0 Problems

This sort of system makes several types of errors.

5.1 Line Fuzz

Rough edged lines on scans cause serious problems. The thinning algorithm takes the points on the rough edge to be different lines, and the result is small lines running from the centre of the true line out to where the peaks on the rough edge were. The final vectorization shows lots of short lines that look like fuzz, as shown in figure 4.22.

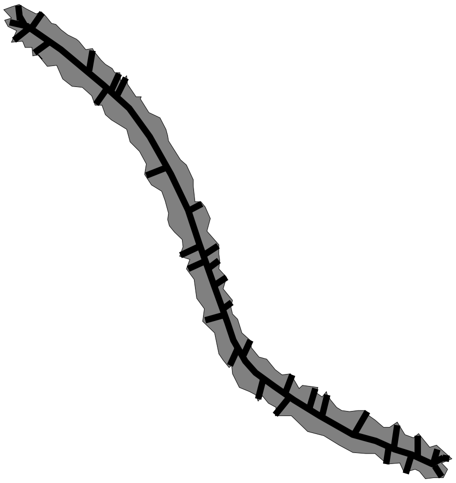


Figure 4.22: Fuzz on line

5.2 End Point Artifacts

The thinning process often introduces a great deal of noise around the ends of lines, as shown in figure 4.23, which then affects the vectorization of the line.

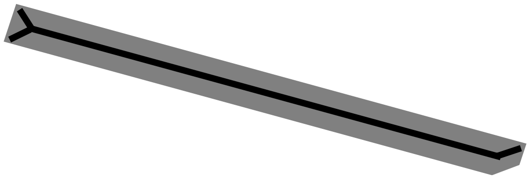


Fig 4.23: Line end noise

In figure 4.23, the right end of the line has been bent upward, while the left end has been split, one line veering up and the other down. This type of distortion is a particular problem because higher level processes often depend on the directions of the ends of lines. For example, a system that found dashed lines would identify very short lines all pointing in the same direction as a single dashed line; the end point directional information would be crucial. Split ends and end deviations mess up directional information just where it is needed most.

5.3 Intersection Artifacts

Just about all thinning processes distort the image near intersections.

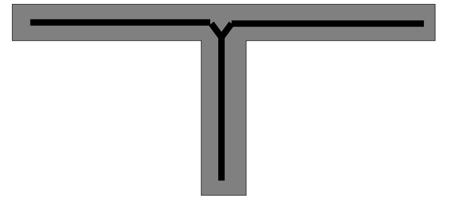


Fig 4.24: Intersection T displacement

There are two major types of intersection distortion, *T displacement* and *X destruction*. T displacement distortion appears in figure 4.24. The horizontal line is drawn toward the line that intersects with it: T starts to look like Y. The second major type of distortion is X destruction, shown in figure 4.25. Two lines that intersect at a small angle are incorrectly vectorized as two lines merging into one and then splitting into two again. This type of distortion is often called “necking.”

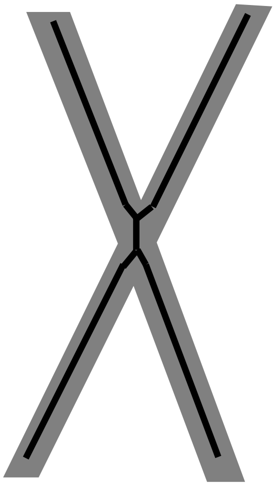


Figure 4.25: Intersection X destruction

5.4 Problems of Scale

The scale problem is of fundamental importance in vectorization. Consider a line with a bump in it, as in figure 4.26. Basically, a very small bump should be ignored and the image should be vectorized as a single line. A large bump should be vectorized as another line touching the first. It is unclear what should be done with a medium sized bump. A line with a bump might be the silhouette of a submarine, or it might be an ordinary line. A human would use knowledge about the rest of the image to determine how to interpret such a bump.

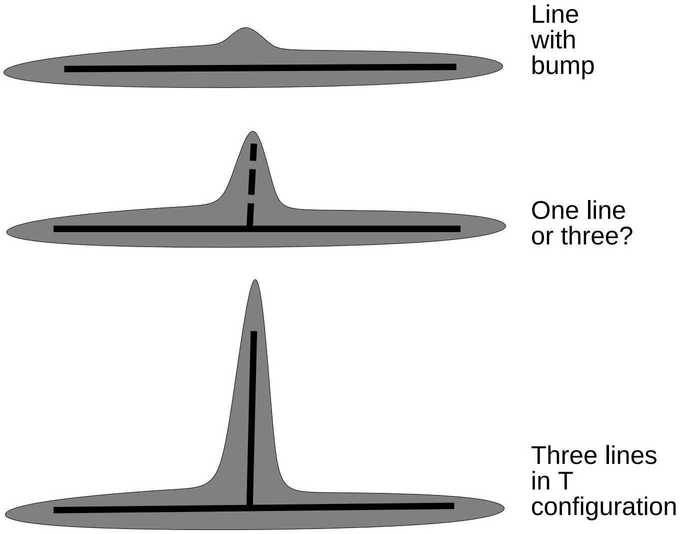


Figure 4.26: Problems of scale

5.5 Inside Bias

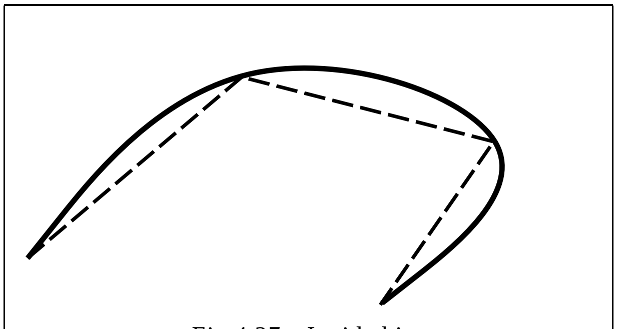


Fig 4.27: Inside bias

One of the major problems with the chain reduction algorithm is that it has an *inside bias,* meaning that the vectors approximating the curve tend to lie along the inside of the curve. In figure 4.27, the dotted line approximates a solid curve. Because of the way the algorithm bends the vectors, the vector approximations to the curves always lie on the inside of the curves in question. The inside biasing error appears in figure 4.21 with the vectors representing the outside circle.

Chapter 5

Vision, Knowledge, and Vectorization

The vectorization of maps and engineering diagrams is a difficult problem. This chapter examines whether or not the vectorization of diagrams is theoretically possible and what knowledge and techniques might prove useful. This chapter also glances at how knowledge of the human vision system might help solve the problem. It ends with a basis upon which a machine vision system may be implemented.

The first step is to define the problem precisely. Since there are theoretically a nearly infinite number of features that could appear on a drawing, this thesis restricts the domain to engineering drawings containing only circles and lines. Although being able to vectorize circles and lines will not solve larger vectorization problems like those posed by optical character recognition, it will suffice for engineering diagrams.

1.0 Inverse Function Theory

The first question to consider is whether or not vectorization is theoretically possible. Consider that graphics is the process of turning a model into an image: vectorization is “antigraphics,” the inversion of the rendering equations. Vectorization takes an image that might have been produced by some rendering and finds a model that could be rendered to get that same image.

This conception of vectorization can be formalized. The method involves considering as a model (M) the CAD file that generates a drawing. The plotting of this file can be considered a function F that generates a raster image I such that

$$I = F(M). \tag{5.1}$$

The problem of vectorization thus becomes one of finding an inverse to the function F and computing

$$M = F^{-1}(I). \tag{5.2}$$

F is not isomorphic, because whether the model contains one long line or two collinear shorter lines, the image I will be the same. F in general is therefore not properly invertible, but an approximation,

$$F_a^{-1} \approx F^{-1} \tag{5.3}$$

does exist for a given I such that

$$I = F(F_a^{-1}(I)). \tag{5.4}$$

As well, let

$$M_a = F_a^{-1}(I) \tag{5.5}$$

then

$$F(M_a) = I. \tag{5.6}$$

The approximation, M_a, to the model, M, is therefore the best vectorization possible given the information in I.

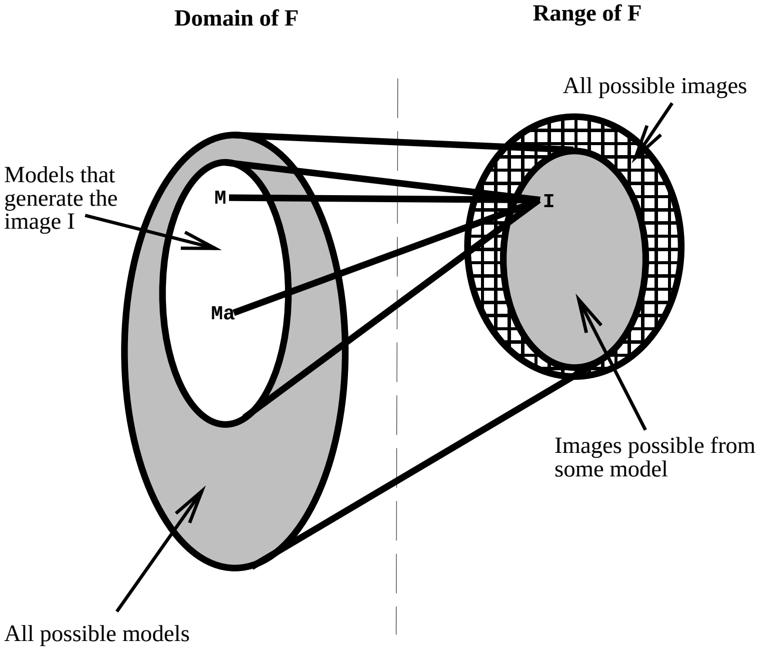

Domain of F
Range of F
All possible images
Models that generate the image I
M
I
Ma
Images possible from some model
All possible models

Figure 5.1: Relation of model and image spaces

This set of relationships is shown in figure 5.1, which shows that while finding the exact model that produced the image may be impossible, recovering a model that would produce the same image is a distinct possibility.

1.1 Size of Search Space

Since finding a good model is possible, the next question is whether or not it is practical. The function $F(M_a) = I$ could be inverted through an exhaustive search of the model space. To calculate an approximate size of this space, we first assume that the diagram is an E size (36x48-inch) engineering drawing scanned at 400 dpi that has fewer than 10,000 drawing elements on it and has no lines wider than $1/8$ of an inch. The model space is for a two dimensional model of a drawing, and it does not matter that the drawing may represent a three dimensional model. We can also assume that the image is wider than it is high without loss of generality. Now we define h to be the height of the image in inches, w to the width in inches, d to be the dpi, and n to be the number of elements. The thickest line on the image will have a thickness of t in inches. For two lines at different angles to be rendered differently, they must have slopes that make the end points differ by at least half a pixel. Since the maximum line length is dw pixels, the minimum angle change must cause an end point change of at least half a pixel. To make the calculation of these values possible, the direction of a line must be stored to an accuracy of

$$\frac{1/2}{dw} \tag{5.7}$$

radians. This level of accuracy means that there are

$$\frac{2\pi}{\frac{1/2}{dw}} = 4\pi dw \tag{5.8}$$

possible different directions for each line. The largest circle on the drawing can have a maximum radius of $w/2$ inches and must be stored to an accuracy of half a pixel, together with its positive/negative curvature indication. There are, accordingly, $2dw$ different radii. The length of a shape needs to be stored to an accuracy of one pixel. If it is an arc, the longest length of the arc will be πw inches, so that there are $wd\pi$ different lengths for shapes. As there are n shapes, the total size of the search space is about

$$wdhd\; dt\; 4\pi dw\; 2dw\; wd\pi\; n \tag{5.9}$$

which reduces to

$$8\pi^2 w^4 h d^6 t n \tag{5.10}$$

or, for this example, about 10^{29} models. For each model we would want to render and then compare the images that have $wldd$ pixels, which amounts to about 10^{37} comparisons. Making so many comparisons is clearly impossible using current computer technology and will probably remain impossible for the foreseeable future.

Another way to invert the rendering equations is to use simulated annealing. The problem is posed as "minimize the value of

$$|F(M_p) - I| \tag{5.11}$$

by varying the vector M_p," where the vector describes a model that can represent every possible drawing. M_p is a particular model in the space of all possible models. This function theoretically finds the model that is a good fit the image. This function cannot be used as a technique, however, because M_p has too many dimensions [Cora:87]. Simulated annealing can only solve thousands of variables when the the objective function is locally very smooth and can be evaluated very rapidly.

Yet another way to consider inverting the rendering equations is to employ the genetic algorithm metaphor [Dave:91], which attempts to make this inversion using a rough interpretation of the theory of evolution. The metaphor posits that creatures are trying to survive and reproduce and thus to evolve. There are major stages of evolution which differ in the level of sophistication attained. These stages – which are interpreted, for the purposes of engineering diagrams as points, lines, and so on – correspond to species. In each species there are those that are weak – small areas – and those that are strong – large areas. Lines or regions (creatures) that are too small starve and disappear. Large entities reproduce and form similar creatures that attempt to grow larger.

The DNA passed from one generation to the next is the vector

$$(x, y, curve, width, direction, length) \tag{5.12}$$

This vector describes any arc or line segment. The coordinates x, y represent the location of the centre of the segment. The direction of the creature is $direction$, and $width$ and $length$ are self explanatory. The element $curve$ is $1/radius$ of the curve. It is positive if the curve bends right, negative if the curve bends left, and zero for a straight line. Any creature can therefore be represented conveniently by a seven dimensional vector. This method has the same sort of problems as simulated annealing in that it is not computationally feasible with this many degrees of freedom.

2.0 Vision

In *Theory of Edge Detection* , Marr comments that “vision is the process of discovering from images what is present in the world and where it is” [Marr:82]. Finding out what is on a diagram by brute search techniques is prohibitive, as observed above. Various animals’ vision systems, however, provide helpful hints about improving machine vision. The study of human vision has already contributed much to this area of knowledge [Marr:82], [Berg:90]. Figure 5.2 is a greatly simplified sketch of the retina. This chapter examines how images travel through the optical systems of humans and certain animals and how information is obtained at different stages before the image gets to the brain. At each stage, information that seems to yield a hint for machine vision is examined.

2.1 Retina

In human vision, light enters the eye and is focused by the optics onto the retina. In the back of the retina the photoreceptors in the rods and the cones turn the light energy into neural signals. These signals are fed into the bipolar, amacrine and horizontal cells that feed into the retinal ganglion cells (see figure 5.2).

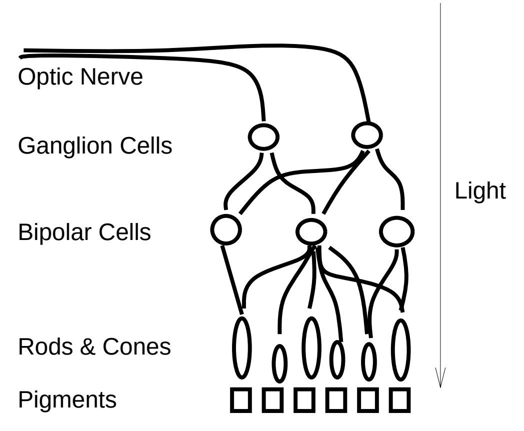


Fig 5.2: Sketch of retinal cells

One clue to improving conversion systems appears when the retina in examined. The human vision system arranges the sensors in the retina in a roughly hexagonal pattern with smaller hexagons towards the centre of the retina [Core:79]. This is probably better than the rectangular arrays used in most image processing. Hexagonal arrays are the densest way to pack circles on a plane [Full:75]. Deutsch has found that thinning problems seem easier to solve on hexagonal arrays than on rectangular ones [Deut:72].

2.2 Ganglion Cells

The ganglion cells are the cells in the retina that are triggered by light and start the signal that is transmitted to the brain. Each ganglion has around it a more or less circular area called a receptive field. When light falls in the receptive field, the ganglion is affected. The receptive field for retinal ganglia for warm blooded vertebrates has at least one characteristic that is thought–provoking for these purposes. Kuffler’s research indicates that light stimulation in the centre of the receptive field for a cell tends to affect the cell in a way exactly opposite to the way stimulation near the edge of the receptive field does. If light in the centre of the field causes the neuron to fire and be active, then light near the edge will suppress the tendency to fire [Kuff:53].

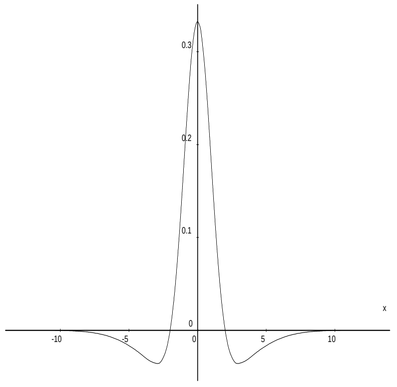


Fig 5.3: DOG function

More recent work has shown that the response of some retinal neurons is directly related to the summation over the receptive field of the stimulation scaled by a factor that is dependent on the distance of the stimulation from the centre of the receptive field. The scale factor versus the distance is a transfer function that closely approximates the difference of two Gaussian functions shown in figure 5.3. A model for the response of the neuron based on these observations was proposed by Rodieck [Rodi:65] and is often called the Difference Of Gaussians (DOG) model. This DOG model accounts for the visual phenomena of Mach bands and the Hermann grid, described below (how the

DOG function accounts for both of these phenomena is described in detail by Sekuler and Blake [Seku:85, pp. 71–75]).

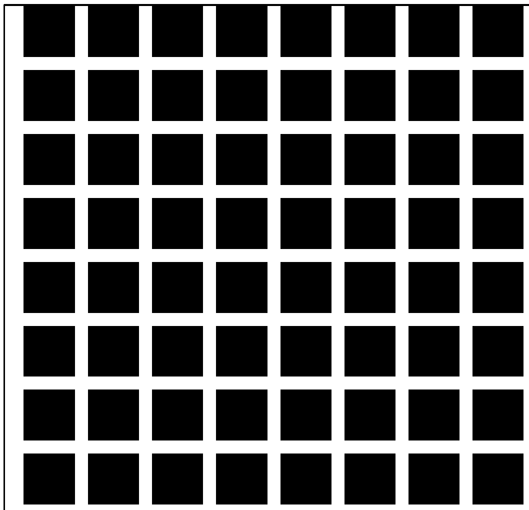


Fig 5.4: Hermann grid phenomenon

The Hermann grid (see figure 5.4) was described in the nineteenth century by Ludimar Hermann. The noteworthy feature of the Hermann grid is that a person who stares at it sees grey circles at the corners of the black boxes. This effect can be explained by the DOG model of the frequency response curve of the human visual system. Consider the receptive field of a single ganglion. The centre is activated by light reflected off the bright white lines, while the edges of the receptive field are activated by dark areas. When the receptive field is centered on a grid line away from an intersection, not only does the white grid line stimulate the cell but the black borders on the edge of the receptive field also stimulate the field and appear very white. When a receptive field centres on a grid intersection, the larger amount of white than black immediately surrounding the intersection falls on the outer part of the receptive field and inhibits the cell slightly, making the viewer see the white intersections as darker than the grid lines. Ultimately the grid intersections fool the perceptual system into seeing them as gray, particularly if they do not fall on the centre of the visual field where the receptive areas are smaller.

The DOG function is one explanation for why ganglion cells respond more to edges and bars than to uniformly lit surfaces. Changes in light intensity, such as those that occur at edges, may be more important for vision than the actual intensity values. Conversion systems can likely learn almost all of what they need to know by looking only at edges.

2.3 Lateral Geniculate Nucleus

The retinal ganglion cells are connected by the optic nerve to the *lateral geniculate nucleus* (LGN) and *superior colliculus* in the optic lobe in the brain. The LGN senses variations in colour and illumination level [Seku:85, p. 107]. It responds little to the total amount of light falling on the retina. This response characteristic implies that the LGN deals mostly with edges.

2.4 Visual Cortex

Experiments by Campbell and Robson were important in introducing frequency domain processing into studies of human vision [Marr:80, p. 187]. One of their important contributions was their multichannel hypothesis, which states that the human vision system contains different kinds of neurons, each highly attuned to perceiving bars of a particular width and orientation, and that these neurons provide input information for neurons performing higher level vision functions. These supporting neurons are especially sensitive to lines at orientations near the horizontal or vertical [Seku:85, p. 119]. Usually a particular cell has a sensitivity range somewhat narrower than 15°. This orientation response of cells is done in the visual cortex. This fact implies that computer vision could be done by having a system recognize rectangles at various orientations. Rectangles can likely be recognized using lower level information about where the edges of the image are. This is a useful idea that is used in the model proposed in this thesis.

In the visual cortex there is a large “3D array” of cells. Two dimensions of this array correspond to what part of the eye is being activated, but each cell in the third dimension is activated only when the orientation of the stimulus lies within a range appropriate for that cell. Hubel and Wiesel, who discovered this and got a Nobel prize for their work in vision in 1981, showed that the cells in the cortex are arranged into hypercolumns of cells specific to different orientations.

The mere fact that so much of the brain is dedicated to determining the orientation of visual stimuli lends credence to the idea that the direction of each individual “pixel” is very important in vision. The human visual cortex has a representation for the image in which one cell activates for each (x–position, y–position, direction) triplet. For machine vision, knowing the natural direction of a pixel is probably important.

2.5 Gestalt Perception Ideas

The full operation of the visual cortex and how the brain makes sense of what is viewed cannot currently be explained from this neural biology point of view, but traditional psychology offers some insights. The basic model of vision employed by Gestalt psychologists has provided much of the basis for contemporary work and has generated several fundamental principles of human perception [Seku:85]. These principles are described by Lowe [Lowe:85] as:

- Proximity

  Objects near one another are grouped together as one perceptual unit.
- Similarity

  Objects similar in colour, size or orientation are often grouped together.
- Continuation

  Objects that lie along a smooth path or line are grouped together.
- Closure

  Curves are completed so that they form a closed region.
- Symmetry

Objects that are bilaterally symmetrical in their relative positions are grouped together.

- Familiarity

  Objects that humans are used to seeing together are grouped together.

All of these principles can be used in implementing a computer vision system that links lower level information together to form higher level information. Some of these ideas are used by Bergevin and Levine [Berg:90] and Connelly and Rosenfeld [Conn:90].

2.6 Relation to Machine Vision

Current neural psychology suggests that human vision relies on low level recognition in the retina and higher level recognition as the signal travels along the optic nerve and into the visual cortex. In a computer vision system designed with this model of vision in mind, first the system should find low level information, which it should then use to build up more complex high level information about the image. This brief analysis of natural vision has raised several other ideas, as well, that could be useful for machine vision:

- A hexagonal image array may be better than a rectangular array for image representation.
- Changes in image intensity (edges) are important.
- The direction of individual “pixels” is important.
- Sensors for particular directions should have a resolution better than 15^o.
- Lower level information can be used to find an intermediate level of information that represents the image in terms of bars at various orientations and widths. This information can be used to find higher level information.
- Objects that are close or collinear can be grouped together.
- Small gaps should be closed.

These ideas from human vision can simplified and modelled to some extent in a computer vision system. They are used in the VKV vectorization system proposed later in this thesis.

3.0 Domain Knowledge

This section looks at what is known about the domain of line drawings and how this knowledge can be used. Mackworth has shown that vision only becomes possible through the use of domain knowledge [Mack:83]. The feat that the human brain performs when, given an image and all the hundreds of models that could have possibly computed that image, it picks out one model it thinks is correct only becomes explicable through the realization that humans use sophisticated “a priori” knowledge about what is possible in the model.

Cugini *et al*approach the problem of the a priori knowledge required in mechanically vectorizing engineering drawings through an exploration of the technical but not formal language of engineers and people who draft engineering drawings. They find that “[the] language used for this communication has well defined syntactical and orthographic rules. From the orthographic point of view, the basic elements are [line] segments, arcs and other well defined curves.” [Cugi:84, p. 338] Engineers can look at engineering diagrams and understand what they mean because engineers have a well defined syntactical and orthographic system, and they expect the marks on the page to fit into this system. Another way to conceptualize what engineers do when they look at an engineering drawing – and what, by extension, computers must do as well in machine vision – is through the analogy of the AMES room. The size and shape of the AMES room is distorted, but when viewed from the right spot it looks like a normal room containing objects of unusual sizes. Like an engineer reading an engineering diagram, a human finds the right place to look at the AMES room by relying on a priori knowledge about what the room ought to look like.

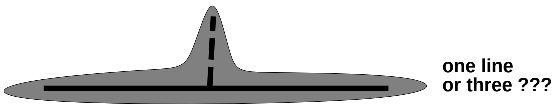


Fig 5.5: Scale problem

The need for a priori or domain knowledge becomes apparent when the scale problem discussed in section 4 is considered. In this problem, which is of fundamental importance in vectorization, there is a line with a bump in it, as in figure 5.5. A small enough bump is just a bump in the line and should be ignored; a large enough bump is a separate intersecting line. A human would use knowledge about the rest of the image to determine how to interpret each bump.

Because knowledge about the domain is so important in interpreting images, it is useful to list some facts about line drawings:

- Once pepper noise is removed, every black pixel is part of an object.
- The nearby edges often make it possible to figure out the direction of the object of which a given pixel is an element.
- A line consists of at least one point.
- A line is a good approximation of a very thin rectangle.
- A thick rectangle contains a thin rectangle.
- The direction of a point on a circle is the same as the tangent line to the circle.

- The engineering diagrams that this system will vectorize will be able to contain only lines and circles.

These facts can be used various ways to help understand the image.

4.0 Proposed Model

This section proposes a model for a machine vision system based on some hints from natural vision and domain knowledge of line drawings. In this system the recognition works up in layers. Each layer detects a higher level object and feeds information to the next layer above it to detect even higher level information. The layer order is:

- Pixels

  These correspond to individual sensors in the retina (rods and cones). Two kinds of pixels are considered important: black pixels that must belong to some object and edge pixels that determine the boundary of the object.
- Edges

  The edges define the object completely. They correspond closely to the information available after the processing by the retinal ganglion cells.
- Direction

  The directions of the pixels are determined to get an initial estimate of the direction of the object. Directions are determined using the longest line of sight that lies entirely on black pixels. This stage aims to get the direction accurate to within 15^o and corresponds to a little bit of what happens in the visual cortex.
- Thin Lines

  The directional information is used to find thin lines that approximate the image. These lines are used as the first approximation of the thicker lines that form the Blocks stage.
- Blocks

  Rectangular blocks that have position, orientation, and width are found.
- Curves

  The blocks are used to find lines and curves in the image.
- Intersections and Gaps

  Collinear lines and curves are connected to form complex objects. Small gaps are closed, and objects that should be connected are.
- Objects

  The objects that finally emerge provide a machine interpretation of the image that is used to form the model that produced it. This model is the vectorized image.

Chapter 6

Implementation

This chapter uses the ideas about vision and knowledge to implement a vectorization system called VKV. This system consists of over 5000 lines of C++ code that do the vectorization and another 10000 lines of C++ and X11 that view and manipulate input images and results.

1.0 Finding Shapes

The complete search space – all the possible symbols that could be in a diagram – is clearly too large for an exhaustive search, but there are many heuristics that can be applied to reduce the search space. These heuristics can be designed from domain knowledge about engineering diagrams.

One of the simplest of these heuristics is a structured search based on the fact that a line must contain a point. The idea is to find a simple shape like a point quickly and then use it to find a more complex shape that contains it, such as a line. Because every black pixel belongs to some shape, any black pixel can be used to find a line. Since the pixel in question must, by definition, be on the line, the search space for the line is reduced significantly. This structured search principle – in which simple objects like points are used to reduce the search space for more complex objects like lines – is a basic organizing principle of this vectorization system. The procedure used here is:

- Find a point
- Find edges
- Find a line
- Find a rectangle
- Find an arc
- Find an arctangle (an arctangle is an arc with width)

- Repeat these steps for another line or arc on the diagram

At each one of these levels, other knowledge is used to reduce the search space.

1.1 Data Representation

The representation of these shape objects in memory is:

- POINT

  Each point is represented by two real numbers, its x and y coordinates.

- LINE SEGMENT

  A line segment is described by POINT, which is the centre of the line, and two real numbers that represent the direction and length of the line. Directions are always normalized to lie in the first two quadrants of a circle, and they are stored in radians.

- RECTANGLE

  A rectangle is specified by a LINE SEGMENT that represents the centre of the rectangle and a real number that specifies the width.

- ARC

  ARC is similar to LINE SEGMENT, in that it has a centre point, length, and direction. Its length, though, is the length along the arc, and its direction is the direction of the tangent to the arc at the centre point. The radius of the arc is represented by a value called “curvature”, which is the inverse of the radius. The curvature is positive if the arc bends right from the tangent and negative if it bends left.

  Curvature is a useful value which simplifies many of the formulas that involve computing with arcs. Figure 6.1 shows an arc that bends to the right. If this arc has a radius of 10, its curvature would be $+0.1$. The curvature of straight lines is zero.

- ARCTANGLE

  ARCTANGLE is specified by an ARC that represents the centre of the arctangle and a real number that specifies the width.

- SHAPE

  SHAPE unites all the previous types. It includes five real numbers: x–coordinate, y–coordinate, direction, length, width, and curvature.

1.2 Finding Points

This procedure for identifying shapes starts at a black point not part of any shape already found. Such a point is chosen at random from the image. Since most black pixels are not edge pixels, a point that is inside an object is more likely to be picked than a point the edge of the object would be.

The direction of the line on which this point falls is computed next. This direction is called the line of sight direction and is presented by Chen and Hsu [Chen:89].

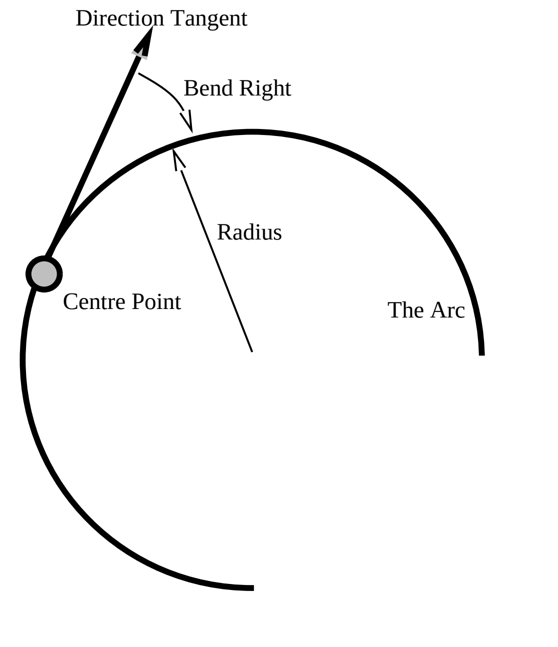


Fig 6.1: Arc curvature

The line of sight direction is taken from the longest straight line that intersects the point and crosses only black pixels. A flaw in this technique arises, however, because each point lies in a rectangular black area. This technique identifies the direction of the rectangle's diagonal but fails to find the direction of the actual rectangle (see figure 6.5). In many circumstances, though (as long as the rectangle is fairly narrow), this technique does provide a simple way to find a good approximation of the direction.

This "line of sight" direction gives an initial approximation of the direction of the shape. To normalize the shape, the point now considered its origin is the centre of this line. This new point is stored in the SHAPE data structure and used for further processing. The initial point is discarded.

An example of a line of sight is shown in figure 6.2. The raster image is black on a speckled white background. A point and line of sight are displayed as a white line on the left side of the raster circle.

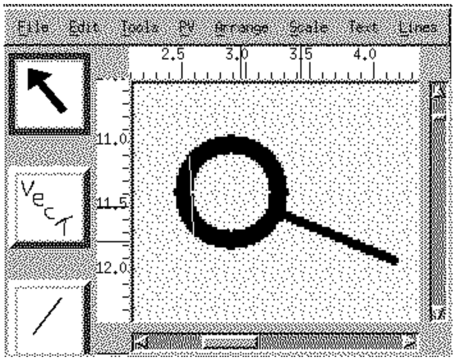


Figure 6.2: Line of sight

1.3 Finding Edges

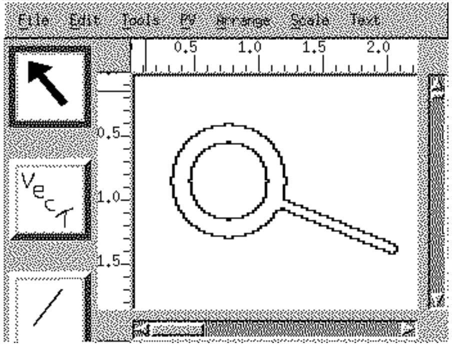


Fig 6.3: Edge cache

The vectorization system next finds the edges of the shape. Since the system operates on black and white images, finding edges is trivial. Any black pixel with an eight connected white neighbour is considered an edge pixel. The edge pixels of the current object are cached for use in later processing. (If this work were extended to process grayscale input images, an edge detection system like the one proposed by Shen and Castan [Shen:92] could be used to find the edges.) An example of the edge cache appears in figure 6.3.

1.4 Finding Lines

Once the direction of the initial point has been calculated and the edges of the shape have been found, the vectorization system next identifies the line upon which the point lies. The position and direction of the current point are randomly perturbated many times, and the longest entirely black line going through this point is found and kept. The length of the line is found by intersecting the line with the cached edge information. This stage of the vectorization process tends to move the centre point of the line near to the centre of the real shape.

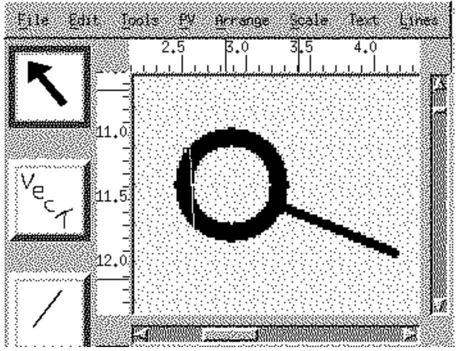


Fig 6.4: Moved line

Statistically, the distribution used for the perturbations approximates a normal curve. Small perturbations are therefore more often used than large ones. In empirical tests that compared how long the system took to find the longest line with various distributions, this distribution converged and produced this line much faster than when all possible amounts of perturbation were equally likely.

The line found using the perturbations is shown in figure 6.4. This figure is almost identical to figure 6.2. In a more complex shape, the centre of the line would need to be moved before it could achieve its maximum length.

1.5 Finding Rectangles

In this stage the centre of the shape is identified precisely, and the accuracy of the direction of the line is updated. The line found in the last stage is slowly widened into a rectangle. As the rectangle widens, the centre line of the rectangle must remain on a black area of the image, but its edges only have to be mainly on black. As the width grows, the direction and position of the rectangle are perturbated according to the same distribution used when finding lines. The rectangle with the largest area is saved and passed to the next stage of processing. The direction saved at this stage is the direction of the line through the centre of the rectangular region rather than the direction of the diagonal. Figure 6.5 shows the distinction between these two directions. At the end of this stage, accurate directional information about the shape has been found, even if the shape is curved, as is shown in figure 6.6.

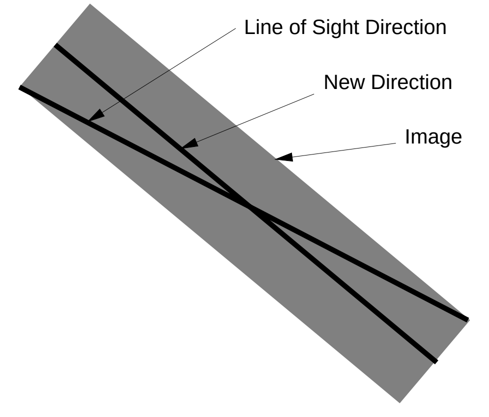


Fig 6.5: Shape directions

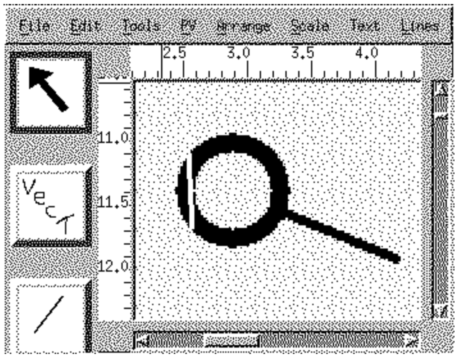


Figure 6.6: Rectangle

1.6 Finding Arcs

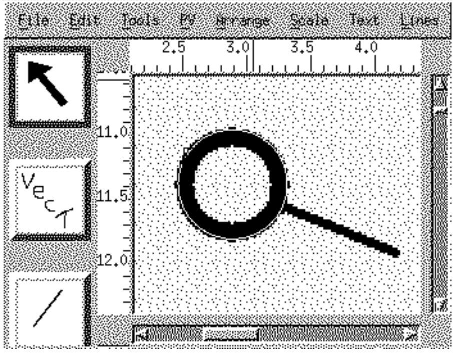


Fig 6.7: Arc

Arcs are found from the direction and position of lines. A wide range of possible curvatures for an arc are tried, and the longest arc found is passed to the next stage of processing. Usually the complete arc is found, but its position within the shape tends not to be very well identified. This problem is apparent in figure 6.7: the arc hugs the outside edge of the circle because arc length is being maximized. Note that if the arc extent angle is too small, the shape is assumed to be a straight line and the rectangle found in the previous stage of processing is used as the final result. Otherwise the algorithm proceeds to the next stage of the search, finding arctangles.

1.7 Finding Arctangles

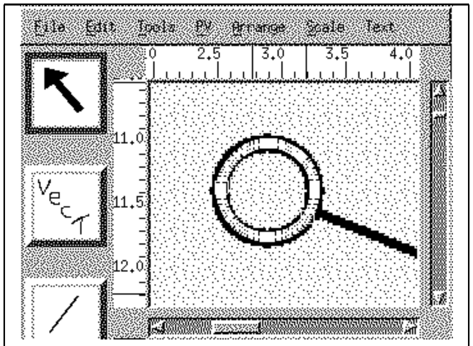


Fig 6.8: Arctangle

The procedure for finding what are here called “arctangles” – arcs with width, like pieces of a washer – is much like that for upgrading lines to rectangles. The arc is slowly grown wider into an arctangle. The arctangle with the maximum area is kept. The position, width, and curvature of the arc shape are perturbated as the arc is grown, but the direction is untouched because it was found accurately in the stage that finds rectangles. An arctangle has been found in figure 6.8.

1.8 Finding More Shapes

One shape in the drawing has now been found. It is marked as found, and the vectorization process reiterates to find another shape. A shape, once identified, may be considered one “stroke” in the image if it meets the qualifications imposed by domain specific knowledge, described below.

1.9 Domain Knowledge Used

This system relies on the user’s input of a large amount of critical domain knowledge. The size of the perturbations applied at each stage and the decision whether or not a shape that has been found qualifies as a stroke are controlled by domain knowledge about the drawing. The specific items used are:

- Maximum Line Width

  The maximum allowable width of a line;

- Minimum Radius

  The minimum radius of any curve on the drawings. This should be more than half the maximum line width or else it will be possible to get circles that lie completely inside one line.

- Maximum Radius

  Any curve whose radius is greater than this will be considered a straight line.

- Minimum Line Length

  Any line shorter than this is a misinterpretation of the drawing. This should be larger than the maximum line width.

- Maximum Overlap

  The maximum length of an area where two lines overlap.

- Minimum Shape Area

  The minimum area of any stroke in the drawing.

- Maximum Shape Area

  The maximum area of any stroke in the drawing. This is useful in filtering out large solid regions that should not be vectorized.

- Maximum Percent Overlapped

  The maximum percentage of one stroke that can be covered by another stroke.

These parameters can all be set in a file that describes a class of documents. They reduce the search space enormously and considerably improve the interpretation of the image. Experimentation was done with these parameters and others, but only these had a significant impact on the final results. All variables are floating point, so that results can have a subpixel level of accuracy. The sample image, having undergone all the steps in processing described so far, appears in figure 6.9.

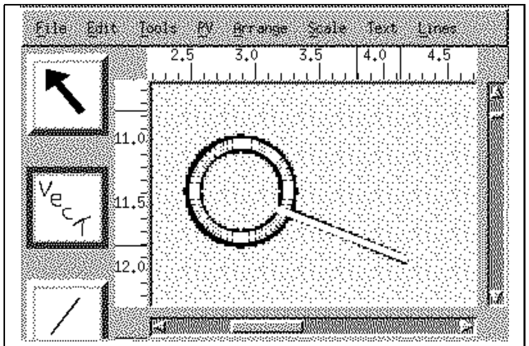


Fig 6.9: Two shapes

2.0 Finding Intersections

Once all the strokes – rectangles and arctangles – have been found, the connections between them need to be identified exactly. The method used to connect the strokes reflects the Gestaltist principles of line continuation and termination.

The aim of this stage is to connect the ends of strokes that are close but not quite touching, to try to make end points meet or to make the end point of one stroke lie on another stroke. First, all the strokes on the image are examined to see if the end of one stroke lies within half a line width of another stroke. If so, the strokes are lengthened or shortened so that they intersect with others. All the strokes are then reexamined, and the process is repeated until no strokes are changing. It is important to note that the basic shape and direction of a stroke never change when the stroke is lengthened or shortened to meet another. Only its length changes. With arcs and arctangles, it is the arc length that is adjusted. The algorithm is easiest to show with pseudo code.

```
set somethingChanged to true
while somethingChanged
    set somethingChanged to false
    for each shape in the drawing
        set shp to this shape
        for each end point of this shape
            set p to be this end point
            set s to shape closest to p
            if p is near s
                change the length of shp to make p
                        as close as possible to s
                if the length of shp changed
                    set somethingChanged to true
                end if
            end if
        end for
    end for
end while
```

To help illustrate the algorithm, figure 6.10 walks through all the stages of an intersection adjustment. The process of intersection adjustment appears in figures 6.11–6.13. The adjusted strokes are saved in a file, and the conversion is complete.

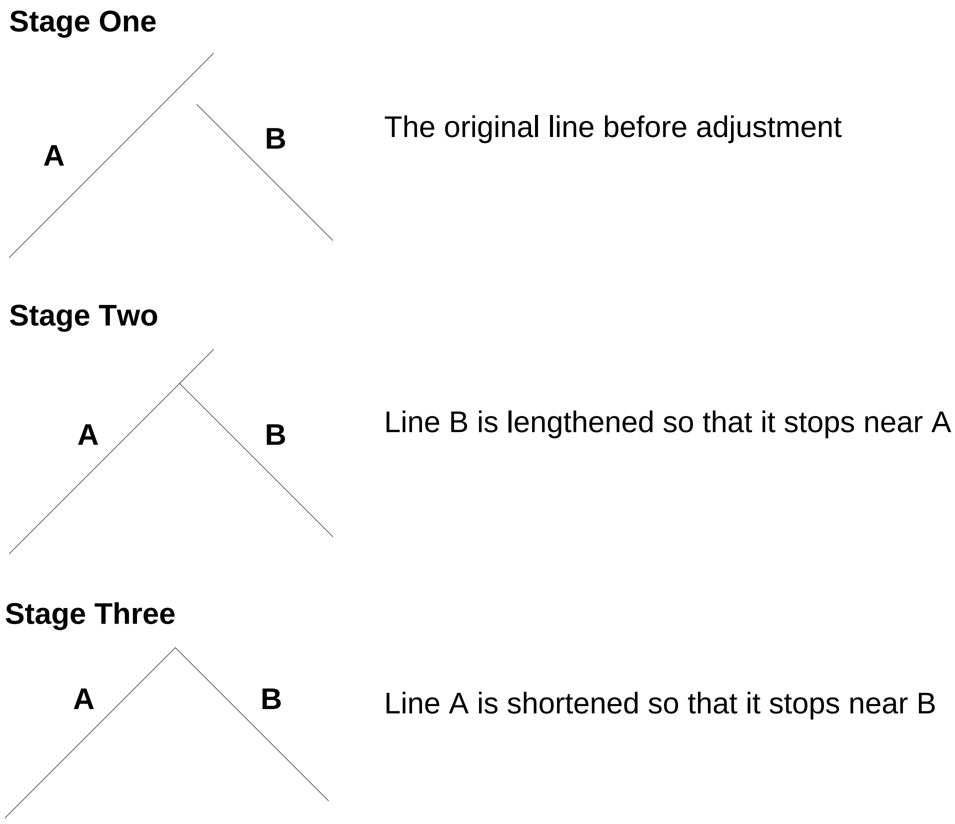

Stage One
A
B
The original line before adjustment
Stage Two
A
B
Line B is lengthened so that it stops near A
Stage Three
A
B
Line A is shortened so that it stops near B

Figure 6.10: Intersection adjustment

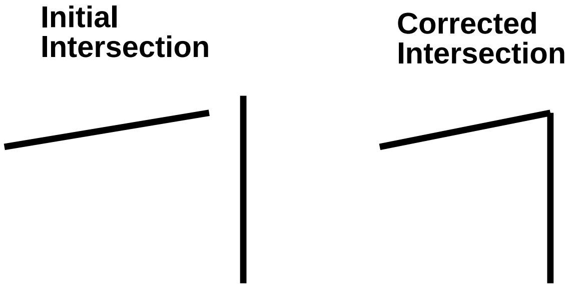


Figure 6.11: Corner adjustment

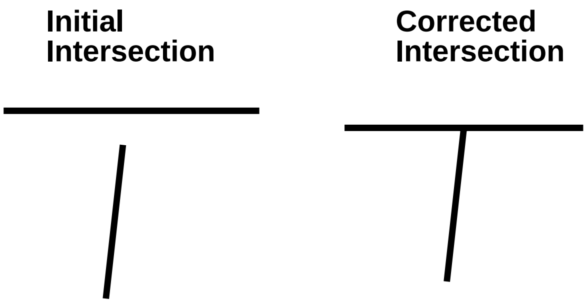


Figure 6.12: Tee adjustment

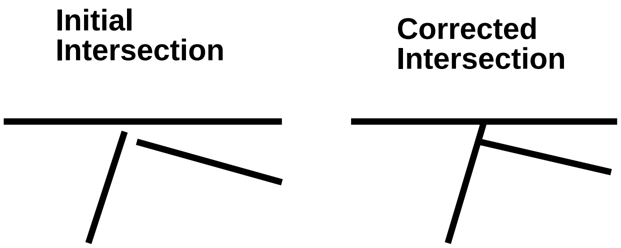


Figure 6.13: Complex adjustment

Chapter 7

Analysis

This chapter describes a comparison of the vectorization method proposed in this thesis–VKV–and classical vectorization methods.

1.0 Experiment

So that the “correct” results may be known, a drawing that had been electronically drafted was plotted. The plotted version was blueprinted (this technique is actually called the diazo process, and the product is not technically a blueprint) and then scanned. Blueprinting injects realistic noise into the process. The blueprint was scanned, and the scan was processed under both VKV and the classical system described in chapter 4. The results were compared to the electronic drafted version to determine the errors made by each.

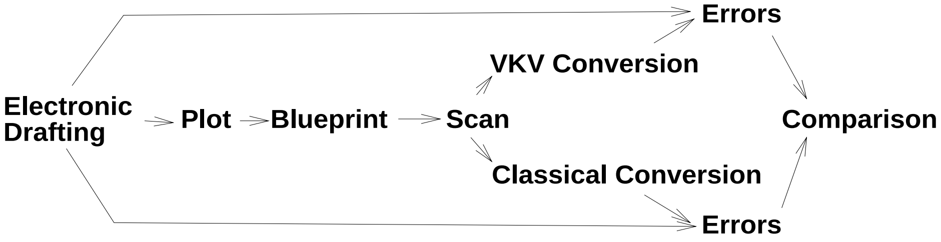


Figure 7.1: Data flow in experiment

The drawing (figure 7.2) contains lines, curves and intersections like those that occur on engineering diagrams. Unlike many engineering diagrams, it contains no text, but converting text is beyond the scope of this system. The drawing chosen has curves, lines, lines that join curves, lines that cross curves, lines at various angles, and intersections of thick and thin lines.

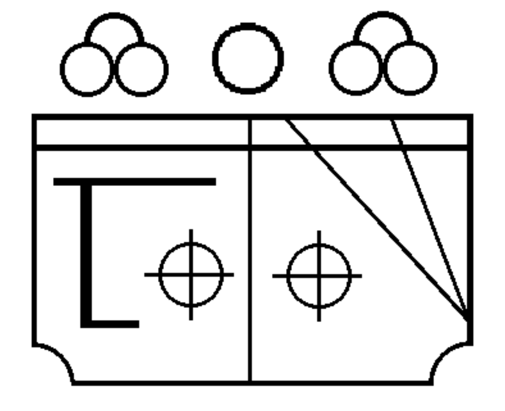


Fig 7.2: Original CAD drawing

It has a great variety of potential problems. The drawing was plotted, and a blueprint was created. It was scanned with a Microtek 300zs scanner to get the input image.

The scan was automatically converted to get the results shown in figures 7.3 and 7.4. Just to point out what happens with some commercial products, Adobe's Streamline 2.0 was also used to vectorize the image. It produced the results shown in 7.5.

2.0 Analysis

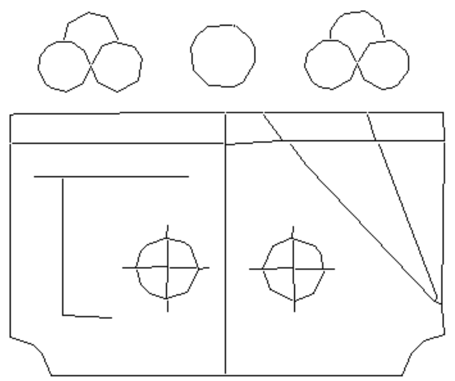


Fig 7.3: Classical vectorization

The results of both VKV and the classical vectorization methods are basically correct, but another relevant criterion for comparing the effectiveness of the two systems is how easily the resulting data may be manipulated. Where the original had one straight line, the vectorized version should have a one line primitive as well. If there are two primitives instead, editing will have to be done on two lines instead of one.

In the analysis of the results, errors will be classified into several types, and then an approximate time for correcting each type will serve as a weight. The final comparison will turn on how long it would take a human operator to fix all the errors in the vectorized drawing using a current industry standard tool, such as AutoCAD running a heads up digitization tool like CadOverlay on an Intel 486 with hardware graphics acceleration for the CadOverlay system.

2.1 Lines Missed

This evaluation criterion is simply a count of the straight lines that the vectorization completely missed. To add these lines, the operator just has to enter the end points of the line manually, which takes approximately 10 seconds per

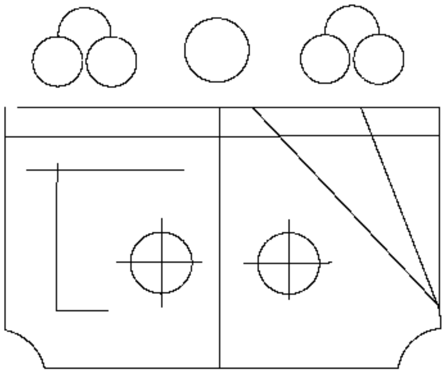


Figure 7.4: VKV vectorization

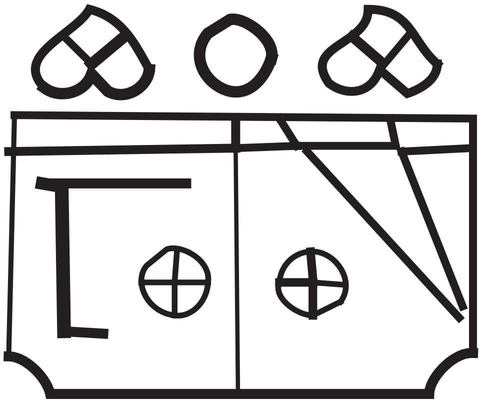


Figure 7.5: Streamline vectorization

line.

2.2 Circle Missed

This error is similar to missing lines, but to fix it the operator must enter three points on the arc, which takes slightly longer (30 seconds). Usually several incorrect line primitives need to be deleted before the circle can be added.

2.3 Tolerance Errors

These errors occur when the vectorized line is in the wrong place and must be moved slightly. It is time-consuming to figure out exactly what point on the primitive needs to be adjusted, so fixing this error takes about 35 seconds.

2.4 Intersection Errors

Intersection errors occur where two lines that should intersect fail to touch properly and one end point needs to be moved slightly. Intersection errors are faster to fix than tolerance errors because what needs to be adjusted is obvious. They take 25 seconds.

2.5 Crossing Errors

This type of error occurs where two lines that should cross precisely have acquired X–destroying intersection distortion (see the chapter 4 for a description of this distortion). This error takes a long time to fix (50 seconds). The segments of the two lines have to be joined and the intersection point deleted.

2.6 Fuzz Errors

This is one of the most frustrating errors to fix. Little lines like hairs come off the main line, making it look fuzzy. At each intersection point the intersection must be broken, the two segments of the main lines reconnected, and the fuzz deleted. Although fixing one hair only takes about 30 seconds, operators soon become frustrated. It takes about 45 seconds on average to fix this kind of error.

2.7 Comparison

The raw data from the comparison is shown in table 7.1. The VKV image had an estimated correction time of 50 seconds, while the classical method had a time of 700 seconds. The drawing could be redrafted by someone using AutoCAD with CadOverlay in approximately 300 seconds. The accuracy of these times would obviously vary widely depending on the AutoCAD operator, but the relative times would likely stay consistent among users.

Category	Classical	VKV	Time to Fix
Lines Missed	0	0	10
Circles Missed	11	0	30
Tolerance Errors	0	0	35
Intersection Errors	3	2	25
Crossing Errors	5	0	50
Fuzz Errors	1	0	45

Table 7.1: Vectorization errors

3.0 Critique of System

The VKV system is suitable only for line drawings formed from a small set of primitives, but fortunately engineering drawings fall mainly into this category. VKV will not work with documents composed of figures other than lines and circles. In OCR, for example, it would be possible to vectorize a font like Helvetica, but a font like Times with lots of points and curves would not work. VKV could be extended to vectorize more complex shapes like ellipses and perhaps splines, but it would never be able to vectorize all shapes, particularly complex shapes like fractal curves.

An interactive conversion system that allowed the user to correct mistakes and which made future choices based on the corrections would be an improvement over VKV. Often a mistake on one vector causes several other vectors to be distorted as well. An interactive system would help solve this problem. Another drawback of VKV is that it is computationally intensive compared to many existing systems.

4.0 Future Work

The work described in this thesis suggests several avenues for future research. One possibility would be a more interactive system that allowed the user to help the vectorizer and provided extra context knowledge where required. Methods like this can be very successful, as Jansen and Krause have shown [Jans:84]. One reason is that when the system proposed in this thesis makes a mistake, it usually causes several other mistakes in the same area. If a user could interactively correct the first mistake, the system could be prevented from making several subsequent mistakes.

Another possible area for future work would be to expand the types of shapes that the system recognizes from lines, circles, rectangles, and arctangles to ellipses and other primitive shape types. Such work could be combined with a system for OCR to produce useful results.

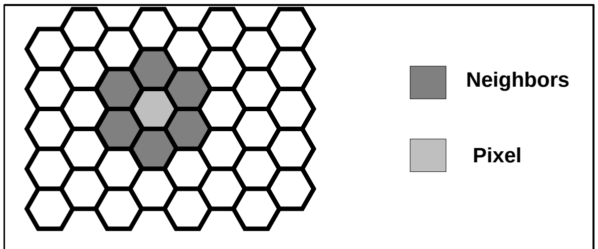


Fig 7.6: Hexagonal rasters

Currently almost all image processing is done on a raster image constructed from a square grid. Human eyes, however, seem to use an hexagonal grid – likely because hexagons provide the best packing of circles in an infinite plane. Vectorization on hexagonal grids has been pursued by Gibson and Lucas, among others [Gibs:82]. The fundamental problem of the connectivity of neighbouring pixels is much simpler on hexagonal grids, because all a pixel's neighbours are an equal distance away from it, as shown in figure 7.6. The symmetry of the whole situation is much better. As well, a thinning algorithm proposed by Deutsch [Deut:72] was easily implemented on hexagonal grids. If the vectorization method described in this thesis were extended to hexagonal rasters, the results would likely be noticeably better.

The human visual system also makes heavy use of domain knowledge, which this system does not. In particular, a human would use knowledge about the probable angles of lines in an engineering drawing, and this system would not.

A good conversion system should be able to find and reuse similar objects in the drawing as well. For example, if there is a valve in one place on the diagram and a very similar valve somewhere else, the system should recognize that they were probably meant to be drafted exactly the same way. A good system should find all the valve objects, make an optimal valve object and indicate that there are valves at all the right locations in the drawing, instead of merely describing the lines and shapes that make up each valve. This, of course, is a hard machine learning problem.

In summary, there is a tremendous number of problems that will have to be solved before a computer will be able to convert a raster image into the same sort of CAD vector file that a human CAD operator would produce. Although two dimensional CAD line drawings are surely one of the very simplest vision problems, they still present many unsolved problems.

Chapter 8

Conclusion

In a manual vectorization system, a user indicates a long straight line by pushing two keys on a digitizing tablet's cursor. An automatic system fails to improve on a manual system if a user has to correct one end point, delete some fuzz on the line, or move the line. Systems for automatically digitizing line drawings must therefore have extremely low error rates if they are to require less human operator time than manual digitization.

Currently available digitization systems have not solved the automatic digitization problem. One significant problem is that thinning algorithms introduce undesirable artifacts.

In designing a new vectorization system, the work described in this thesis began with the idea that the successful recognition of a drawing requires knowledge about the domain in which the drawing was produced. Engineering diagrams are of a certain size and have certain features, including circles, rectangles, lines, arcs, and text. They can also be understood as having potentially been produced by being plotted from a CAD system; this fact limits the number of possibilities for what a given diagram might contain. As well as domain knowledge about engineering diagrams, another useful kind of knowledge to bring to bear on this problem is the understanding of the visual systems of animals, particularly the parts of the vision system that deal with the entrance of light into the eye and the progress of the signal through the various retinal cells and into the brain.

Using knowledge about vision and domain knowledge about engineering drawings, a vectorization system called VKV was designed and implemented.

The main advantages of system like VKV are:

- Because there is no thinning stage, there are no thinning artifacts;
- Vectorization occurs to a subpixel level of accuracy;

- Crossing and intersecting lines do not result in intersection artifacts;
- The system is easy to implement in a distributed environment; and
- The vectors it has produced have had fewer errors than other state of the art systems.

One of the key features of the VKV system is that it obtains results that have an accuracy well beyond the size of a pixel. This is very important for maps and engineering diagrams, since the initial input can be scanned at a much lower resolution and still achieve the same final accuracy. The scanned image that must be saved and manipulated also requires less time and space.

To provide something to compare with VKV, another system was implemented that combines the best features of classical vectorization methods. The two systems were compared using engineering type documents. The VKV system performed considerably better than the classical system.

This thesis has:

- shown that knowledge of animal vision is very helpful in designing a machine vision system;
- shown that line drawings cannot be interpreted completely correctly without knowledge about the domain of the drawings. Even then, although perfect interpretation is impossible in some cases, good interpretation is usually possible;
- developed a method, VKV, for solving the complex problem of document conversion. VKV works better than other current solutions on a wide class of drawings;
- provided a comprehensive survey of research pertaining to document conversion; and
- provided a model and evaluation methods for scanning and vectorization technology.

Many claims have been made that systems already developed have solved the automatic digitization problem, and many systems in the future will continue to make such claims. Hopefully this thesis has brought us closer to an automatic, intelligent understanding of line drawings.

1.0 Reference

[Arce:85] C. Arcelli and G.S. Di Baja. A width–independent fast thinning algorithm. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 7(4):463–474, July 1985.

[Alba:74] A. Albano. Representation of digitized contours in terms of conic arcs and straight–line segments. *Computer Graphics and Image Processing*, 3:23–33, 1974.

[ANAT:92] Intergraph Corporation, Huntsville, AL. 35894-4201. *ANA Tech Eagle 4080ET Scanner*, 1992.

[Arvi:84] K. Arvind. Extraction of lines and regions from grey tone line drawing images. Master's thesis, Computer Science Dept., Virginia Polytechnic Institute and State University, Blacksburg, VA., February 1984.

[Abdu:88] W.H. Abdulla, A.O.M. Saleh, and A.H. Morad. A preprocessing algorithm for hand–written character recognition. *Pattern Recognition Letters*, 7:13–18, 1988.

[Baru:88] O. Baruch. Line thinning by line following. *Pattern Recognition Letters*, 8:271–276, 1988.

[Boat:92] L. Boatto, V. Consorti, M. Del Buono, S. Di Zenzo, V. Eramo, A. Esposito, F. Melcarne, M. Meucci, A. Morelli, M. Mosciatti, S. Scarci, and M. Tucci. An interpretation system for land register maps. *Computer*, 25(7):25–33, July 1992.

[Blac:81] W. Black, T.P. Clement, J.F. Harris, B. Lewellyn, and G. Preston. A general purpose follower for line–structured data. *Pattern Recognition*, 14(1-6):33–42, 1981.

[Bhas:89] P. Bhaskaran and R. Flandrena. Artificial intelligence technologies in a drawing management system. In *NCGA '89 Conference Proceedings. 10th Annual Conference and Exposition Dedicated to Computer Graphics*, volume 1 of 3, pages 65–70. Nat. Computer Graphics Assoc., Philadelphia, PA, 17-20 Apr. 1989.

[Berg:90] R. Bergevin and M.D. Levine. Extraction of line drawing features for object recognition. In *Proceedings of the Tenth International Conference on Pattern Recognition*, volume 1 of 2, pages 496–501. IEEE Comput. Soc., Atlantic City, NJ, 16-21 June, 1990.

[Bono:88] P.R. Bono. Uses for CGM in raster–to–vector conversion. In A. Mumford and M. Skall, editors, *CGM in the Real World*, pages 113–143. Springer, Berlin, 1988.

[Bixl:85] J.P. Bixler and J.P. Sanford. A technique for encoding lines and regions in engineering drawings. *Pattern Recognition*, 18(5):367–377, 1985.

[Full:75] R. Buckminster–Fuller. *Synergetics*. Macmillan Publishing Company, New York, 1975.

[Bury:89] A.S. Bury. Raster to vector conversion: A methodology. In *GIS/LIS '89 Proceedings*, volume 1, pages 9–11. Amer. Soc. Photogrammetry and Remote Sensing, (Orlando, FL, 26-30 Nov., 1989).

[Bowy:83] A. Bowyer and J. Woodwark. *A Programmer's Geometry*. Butterworths, London, 1983.

[Bixl:88] J.P. Bixler, L.T. Watson, and J.P. Sanford. Spline–based recognition of straight lines and curves in engineering line drawings. *Image and Vision Comput.*, 6(4):262–269, November 1988.

[Cann:86] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8(6):679–698, November 1986.

[Cugi:84] U. Cugini, G. Ferri, P. Mussio, and M. Protti. Pattern–directed restoration and vectorization of digitized engineering drawings. *Computers & Graphics*, 8(4):337–350, 1984.

[Chen:89] Y-S. Chen and W-H. Hsu. An interpretive model of line continuation in human visual perception. *Pattern Recognition*, 22(5):619–639, 1989.

[Clem:81] T.P. Clement. The extraction of line–structured data from engineering drawings. *Pattern Recognition*, 14(1-6):43–52, 1981.

[Cora:87] A. Corana, M. Marchesi, C. Martin, and S. Ridella. Minimizing multimodal functions of continuous variables with the "Simulated Annealing" algorithm. *ACM Trans. on Mathematical Software*, 13(3):262–280, September 1987.

[Conn:90] S. Connelly and A. Rosenfeld. A pyramid algorithm for fast curve extraction. *Computer Vision, Graphics, and Image Processing*, 49:332–345, 1990.

[Core:79] S. Coren and L. Ward. *Sensation and Perception, 3rd ed.* Academic Press, New York, 1979.

[Dave:91] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[Deri:90] R. Deriche. Fast algorithms for low–level vision. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(1):78–87, January 1990.

[Deut:72] E.S. Deutsch. Thinning algorithms on rectangular, hexagonal, and triangular arrays. *Commun. ACM*, 15(9):827–837, September 1972.

[Donn:87] T. Donnelly and W.N. Martin. DRACAP: Drawing capture for electronic schematics. In *Proc. IEEE Computer Society Workshop on Computer Vision*, pages 274–276. IEEE, New York, 1987.

[Dori:89] D. Dori. A syntactic/geometric approach to recognition of dimensions in engineering machine drawings. *Computer Vision, Graphics, and Image Processing*, 47:271–291, 1989.

[Dori:92] D. Dori. Dimensioning analysis: Toward automatic understanding of engineering drawing. *Commun. ACM*, 35(10):92–103, October 1992.

[Espe:89] R. Espelid and J. Arild Eileng. A raster–to–vector conversion system producing high quality geometric entities. In *Proc. 6th Scandinavian Conference on Image Analysis*, volume 2 of 2, pages 1247–1253. Pattern Recognition Soc. Finland, Oulu, Finland, 19-22 June, 1989.

[Ejir:84] M. Ejiri, S. Kakumoto, T. Miyatake, S. Shimada, and H. Matsushima. Automatic recognition of design drawings and maps. In *Proc. Seventh International Conference on Pattern Recognition*, volume 2 of 2, pages 1296–1305. IEEE Comput. Soc., Silver Spring, MD, Montreal, PQ, 30 July - 1 Aug., 1984.

[Fole:82] J.D. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. The System Programming Series. Addison-Wesley Publishing Company, Reading, Massachusetts, 1882.

[Free:77] H. Freeman and L.S. Davis. A corner–finding algorithm for chain–coded curves. *IEEE Trans. Computers*, 26(1):297–303, January 1977.

[Fisc:87] M.A. Fischler and O. Firschein. *Intelligence: The Eye, the Brain, and the Computer*. Addison–Wesley, Reading, Mass., 1987.

[Fu:81] K.S. Fu and J.K. Mui. A survey on image segmentation. *Pattern Recognition*, 13:3–16, 1981.

[Free:61] H. Freeman. On the encoding of arbitrary geometric configurations. *IEEE Trans. Elec. Computers.*, 10:260–268, 1961.

[Free:74] H. Freeman. Computer processing of line–drawing images. *Computing Surveys*, 6(1):57–97, March 1974.

[Fulf:81] M.C. Fulford. The FASTRACK automatic digitizing system. *Pattern Recognition*, 14(1):65–74, 1981.

[Geor:90] R. George. Tracing a solution. *Cadence*, pages 117–119, November 1990.

[VNR:77] W. Gellert, H. Küstner, M. Hellwich, and H. Kästner, editors. *The VNR Concise Encyclopedia of Mathematics*. Van Nostrand Reinhold, New York, N.Y., 1977.

[Gibs:82] L. Gibson and D. Lucas. Vectorization of raster images using hierarchical methods. *Computer Vision, Graphics, and Image Processing*, 20:82–89, 1982.

[Greg:91] R.L. Gregory. Origins of eyes – with speculations on scanning eyes. In A.G. Leventhal, editor, *The Neural Basis of Visual Function*, volume 4 of *Vision and Visual Dysfunction*, chapter 2, pages 10–40. CRC Press, Inc, Boca Raton, FL, 1991.

[Grim:90] W. Eric L. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints*. MIT Press, Cambridge, Massachusetts, 1990.

[Govi:87] V.K. Govindan and A.P. Shivaprasad. A pattern adaptive thinning algorithm. *Pattern Recognition*, 20(6):623–637, 1987.

[Gonz:87] R.C. Gonzalez and P. Wintz. *Digital Image Processing*. Addison–Wesley, Reading, Mass., 2 edition, 1987.

[Gunt:90] O. Gunther and E. Wong. The arc tree: An approximation scheme to represent arbitrary curved shapes. *Computer Vision, Graphics, and Image Processing*, 51:313–337, 1990.

[Hild:69] C.J. Hilditch. Linear skeletons from square cupboards. In *Machine Intelligence IV*, pages 403–420. American Elsevier, 1969.

[Hosh:86] T. Hoshino, S. Suzuki, and M. Kosugi. Automatic input method for large–scale maps. In *Proc. Eighth International Conference on Pattern Recognition*, pages 449–453. IEEE Comput. Soc. Press, Washington, DC, Paris, France, 27-31 Oct., 1986.

[IGEO:92] Intergraph Corporation, Huntsville, AL. 35894-4201. *Mapping Sciences Applications (I/GEOVEC)*, 1992. DDGC186A0 6/91.

[IVEC:92] Intergraph Corporation, Huntsville, AL. 35894-4201. *Intergraph Vectorization Software (I/VEC)*, 1992. DTP025 B0 12/88.

[Jain:89] A.K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[John:90] R.B. Johnson and B.M. Bird. Schematic diagrams conversion to CAD files. In *Image Processing 90 – the Key Issues Conf. Proc.*, pages 1–10. Blenheim Online, London, UK, 9-11 Oct., 1990.

[Jain:80] A.K. Jain and C.R. Christensen. Digital processing of images in speckle noise. *Proc. SPIE*, 234:46–50, July 1980.

[Jenn:93] C. Jennings and N. Flanagan. Automatic GIS data capture and conversion. In *Proceedings of GIS Research – UK 1993*, Keele, England, 18-20 March, 1993.

[Jenn:93b] C. Jennings, N. Flanagan, and C. Flanagan. Automatic GIS data capture and conversion. In *Innovations in Graphical Information Systems*. Taylor and Francis, London, 1993.

[Jenn:93c] C. Jennings, N. Flanagan, and C. Flanagan. Gis data capture. In *Image Processing Conference 1993*, June 1993 University of Utrect. Poster presentation.

[Jans:84] H. Jansen and F-L. Krause. Interpretation of freehand drawings for mechanical design processes. *Computers & Graphics*, 8(4):351–369, 1984.

[Jimi:82] J. Jiminez and J.L. Navalon. Some experiments in image vectorization. *IBM J. Res. Develop.*, 26(6):724–734, November 1982.

[Jose:89] S.H. Joseph. Processing of engineering line drawings for automatic input to CAD. *Pattern Recognition*, 22(1):1–11, 1989.

[Jose:92] S.H. Joseph and T.P. Pridmore. Knowledge–directed interpretation of mechanical engineering drawings. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14(9):928–940, September 1992.

[Jenn:93a] C. Jennings, J.R. Parker, and D. Molaro. A force–based thinning strategy with sub–pixel precision. Technical Report 93/504/09, University of Calgary, Department of Computer Science, 2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4, February 1993.

[Jenn:93d] C. Jennings, J.R. Parker, and D. Molaro. Comparative performances of hpc systems for signal processing. In *Proceedings of SS '93 High Performance Computing*, Calgary, Alta., 6-9 Mar. 1993.

[Kapl:91] E. Kaplan. The receptive field structure of retinal ganglion cells in cat and monkey. In A.G. Leventhal, editor, *The Neural Basis of Visual Function*, volume 4 of *Vision and Visual Dysfunction*, chapter 2, pages 10–40. CRC Press, Inc, Boca Raton, FL, 1991.

[Kast:90] R. Kasturi, S.T. Bow, W. El-Masri, J. Shah, J.R. Gattiker, and U.B. Mokate. A system for interpretation of line drawings. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(10):978–992, 1990.

[Kikk:84] W. Kikkawa, M. Kitayama, K. Miyazaki, H. Arai, and S. Arato. Automatic digitizing system for PWB drawings. In *Proc. Seventh International Conference on Pattern Recognition*, volume 2 of 2, pages 1306–1309. IEEE Comput. Soc., Silver Spring, MD, Montreal, PQ, 30 July - 1 Aug., 1984.

[Kaku:83] S. Kakumoto, T. Miyatake, S. Shimada, and M. Ejiri. Development of auto–digitizer using multi–color drawings recognition method. In *Proc. IEEE Computer Vision and Pattern Recognition Conference 1983*, pages 508–509. IEEE Computer Soc. Press, 1983.

[Kuff:76] S. Kuffler, J. Nichols, and A. Martin. *From Neuron to Brain, 2nd ed.* Sinauer Assoc., Sunderland, Mass., 1976.

[Kong:89] T.Y. Kong. Digital topology: Introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48:357–393, 1989.

[Kuff:53] S.W. Kuffler. Discharge patterns and functional organization of mammalian retina. *Journal of Neurophysiology*, 16:37–68, 1953.

[Kwok:88] P.C.K. Kwok. A thinning algorithm by contour generation. *Commun. ACM*, 31(11):1314–1324, November 1988.

[Land:87] U.M. Landau. Estimation of a circular arc center and its radius. *Computer Vision, Graphics, and Image Processing*, 38:317–326, 1987.

[Land:91] M.F. Land. Optics of the eyes of the animal kingdom. In J.R Cronly-Dillon and R.L. Gregory, editors, *Evolution of the Eye and Visual System*, volume 2 of *Vision and Visual Dysfunction*, chapter 6, pages 118–135. CRC Press, Inc, Boca Raton, FL, 1991.

[Leu:88] J-G. Leu and L. Chen. Polygonal approximation of 2–D shapes through boundary merging. *Pattern Recognition Letters*, 7:231–238, 1988.

[Lind:88] M. Lindenbaum and J. Koplowitz. Compression of chain codes using digital straight line sequences. *Pattern Recognition Letters*, 7:167–171, 1988.

[Lam:92] L. Lam, S. Lee, and C. Suen. Thinning methodologies – A comprehensive survey. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14(9), September 1992.

[Lim:80] J.S. Lim and H. Nawab. Techniques for speckle noise removal. *Proc. SPIE*, 234:35–44, July 1980.

[Lowe:85] D.G. Lowe. *Perceptual Organization and Visual Recognition*. Kluwer Academic Publishers, Boston, 1985.

[Lowe:87] D.G. Lowe. Three–dimensional object recognition from single two–dimensional images. *Artificial Intelligence*, 31:355–395, 1987.

[Lu:91] S.W. Lu, Y. Ren, and C.Y. Suen. Hierarchical attributed graph representation and recognition of handwritten chinese characters. *Pattern Recognition*, 24(7):617–632, 1991.

[Li:91] B. Li and C.Y. Suen. A knowledge–based thinning algorithm. *Pattern Recognition*, 24(12):1211–1221, 1991.

[Lee:90] K-J. Lee, Y. Shirai, and T.L. Kunii. Attribute–grammar based approach to vector extraction from a raster image. In T.S. Chua and T.L. Kunii, editors, *CG International '90: Computer Graphics Around the World*, pages 225–239. Springer–Verlag, Tokyo, 1990.

[Mack:83] A.K. Mackworth. Constraints, descriptions and domain mappings in computational vision. In O.J. Braddick and A.C. Sleigh, editors, *Physical and Biological Processing of Images*, pages 33–40. Springer-Verlag, Berlin, 1983.

[Macl:91] S. Maclean. Raster editor merges manual drawings and CAD: Taming the two headed AEC monster with technology. *CAD Systems*, page 10, November 1991.

[Marr:82] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W.H. Freeman, New York, 1982.

[Meke:92] Mekel Engineering Inc. *Operating Instructions and Maintenance Manual for the Mekel M460 Microfiche Digitizer*, January 1992.

[Mero:81] L. Mero. An optimal line following algorithm. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 3(5):593–598, September 1981.

[Marr:80] D. Marr and E. Hildreth. Theory of edge detection. *Proc. R. Soc. Lond.*, 207:187–217, 1980.

[Mola:93b] D. Molaro and C. Jennings. A methodology for writing multiprocessor programs on a network of unix workstations. In *Proceedings of SS '93 High Performance Computing*, Calgary, Alta., 6-9 Mar. 1993.

[Mola:93] D. Molaro, C. Jennings, and J.R. Parker. Distributed force–based thinning and a general distribution method. Technical Report 93/505/10, University of Calgary, Department of Computer Science, 2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4, February 1993.

[Mont:69] U. Montanari. Continuous skeletons from digitized images. *Journal of the ACM*, 16(4):534–549, October 1969.

[Mont:70] U. Montanari. A note on minimal length polygonal approximation to a digitized contour. *Commun. ACM*, 13(1):41–47, January 1970.

[Morr:91] J.C. Morrison. Distance from a point to a line. In J. Arvo, editor, *Graphics GEMS II*, chapter 1, pages 10–13. Academic Press, Inc., 1250 Sixth Ave., San Diego, CA 92101, 1991.

[Musa:88] M.T. Musavi, M.V. Shirvaikar, E. Ramanathan, and A.R. Nekovei. A vision based method to automate map processing. *Pattern Recognition*, 21(4):319–326, 1988.

[Mane:90] A. Manesh, J. Wrobel, and J. Gao. Automatic vectorization of scanned engineering drawings. In *Proc. of the 1990 Symposium on Applied Computing*, pages 320–324. IEEE Comput. Soc., Lafayetteville, AR, 5-6 Apr., 1990.

[Naka:84] A. Nakamura and K. Aizawa. Digital circles. *Computer Vision, Graphics, and Image Processing*, 26:242–255, 1984.

[Nadl:84] M. Nadler. Document segmentation and coding techniques. *Computer Vision, Graphics, and Image Processing*, 28:240–262, 1984.

[Nish:89] T. Nishimura and T. Fujimoto. Fast contour line extraction algorithm observing line continuation. *SPIE Visual Communications and Image Processing IV*, 1199:704–711, 1989.

[Niem:90] H. Niemann. *Pattern Analysis and Understanding*, volume 4 of *Springer Series in Information Sciences*. Springer–Verlag, Berlin, 2 edition, 1990.

[Naga:88] V. Nagasamy and N.A. Langrana. Automated restoration of engineering drawings into a CAD data base. *Engineering with Computers*, 4:165–171, 1988.

[Naga:90] V. Nagasamy and N.A. Langrana. Engineering drawing processing and vectorization system. *Computer Vision, Graphics, and Image Processing*, 49(3):379–397, March 1990.

[Nacc:84] N.J. Naccache and R. Shinghal. SPTA: A proposed algorithm for thinning binary patterns. *IEEE Trans. Systems, Man, and Cybernetics*, 14(3):409–418, May/June 1984.

[OGor:90] L. O'Gorman. K x K thinning. *Computer Vision, Graphics, and Image Processing*, 51:195–215, 1990.

[Park:91] J.R. Parker. Gray level thresholding in badly illuminated images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(8):813–819, August 1991.

[Pave:79] M. Pavel. Skeletal categories. *Pattern Recognition*, 11:325–327, 1979.

[Pavl:82b] T. Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, Rockville, MD, 1982.

[Pavl:82] T. Pavlidis. An asynchronous thinning algorithm. *Computer Vision, Graphics, and Image Processing*, 20:133–157, 1982.

[Pavl:86] T. Pavlidis. A vectorizer and feature extractor for document recognition. *Computer Vision, Graphics, and Image Processing*, 35:111–127, 1986.

[Prin:90] J. Princen, J. Illingworth, and J. Kittler. A hierarchical approach to line extraction based on the Hough transform. *Computer Vision, Graphics, and Image Processing*, 52:57–77, 1990.

[Park:92] J.R. Parker and C. Jennings. Defining the digital skeleton. In *Proc. SPIE Vision Geometry*, volume 1832, pages 224–234, Boston, Massachusetts, 15-16 November, 1992.

[Park:93t] J.R. Parker, C. Jennings, and A.G. Salkauskas. Thresholding using an illumination model. Technical Report 93/506/11, University of Calgary, Department of Computer Science, 2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4, February 1993.

[Park:93] J.R. Parker, C. Jennings, and A.G. Salkauskas. Thresholding using an illumination model. In *Proceedings of Second International Conference on Document Analysis and Recognition*, Japan, Oct 20-22, 1993.

[Pale:83] K. Paler and J. Kittler. Graylevel edge thinning: A new method. *Pattern Recognition Letters*, 1:409–416, 1983.

[Piwo:90] J.M. Piwowar, E.F. LeDrew, and D.J. Dudycha. Integration of spatial data in vector and raster formats in a geographic information system environment. *International Journal of Geographical Information Systems*, 4(4):429–444, 1990.

[Powl:92] J. Powlesland. Autotracing with Adobe Streamline 2.0. *The Micro Byte*, 5(2), May 1992.

[Pare:89] P. Parent and S.W. Zucker. Trace inference, curvature consistency, and curve detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 11(8):823–839, August 1989.

[QC:87] QC Data. Mews map editing work station. *QC Data News*, 1(1), 1987. This is a company internal publication used largely for marking by QC–Data. It was obtained from: QC Data, 3838 N. Belt East, Houston, TX 77032.

[Rame:72] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1:244–256, 1972.

[Rama:80] K. Ramachandran. Coding method for vector representation of engineering drawings. *Proc. IEEE*, 68(7):813–817, July 1980.

[Sund:86] P.A. Sundar Raj and J. Koplowitz. On bit reduction of chain coded line drawings. *Pattern Recognition Letters*, 4:99–102, 1986.

[Rodi:65] R.W. Rodieck. Quantitative analysis of cat retinal ganglion cell response to visual stimuli. *Vision Research*, 5:583–601, 1965.

[Roge:85] D.F. Rogers. *Procedural Elements for Computer Graphics*. McGraw–Hill Book Company, New York, 1985.

[Russ:90] J.C. Russ. *Computer–Assisted Microscopy: The Measurement and Analysis of Images*. Plenum Press, New York, N.Y., 1990.

[RVCS:90] DataSpan Technology Inc., Calgary, Alberta. *RVCS Users Manual*, 1990.

[Rosi:89] P.L. Rosin and G.A.W. West. Segmentation of edges into lines and arcs. *Image and Vision Comput.*, 7(2):109–114, May 1989.

[SAMI:92] Universal Systems Ltd., Fredericton, New Brunswick, Canada. *SAMI Product Information*, 1992.

[Seku:85] R. Sekuler and R. Blake. *Perception*. Knopf, New York, N.Y., 1985.

[Sark:91] S. Sarkar and K.L. Boyer. On optimal infinite impulse response edge detection filters. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(11):1154–1171, November 1991.

[Shen:92] J. Shen and S. Castan. An optimal linear operator for step edge detection. *CVGIP: Graphical Models and Image Processing*, 54(2):112–133, 1992.

[Schu:88] R.D. Schulman. Intelligent recognition – key to productivity. In *Electronic Imaging '88: International Electronic Imaging Exposition and Conference. Advance Printing of Paper Summaries.*, volume 2, pages 825–826. Inst. Graphic Commun., Waltham, MA, 3-6 Oct., 1988.

[Suet:81] P. Suetens, P. Dierch, R. Piessens, and A. Osterlinck. A semiautomatic digitization method and the use of spline functions in processing line drawings. *Computer Graphics and Image Processing*, 15:390–400, 1981.

[Skla:80] J. Sklansky and V. Gonzalez. Fast polygonal approximation of digitized curves. *Pattern Recognition*, 12:327–331, August 1980.

[Sinh:87] R.M.K. Sinha. A width–independent algorithm for character skeleton estimation. *Computer Vision, Graphics, and Image Processing*, 40:388–397, 1987.

[Shih:89] C. Shih and R. Kasturi. Extraction of graphic primitives from images of paper based line drawings. *Machine Vision and Applications*, 2(2):103–113, 1989.

[Srir:89] R. Sriraman, J. Koplowitz, and S. Mohan. Tree searched chain coding for subpixel reconstruction of planar curves. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 11(1):95–104, January 1989.

[Smit:87] R.W. Smith. Computer processing of line images: A survey. *Pattern Recognition*, 20(1):7–15, 1987.

[Shim:88] S. Shimotsuji, A. Okazaki, O. Hori, and S. Tsunekawa. A high–speed raster–to–vector conversion using special hardware for contour tracking. In *Proc. of IAPR Workshop on Computer Vision: Special Hardware and Industrial Applications*, pages 18–23. Univ. Tokyo, 1988.

[SVR:90] Scorpion Technologies. *SRV Training Outline*.

[Sche:82] A. Scher, M. Shneier, and A. Rosenfield. Clustering of collinear line segments. *Pattern Recognition*, 15(2):85–91, 1982.

[Saho:88] P.K. Sahoo, S. Soltani, and A.K.C. Wong. A survey of thresholding techniques. *Computer Vision, Graphics, and Image Processing*, 41:233–260, 1988.

[Saka:89] S. Sakashita and Y. Tanaka. Computer–aided drawing conversion (an interactive approach to digitize maps). In *GIS/LIS '89 Proceedings*, volume 2, pages 578–590. Amer. Soc. Photogrammetry and Remote Sensing, Orlando, FL, 26-30 Nov., 1989.

[Stev:87] D.B. Stevens. Line detection in blueprint analysis. Master's thesis, Dept. of Electrical Engineering, U of Virginia, Blacksburg, VA., 1987.

[Sea:90] Seaports and the Shipping World. New document management system to help maintain 22,000 canadian navy frigate engineering drawing sheets installed at Saint John Shipbuilding. *Seaports and the Shipping World*, page 45, December 1990.

[Sugi:86] Kokichi Sugihara. *Machine Interpretation of Line Drawings*. MIT Press, Cambridge, Massachusetts, 1986.

[Suzu:90] S. Suzuki and T. Yamada. MARIS: Map recognition input system. *Pattern Recognition*, 23(8):919–933, 1990.

[Thom:89] S.M. Thomas and Y.T. Chan. A simple approach for the estimation of circular arc center and its radius. *Computer Vision, Graphics, and Image Processing*, 45:362–370, 1989.

[Taxt:89] T. Taxt, P.J. Flynn, and A.K. Jain. Segmentation of document images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 11(12):1322–1329, December 1989.

[Ande:86] The Anderson Report. *Automatic Digitizers – A Special Report*, October 1986.

[Udup:75] K.J. Udupa and I.S.N. Murthy. Some new concepts for encoding line patterns. *Pattern Recognition*, 7:225–233, 1975.

[VECT:92] Winchester Data Products Inc., Raleigh, NC 27609. *VECTRESS – Give Your CAD System Vision*, 1992.

[Vaxi:92] P. Vaxiviere and K. Tombre. Celesstin: CAD conversion of mechanical drawings. *Computer*, 25(7):46–54, July 1992.

[Wats:84] L.T. Watson, K. Arvind, R.W. Ehrich, and R.M. Haralick. Extraction of lines and regions from grey tone line drawing images. *Pattern Recognition*, 17(5):493–507, 1984.

[Waka:82] T. Wakayama. A core–line tracing algorithm based on maximal square moving. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 4(1):68–74, January 1982.

[Wall:84] K. Wall and P-E. Danielsson. A fast sequential method for polynomial approximation of digitized curves. *Computer Vision, Graphics, and Image Processing*, 28:220–227, 1984.

[Yosh:83] Y. Yoshino, K. Mori, A. Okazaki, and S. Tsunekawa. Flexible drawing reader with high–speed hierarchical processors. In *Proc. IEEE Computer Vision and Pattern Recognition Conference 1983*, pages 510–514. IEEE Computer Soc. Press, 1983.

[Yu:90] S-S. Yu and W-H. Tsai. A new thinning algorithm for gray–scale images by the relaxation technique. *Pattern Recognition*, 23(10):1067–1076, 1990.

[Yama:91] H. Yamada, K. Yamamoto, T. Saito, and S. Matsui. Map: Multi–angled parallelism for feature extraction from topographical maps. *Pattern Recognition*, 24(6):479–488, 1991.

[Zhan:84] T.Y. Zhang and C.Y. Suen. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27(3):236–239, March 1984.

Index