

USING A DISK OPERATOR TO CONVERT RASTER
IMAGES OF ENGINEERING DRAWINGS TO
VECTOR IMAGES

By

REDDY V. V. S. MALLIDI

Bachelor of Engineering
Andhra University
Waltair, India
1985

Master of Technology
Indian Institute of Technology
Kharagpur, India
1990

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1992

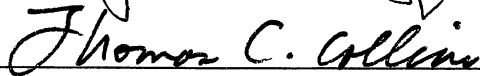
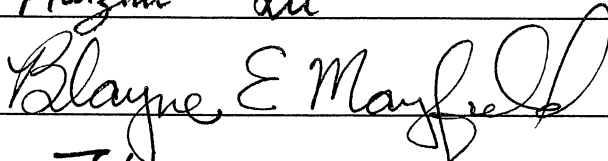
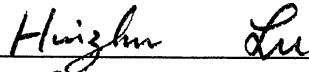
Shelby
1/1/14
RISSE

USING A DISK OPERATOR TO CONVERT RASTER
IMAGES OF ENGINEERING DRAWINGS TO
VECTOR IMAGES

Thesis Approved:



Thesis Adviser



Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude and appreciation to my main adviser, Dr. William D. Miller for his invaluable guidance, encouragement, availability and patience. I am grateful to my other committee members Dr. Blayne Mayfield and Dr. H. Lu for their encouragement and inspiration throughout my project.

Special thanks are due to Mr. Robert Webster, and Mr. Bruce Taylor of American Small Business Computers, Pryor, for providing financial support and facilities for research and for their constant encouragement.

I also wish to thank Dr. Frank Collins, Department of Psychology, for his encouragement in all my efforts.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.	1
General Statement of the Problem.	1
II. LITERATURE REVIEW	3
Introduction.	3
Thresholding Methods.	3
Edge Fitting Methods.	7
Linking of Edge Elements.	9
III. GENERAL APPROACH TO THE PROBLEM	13
Introduction.	13
Compression of Bilevel Images	13
Locating Edges/Lines.	16
Conversion to Vectors	22
Storing the Vectors in a File	24
IV. IMPLEMENTATION DETAILS.	25
Introduction.	25
Scanning Algorithm.	25
Linking Algorithm	28
V. RESULTS AND CONCLUSIONS	31
Introduction.	31
Results and Conclusions	32
Recommendations for Further Study	34
BIBLIOGRAPHY	35
APPENDIXES	39
APPENDIX A - TAG IMAGE FILE FORMAT.	40
APPENDIX B - DESIGN CAD FILE FORMAT	50
APPENDIX C - SCANNING ALGORITHM CODE.	54

LIST OF FIGURES

Figure	Page
1. TIFF File Header.	15
2. Hueckel Edge Operator Disk.	17
3. Approximation of Circular and Discrete Disks. .	19
4. Frequencies of H_i	21

CHAPTER I

INTRODUCTION

General Statement of the Problem

As more and more business and technical information is stored and manipulated electronically, there is an increasing gap between the accessibility of this information and older records stored on paper. Accordingly, finding ways to scan old documents and store them efficiently is becoming more important. In particular, millions of old engineering drawings describing objects or products that are still in use need to be scanned in enterprises all over the world.

A typical digitizing device produces the raster scan of a document, so that no structure beyond individual brightness variation is directly recorded. This representation not only omits some of the information which actually interests the user, but it is also far more bulky than a representation in terms of vectors or other more meaningful objects. This is especially true for engineering drawings, which typically consist of figures depicted with narrow lines, surrounded by blank space. Thus, converting line drawings into vector form which could be represented by a standard CAD system would bring many benefits. Some

software which does this is currently available, but its performance is not completely satisfactory, since it needs considerable effort from the user.

Therefore, the purpose of this study is to investigate conversion of line drawings to vector form, using a mathematically sophisticated edge/line detection algorithm proposed by Manfred H. Hueckel. The aim is to produce a scan conversion program which is fully automatic, or at least requires much less operator effort than anything presently available.

CHAPTER II

LITERATURE REVIEW

Introduction

The most important aspect of conversion of raster images to vectors is the recognition of edges in scenes. Edges are primitive features of an image that are widely used to outline the boundaries of objects [ABDQDA79]. They are the input to higher level routines which assemble them into vectors. There are two approaches to object recognition

- i) Thresholding - In this method discontinuities in an image are enhanced by some operator. If the discontinuity is greater than a threshold value an edge is deemed to be present [ABDQDA79].
- ii) Edge Fitting - It is the process of finding the boundary between two regions of different light intensities, or some other image attribute in a picture by fitting some ideal function and determining whether the match is good in terms of the mean square method.

Thresholding Methods

Abdou and Pratt [ABDQDA79] described two techniques for the design and evaluation of edge detectors using the

thresholding technique. In one approach, edge detection is considered as a statistical decision process, and in the other, edge detection is considered as a deterministic pattern recognition and classification task. They have shown that both these methods gave consistent results. They concluded that

- i) template edge detectors require more operations than differential edge detectors, and
- ii) Prewitt and Sobel differential edge detectors are the best of the class of 3x3 pixel differential edge operators.

Template matching is one of the most widely used approaches. A pattern is compared with stored models of known patterns, also known as edge masks, and the best match is chosen.

Prewitt [PREOEA70] described a template matching technique using eight edge masks. In this approach, a match with each of the eight masks is calculated at each of the pixels in the image, and the edge magnitude is determined by the mask with the highest output.

Rosenfeld and Thurston [ROSEAC71] conducted several experiments to find out the optimal size of the edge masks.

Template matching has the following drawbacks [DAVCOT86]:

- i) Different masks are needed for the accurate estimation of edge magnitude and orientation.

- ii) Optimization of noise suppression imposes additional conditions on template matching masks.

Griffith [GRIEDI73] described a method of edge detection in simple scenes using a priori information. An initial investigation is made on an entire scene with just enough effort per unit area to find the most distinct edge lines. Suppose that knowing the positions of these lines allows us to single out 1/10th of area of the field as constituting the only regions in which more subtle lines could lie. We concentrate on these areas applying the operator more densely to search for previously undetected edges. Griffith employed a five stage line detection system.

- 1) Feature point extraction based on intensity values of pixels.
- 2) Line extraction from the feature points
- 3) Heuristic approach to join the lines.
- 4) Heuristic approach to predict the locations of missing lines.
- 5) Verification of the existence of proposed edge lines.

The main drawbacks of Griffith's operator are [DAVAS075] as follows:

- 1) It is not clear if the analysis can be extended to objects that are inherently noisy or to textured objects.
- 2) Extension to include curved surfaces is not clear.
- 3) The operator ignores regions with edge or line not

centered, and if such regions are not ignored it results in a very high (70%) false positive error rate, i.e., the operator finds an edge, where in fact there is none.

Chow [CHOABD72] described an edge detection method for histograms of cineangiograms. An image is divided into a regular array of overlapping subimages and individual histograms are constructed for each one. A threshold value is selected to construct a binary picture from a gray image. Disadvantages attributed to this method are that it involves a lot of computation [DAVMVT90], there is no good global threshold value, and it is not suited to complex pictures [DAVAS075].

Nevatia and Babu [NEVLFE80] described an edge detection approach to find out edges in aerial pictures by using edge masks followed by thinning and thresholding the edge magnitudes. Then the linked elements are approximated by piecewise linear segments.

Canny [CANACA86] described a procedure for the detection of edges using a computational approach. He proposed a detector which used an adaptive thresholding technique with hysteresis to eliminate streaking of edge contours.

Davies [DAVMVT90] observes that the thresholding approach is error prone, or else quite difficult to implement in practice for real images.

Edge Fitting Methods

Hueckel [HUEAOW71] used an analytical approach to edge detection. His approach, described here briefly, is explained in detail in the next chapter. Define the difference N , between the neighborhood of an image and the ideal step by

$$N^2 = \sum_{x \in R} (A(x) - S(x, \epsilon))^2$$

where $A(x)$ - image intensity at point x , and

$S(x, \epsilon)$ - intensity of the ideal step function for
a chosen vector of parameter ϵ .

The problem is reduced to finding minimum N^2 on an area R of a circular disk. Hueckel approximated $A(x)$ and $S(x, \epsilon)$ by a radial Fourier series where only the first nine functions are used. So,

$$N'^2 = \sum_{i=0}^8 (a_i - s_i(\epsilon))^2$$

where a_i , and s_i are coefficients of expansion of Fourier series. Hueckel obtained a solution for ϵ , and having calculated N' , the presence of an edge is determined by requiring the step height h to be large and N' to be small. Hueckel [HUEALV73] proposed a modified operator to detect edges and lines.

Nevatia [NEVEOA77] tried to simplify the Hueckel edge operator using a fewer terms for approximating a step. But, he found that there was a performance loss when noise was present.

Nevatia [NEVACE77], extended the Hueckel edge operator to color images. He gave three alternative definitions of a color edge. Nevatia defined the step in color as having three components, and let the degrees of fit for the three be N_1 , N_2 , and N_3 , and

$$N_1^2 = \sum_{i=0}^8 (a_{1i} - s_{1i})^2$$

where a_{1i} , and s_{1i} are the terms of expansion for the signal and the step for the first component of a color image.

Similarly N_2 , and N_3 are defined. Now, an optimal step in color space is defined by determining the three components such that

$$N^2 = N_1^2 + N_2^2 + N_3^2$$

is minimum.

Kundu [KUNRED90] proposed another edge detection method based on a statistical classification technique. It uses two characteristics of natural edges for their detection:

- i) near and around the step and linear edges the pixels, when classified into two nearly equal groups, display markedly different average intensity values, and
- ii) members of each group show strong spatial correlation.

This approach combines thresholding and spatial correlation. The edge detector described tries to locate the edges at points where these two conditions are satisfied with strong statistical evidence. An edge operator is described in 3 phases. In phase I step and linear edges are detected, in

phase II thin lines are detected, and in phase III spurious and missing edges are treated. Kundu says that this operator is computationally simple.

Stein and Medioni [STEETD90] described a different approach to recognize two-dimensional objects without resorting to thresholding or edge fitting. Their representation of a model or scene is based on polygonal approximation wherein curves are approximated by several polygons with different line fitting tolerances. Groups of consecutive segments are gray coded and entered into a hash table. Then recognition of objects proceeds by segmenting the scene into a polygonal approximation. The models for matching objects are created by random overlapping of more than 3 and less than 7 random triangles.

Gökmen and Li [GOKEDW90] presented an edge detection algorithm using regularization theory. In the algorithm, the energy functional in the standard regularization has been modified to spatially control the smoothness over the image to locate the edges accurately. The authors say that their algorithm smoothens noise without degrading discontinuities and offers an efficient alternative to the existing edge detectors for edge detection and surface reconstruction.

Linking of Edge Elements

Once the edges in an image are detected by using any one of the several edge operators available, it becomes

necessary to connect the edge elements properly to form object boundaries. This can be difficult, because many of the edge detection operators may generate edges that may not correspond to the actual boundaries of an object.

There are a number of schemes available for relinking broken boundary elements.

A simple technique as described by Nevatia [NEVMP82] is to link points in a neighborhood of, say 8 pixels, if they have similar or same orientation.

The Hough transform [HOUMAM62] is one of the techniques of edge element aggregation. The theory of Hough Transform technique which became widely known due to Duda and Hart [DUDUOH72] is as follows. The general equation of a straight line can be written as

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

where θ is the angle made by normal to the line with the x axis and Γ is the length of this normal. For any point (x_i, y_i) on this line the above equation becomes

$$x_i \cos \theta + y_i \sin \theta = \Gamma$$

This equation corresponds to a sinusoidal curve in (Γ, θ) space. This relationship between the image plane and the (Γ, θ) plane is called Hough transformation. The curves corresponding to collinear figure points intersect at a point. This point defines the line passing through the collinear points in the (Γ, θ) plane. Now, the detection of a group of collinear points reduces to constructing the

transform curves in the (Γ, θ) plane for each point and picking the points where three or more such curves coincide. In actual practice, Γ is limited to the size of the image plane and $\theta \in [0, 2\pi]$ [DUDUOH72].

Nevatia observes that the Hough transform technique does not examine the proximity of the clustered points and hence groups of contiguous point must be separated from a cluster of collinear points [NEVMP82].

Nevatia [NEVLOB76] described a six step procedure to link the edge elements detected by an operator. Each edge element is assumed to have a position and a direction associated with it. The entire 360 range of directions is divided into a number equiangular intervals (say, 12). Then, linking in an interval is limited to those edges which fall into that interval. Each image is divided into strips/buckets of some fixed size, (say, 3 pixels) and edges in each strip are linked together. If two consecutive edge elements in a bucket differ in y coordinate by a small distance of a threshold value, say 2 pixels, then those two elements belong to the same segment. After linking elements in a bucket, segments in adjacent buckets are linked without resulting in a change of orientation of the segment. Finally, segments which exceed a fixed number of pixels in length are retained.

Davies [DAVMVT90] observes that noise spurs around

boundaries can be eliminated efficiently by removing lines that are shorter than three pixels.

Canny's method [CANACA86] of thresholding is a more computationally efficient method. In his method, intensity gradients above some upper threshold value are taken to indicate an edge in the boundary, and gradients above some lower threshold value are taken to indicate an edge only if they are adjacent to positions that have already been accepted as edges. In this approach, edge detection and linking of edges proceed together.

Marr and Hildreth [MARTOE80] presented an edge detector which is computationally complex in nature. Their theory is based on two main ideas. First, one simplifies the detection of intensity changes by dealing with the image separately at different resolutions. The detection process then can be based on finding zero-crossings in a second derivative operator. Marr and Hildreth used Laplacian operator. Next step of combining the information obtained in the first step is based on their assumption that coincidence of zero crossings sufficient evidence for the existence of a physical edge. Like Canny's approach, this method also tries to combine edge detection and aggregation of edge elements.

Martelli [MARAA076] presented a method for detecting edges and finding contours in noisy pictures. In his method of edge detection the author treated the optimization problem as a shortest path problem on a graph.

CHAPTER III

GENERAL APPROACH TO THE PROBLEM

Introduction

The problem of converting raster images to vector images is divided into four major parts

- i) Decompression of the image data (if the image is compressed).
- ii) Locating edges/lines in an image using an edge detection operator.
- iii) Properly linking the edge elements found by the operator.
- iv) Storing the vector data in a file for subsequent use.

Compression of Bilevel Images

There have been some attempts on the part of scanner vendors and some software developers for the last few years to standardize the format of an image file. One of the widely accepted format is Tag Image File Format (TIFF). The structure of the TIFF file is briefly described below [ACTIF90].

The TIFF file begins with an 8 byte header that points

to one or more image file directories. The image file directories contain information about the images, as well as pointers to the actual image data as shown in fig. 1.

First word in the image file header specifies the byte order used in the file - Big Endian, or Little Endian. In Big Endian byte order, the address of the most significant byte is placed at the lowest address, and vice versa in Little Endian byte order. The second word gives the version number. It is useful in checking if the file is a TIFF file and if nothing has changed. Next long word (bytes 4-7) contains the offset of the image file directory.

The Image file directory consists of a 2-byte count of the number of fields, followed by a sequence of 12-byte field entries, followed by a 4-byte offset of the next image file directory if it exists. Details of the directory entry are as shown in fig. 1.

For bilevel images, there are two compression schemes available to compress data.

- i) CCITT Group 3 1-dimensional Modified Huffman run length encoding [ACTIF90].
- ii) PackBits compression, a simple byte oriented run length scheme [ACTIF90].

Decompression schemes for these two are described in [ACTIF90].

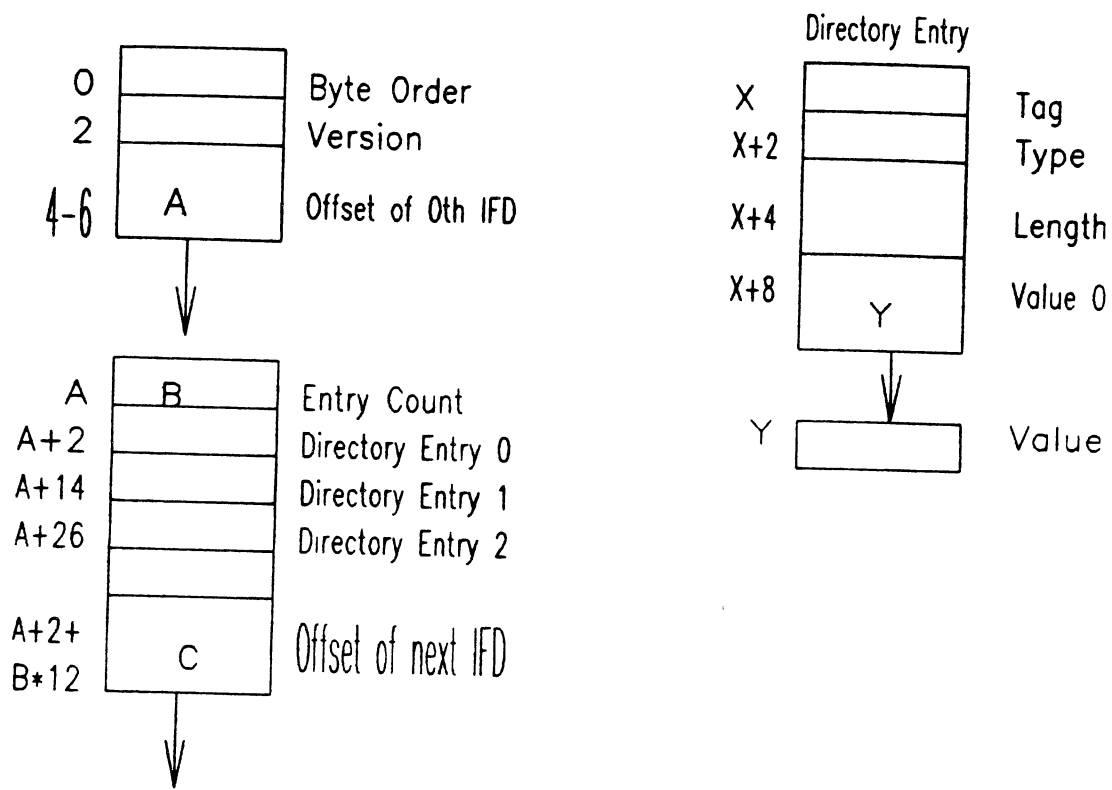


Figure 1. TIFF File Header

Locating Edges/Lines

The following factors are considered before choosing an edge detector.

- i) Since the type of images intended to be processed are confined to engineering drawings, pictures are less likely to be noisy.
- ii) The edge detector must be able to locate edges and lines properly in a bilevel image by averaging out pixels on edges and lines in a region.
- iii) Since the intended images are bilevel as opposed to grayscale, thresholding techniques are not considered.

Under these assumptions, the Hueckel edge detection algorithm is best suited for the present purposes to find edges and lines in an image [HUEALV73]. The theory of the Hueckel edge operator is described here.

Define b_- and b_+ as the brightness values of the two regions as shown in fig. 2, t_- , and t_+ as the step values of brightness, and r_- and r_+ as the radial distances to the two regions as shown in fig. 2. Let $c_x = \cos \alpha$ and $c_y = \sin \alpha$. Define x, y as the Cartesian coordinates in the picture plane. Let $S(x, y, c_x, c_y, r_-, r_+, t_-, t_+, b_-)$ be the function defining a pattern, edge, line, or edge-line.

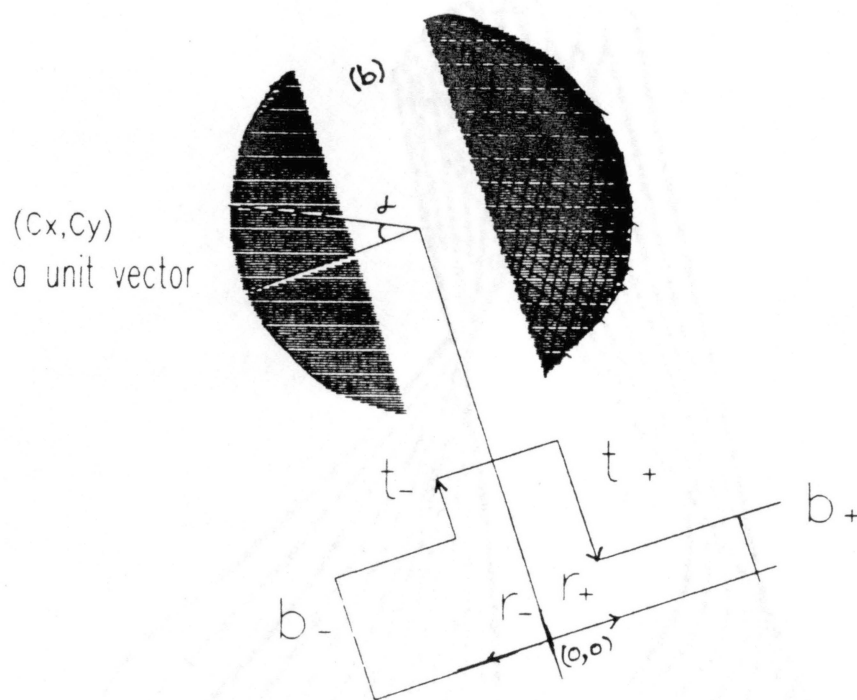


Figure 2. Hueckel Edge Operator Disk

Now, edge and line are defined in terms of the edge-line region (b) in fig. 2. An edge-line is pure edge if $r_- = r_+$ or $t_- = 0$ or $t_+ = 0$ and a pure line if $t_- = t_+$. Therefore, this operator offers a choice of three different modes of pattern recognition.

Define D to be a set of grid squares to best approximate a circular disk. Assume D to be a subgrid of a digitized picture. Define the x, y coordinate system so that

$$D' = \{(x, y) \mid x^2 + y^2 \leq 1\} \text{ to best approximate } D$$

$$I(x, y) = \text{intensity value at } (x, y)$$

$$I = \text{the function which is given by } I(x, y) \text{ over } D$$

The disk to be applied over the image is assumed to be continuous, whereas D is not continuous in actual practice. If a continuous disk and a discrete disk are superimposed then some squares in disk D fall outside D' and some squares outside of D have corners which fall inside D' as shown in fig. 3. To minimize this error due to transition from a discrete disk to a continuous disk, some particular disk sizes are chosen [HUEAOW71]. From an investigation it is found that disks with 32, 52, 69, 88, 137 grid squares are the most favorable ones. The corresponding radii of D' are given by

$$r = (\text{area of } D/\pi)^{\frac{1}{2}}$$

The error due to the approximation of the function I by S given by

$$I - S = \left(\int_{D'} (I(x, y) - S(x, y))^2 dx dy \right)^{\frac{1}{2}}$$

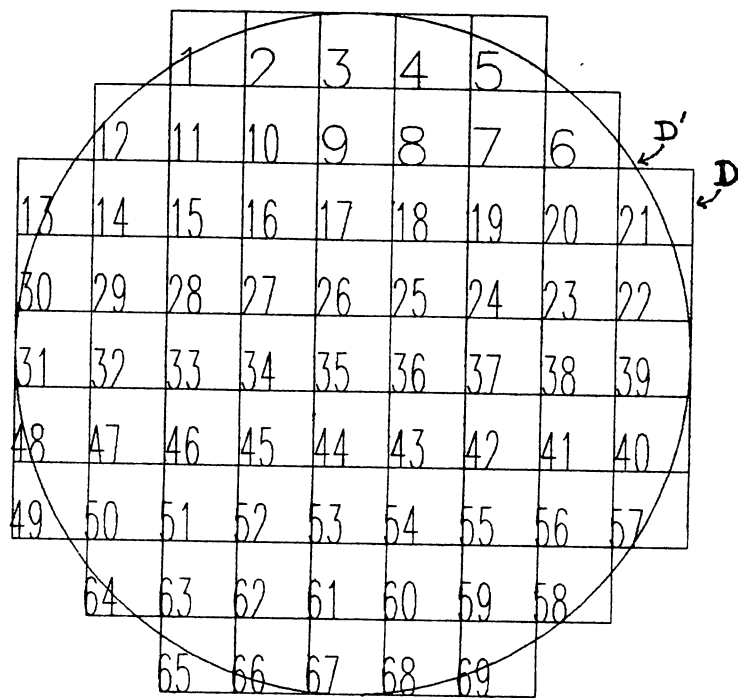


Figure 3. Approximation of Circular and Discrete Disks

should be minimum for the edge-line operator. Depending upon the accuracy of the fit, the operator indicates failure or success of detection.

To minimize computation, Hueckel used radial Fourier series taking the first nine functions to approximate I . The function I is represented as I_0 after this approximation. The set of all real functions over D' is a Hilbert space H . Let $\{H_i(x, y) \mid i = 0, \dots, \infty\}$ be an orthonormal basis of H .

$$\text{tuple} = ((c_x, c_y), r_-, r_+, t_-, t_+, b_-)$$

$$a_i = \int_{D'} H_i(x, y) \cdot I_0(x, y, \text{tuple}) \, dx dy$$

$$s_i = \int_{D'} H_i(x, y) \cdot S(x, y, \text{tuple}) \, dx dy$$

$$N_0^2(\text{tuple}) = \sum_{i=0}^{\infty} (a_i - s_i(\text{tuple}))^2$$

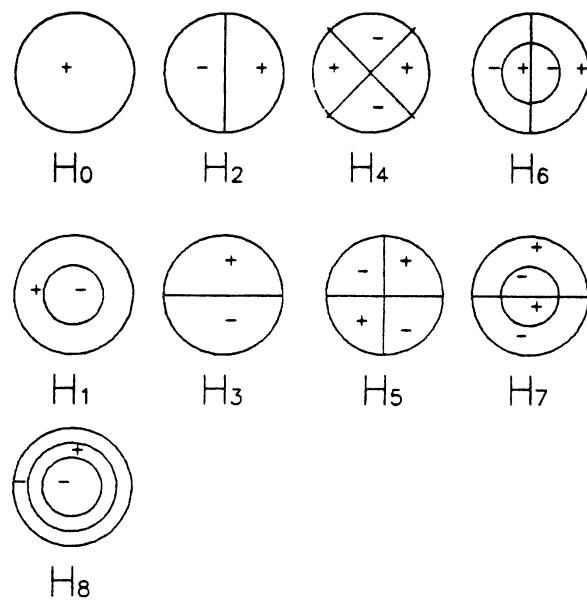
$$N^2(\text{tuple}) = \sum_{i=0}^8 (a_i - s_i(\text{tuple}))^2$$

The operator should minimize N_0 . The equations of H_i for $i = 0$ to 8 are given in [HUEALV73], and are not reproduced here. The frequency of H_i is defined as the number of zero crossing lines, excluding the periphery of the disk D' as shown in fig. 4.

For the acceptance of an edge or line in the operator, the acceptance criterion is as follows.

$$|N|^2/|S|^2 < \text{conf} - \text{diff}/|S|^2$$

where conf and diff are two threshold values to accept an edge/line. Conf is related to the noisiness of a pattern and diff to the contrast of the pattern. The values of conf and diff are to be chosen based on the noise of the pictures likely to be processed, and after conducting several

Figure 4. Frequencies of H_i

experiments with different values. The values used by Hueckel are $\text{conf} = 0.5$ and $\text{diff} = 2.5$ for dark line detection.

The given algorithm gets the intensity values of the pixels in the picture over an area that lies within a chosen disk size, conf , diff , and pattern P . $P = 0, 1, 2$ tells the operator to look for edge-lines, edges, and lines. If $P = 2$ then $PP = -1, 0, +1$ tells the operator to look for dark lines only, all lines, and white lines only respectively. The operator returns either success ($\text{succ} = 1$), or failure ($\text{succ} = 0$) and indicates to the main program to try ($\text{try} = 1$) the operator again after relocating the disk in the vicinity. Absence of a pattern is assumed at a location within the disk area when $\text{succ} = 0$ and $\text{try} = 0$.

Conversion to Vectors

After the edge operator finds the edge/line points in the image, all the related points must be linked and grouped as vectors.

Nevatia [NEVLOB76] observes that the difficulty in locating boundaries of objects is due to the fact that local surface discontinuities do not necessarily correspond to the object boundaries. He presented an algorithm to find groups of edges that connect in an approximate straight line.

In view of its simplicity, and the possible use of some of the parameters from the Hueckel edge operator in

implementing it, the algorithm suggested by Nevatia is more appropriate for the present purpose of vector conversion. Therefore, it is decided to use this algorithm with suitable modifications as applicable to our requirements. The algorithm is described as presented in [NEVLOB76].

The following assumptions are made about the edge elements.

- i) Each edge element is considered to have a position and direction associated with it.
- ii) Two oppositely directed elements are considered to have different directions (differing by 180°).
- iii) Length of an edge element as determined by an edge operator is unimportant.

Following are the steps in the algorithm.

- 1) Examine each edge element and put it in a set E_j if its direction falls within a chosen range $\Delta\theta$, of the direction θ_j .
- 2) Transform the coordinates so that the new x-axis, X' lies along θ_j .
- 3) Divide the image plane into parallel strips of a fixed size, normal to X' . Each edge element in E_j will fall into one group (bucket) of direction. These can be ordered by the value of y coordinate.
- 4) Link the edges in one group (bucket). If two consecutive elements in the edge list for a bucket

- differ by a small amount, within a threshold value, then the two belong to the same segment.
- 5) Then link segments in neighboring buckets. If the end points of two segments in adjacent buckets are within some threshold value of their x and y coordinates then they belong to the same segment.
 - 6) Retain only segments whose length is above a (threshold value) fixed number of pixels.

Storing the Vectors in a File

The x and y coordinates of the vector points making lines are stored in ASCII format in a file so that these points can be joined together by any CAD software for further manipulation. One of the file formats available is Design CAD file format [ASBDCD91]. The specifics of this file format can be found in Appendix B.

CHAPTER IV

IMPLEMENTATION DETAILS

Introduction

For its simplicity and availability of orientation of vectors, Hueckel's edge detector is used to find the instances of edges and lines as discussed in earlier chapters. The scanning algorithm by which the operator is applied over an image and the edge aggregation algorithm are presented here.

Other features like getting the names of the input raster image file and output drawing file, listing the files in a directory, allowing the user to alter or choose some parameters, etc., are added at appropriate stages of the program.

Scanning Algorithm

Starting from the top left hand corner of an image, the image is traversed horizontally by the edge detection operator moving four pixels at one time. Since the diameter of the operator disk is nine pixels, steps of four pixel movement seem to be appropriate horizontally and vertically for a preliminary scan. When the operator finds an instance

of an edge or line, it follows the line as long as successful detections continue. The traversal path of the operator is noted down in a separate memory buffer. When the end of a line is reached, and traversal resumes where the edge was first detected. When the number of vectors reaches a limit (say 3000, depending on the available memory), they are aggregated to reduce the number of continuous line segments. At this point, the number of points that constitute a vector are printed in the output file along with the number of them. The algorithm is as follows.

```

scan()
{
    int  x, y, // x, y coordinates
        midx, midy, // coordinates of the center of
                    // the operator
        succ, // indicates if the operator found a line
        linestart, // boolean, indicates a line has
                    // begun
        dumx[], dumy[]; // vectors along a path go here

    for(y = 4; y < maxvalue of y; y += 4){
        for(x = 4; x < max value of x; x += 4){
            linestart = 0;
            while(the point was not visited){
                read disk; // fill the operator disk with

```



```

        // intensity values.
call edge detector(try,succ,...);
if(try){    // try is global
    if(succ){
        if(the center point is close to the
            center of the operator){
            mark point as visited;
            if(linestart)
                mark the points along the vector;
            else
                linestart = 1; // a line starts
            dumx[count] = midx;
            dummy[count++] = midy;
            move midx,midy along the vector;
        } // end if (the center ..)
        else
            reposition the operator close to
                vector;
    } // end if (succ)
    else{ // no line is found, but retry
        reposition the operator as per parameters
            from operator
    }
    if end of image is reached break;
} // end if (try)
try = 0; // reset try

```

```

    } // end while
    call link_vect(); // connect the vector points
    reset count = 0; // counter for # of vector
                        // points
    if(# of vectors >= threshold value){
        output vector points in a file;
        reset counter for # of vector points;
    }
} // end for
} // end for
output remaining vector points;

}/* end of scan() */

```

Linking Algorithm

As seen above, when the end of a vector is reached, linking of the vector points is done. In this process, starting from the first noted point on the vector, three consecutive points are checked to see if they are close enough to be on a single line. A threshold value (say 3°) is chosen for checking this. If three points can be assumed to be on a line, then the last point of the three points is checked in the same way, until the end of the vector is reached. Whenever the three consecutive points, as found by the edge detector, make two line segments, they are noted down as two separate line segments of a vector. If there is

only one point in a vector it may be omitted. When the total number of vector points reaches a threshold value (say, 3000), they are output into the output file in the appropriate format, and the corresponding counters are reset.

The linking algorithm is as follows.

```

link_vect()
{
    int i = 0;
    double delx1, dely1, delx2, dely2;

    jx[counter] = dumx[i]; //counter indexes into jx, jy
    jy[counter++] = dумы[i++]; // jx[], jy[] are global
    if(count == 2){
        jx[counter] = dumx[i];
        jy[counter++] = dумы[i++];
    }
    if(count < 3)
        return;
    while(i < count){ // check linearity of vector
        // points
        delx1 = dumx[i] - dumx[i-1];
        dely1 = dумы[i] - dумы[i-1];
        delx2 = dumx[i+1] - dumx[i];
        dely2 = dумы[i+1] - dумы[i];
    }
}

```

```
if(abs(tan(dely1,delx1) - tan(dely2,delx2)) <
                                LIM){
    jx[counter] = dumx[i];
    jy[counter++] = dумы[i];
} // end if
i++;
} // end while
jx[counter] = dumx[i];
jy[counter++] = dумы[i];

} /* end of link_vect() */
```

CHAPTER V

RESULTS AND CONCLUSIONS

Introduction

The scan conversion program is menu driven. It has options to set some parameters related to scanning, selecting the input raster image file and output drawing file, listing the files in a directory etc.

When the name of the input raster image file is given, the program checks to see that it exists, decompresses if necessary, and proceeds to scan the image. If an existing output drawing file name is given, it will be reused if the user chooses to do so. The input file is scanned in different passes if it cannot be scanned in one pass. The number of passes required to scan the image is determined by the available memory and the size of the raster file. During the scanning procedure, vector points are stored in an array and attempts to aggregate them are made to reduce the number. When the number of these vector points reaches a threshold value after all aggregation, they are output into the drawing file. The maximum number of points that can constitute a vector is 200. If any vector has more points, it is divided into two vectors in the output drawing file.

This restriction is imposed by the Design CAD file format.

A study was done using a number of ideal cases of line/edge detection by the Hueckel edge operator with 'conf' varying from 0.80 to 0.96 at 0.02 intervals and 'diff' varying from 0.15 to 0.04 at 0.02 intervals. These two parameters are related to the noisiness and the contrast of a pattern respectively. The best results were obtained for $\text{conf} = 0.96$ and $\text{diff} = 0.04$. However, the user can change these values, if he desires so.

The output vector file is in ASCII format which can be used by Design CAD software. This vector file can also be read by other software since it is in ASCII format, and can also be edited as desired to suit other requirements.

Results and Conclusions

A number of TIFF files were made using a hand scanner, and the scan conversion program was used to convert them to vector images.

Comparisons were made between the previously available program and the current program. The following observations were made.

1. The previous program always needs input parameters from the user for the scanning procedure, whereas the current version needs no parameters to be given by the user unless he has encountered an unusual picture which does not

suit the default parameters.

2. The previous program separates the tasks of scanning an image and converting points to vectors. The current program performs both the tasks almost simultaneously, since the track followed by the operator gives the vector points along the line. This also reduces user effort.
3. Straight lines are recognized more accurately now than by the previous program.
4. Since the current version uses a more sophisticated mathematical approach, it involves more numerical calculations and takes more processing time than the previous program. However, if future plans to extend this program to grayscale images are carried out, the current program will take the same amount of time because the number of calculations involved remains the same, whereas the previous program will spend a lot of additional time in thresholding and finding edges
5. The vector images generated from the current program appear to be satisfactory. If the lines in the input raster images are thicker, there is some noise in the vector image. This can be eliminated by editing the output by using any CAD software.

Recommendations for Further Study

One of the problems of the raster to vector conversion is the recognition of text. Small letters of text are usually not recognized properly by the currently available operators which are used to recognize edges and lines. Thus the current conversion program rarely produces readable results from small letters. This area must be studied to find out better operators which can work well with both geometric diagrams and small text.

Processing speed is one of the problems that must be addressed by further studies.

Thick lines create another problem. They appear as two parallel lines, sometimes with some noise present between them. A better method to show such thick lines as single lines should be investigated. Since the Hueckel's method allows for scanning operators with other diameters besides nine pixels, implementing these should add some flexibility for dealing with more or less coarse parts of drawings.

BIBLIOGRAPHY

- [ABDQDA79] Abdou, Ikram E., and Pratt, William K.,
Quantitative Design and Evaluation of
Enhancement/Thresholding Edge Detectors, Proceedings of
the IEEE, 67, 5. (May, 1979).
- [ACTIF90] Aldus Corporation, Tag Image File Format
Specification, Revision 5.0, (1990).
- [ASBD91] American Small Business Computers, Design CAD
Drawing File Format, Design CAD 2-D: Reference Manual,
(1991), 445-450.
- [CANACA86] Canny, J., A Computational Approach to Edge
Detection, IEEE Transactions on Pattern Analysis and
Machine Intelligence, PAMI-8(6), (Nov., 1986) 676-698.
- [CHOABD72] Chow, C.K., and Kaneko, T., Automatic Boundary
Detection of the Left Ventricle from Cineangiograms,
Computers in Biomedical Research, 5, (1972), 388-410.
- [DAVCOT86] Davies, E.R., Constraints on the Design of
Template Masks for Edge Detection, Pattern Recognition
Letters and Signal Processing, 4, (1986), 111-120.
- [DAVMVT90] Davies, E.R., Machine Vision: Theory, Algorithms,
Practicalities, Academic Press, (1990).
- [DAVASO75] Davis, Larry S., A Survey of Edge Detection
Techniques, Computer Graphics and Image Processing,

- (Sep., 1975), 4, 3, 248-270.
- [DUDUOH72] Duda, R.O., and Hart, P.E., Use of the Hough Transformation to Detect Lines and Curves in Pictures, Communications the ACM, 15, (Jan., 1972), 11-15.
- [FOLCGP90] Foley, James D., van Dam, Andries, Feiner, Steven K., Hughes, John F., Computer Graphics: Principles and Practice, Second Edition, Addison-Wesley Publishing Company, (1990).
- [GOKEDW90] Gökmen, M., and Li, C.C., Edge Detection With Iteratively Refined Regularization, 10th International Conference on Pattern Recognition, I, (Jun., 1990), 690-693.
- [GRIEDI73] Griffith, Arnold K., Edge Detection in Simple Scenes Using A Priori Information, IEEE Transactions on Computers, c-22, 4, (Apr., 1973).
- [GRIMMF73] Griffith, Arnold K., Mathematical Models for Automatic Line Detection, Journal of the ACM, 20, 1, (Jan., 1973), 62-80.
- [HOUMAM62] Hough, P.V.C., Method and Means for Recognizing Complex Patterns, U.S. Patent 3069954, (Dec., 18, 1962).
- [HUEAOW71] Hueckel, Manfred H., An Operator Which Locates Edges in Digitized Pictures, Journal of the ACM, 18, 1, (Jan., 1971), 113-125.
- [HUEALV73] Hueckel, Manfred H., A Local Visual Edge Operator which Recognizes Edges and Lines, Journal of the ACM,

20, 4, (Oct., 1973), 634-647.

- [KUNRED90] Kundu, Amlan, Robust Edge Detection, Pattern Recognition, 23, (1990), 423-440.
- [MARTOE80] Marr, D., and Hildreth, E., Theory of Edge Detection, Proc. of the Royal Society of London, B207, (1980), 187-217.
- [MARAAO76] Martelli, A., An Application of Heuristic Search Methods to Edge and Contour Detection, Communications of the ACM, (Feb., 1976), 73-83.
- [MERASA75] Mero, L., and Vassy, Z., A Simplified and Fast Version of the Hueckel Operator for Finding Optimal Edges in Pictures, Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, Tsibili, U.S.S.R, (Sep., 1975), 650-655.
- [NEVLOB76] Nevatia, R., Locating Object Boundaries in Textured Environments, IEEE Transactions on Computers, 25, (Nov., 1976), 1170-1175.
- [NEVEOA77] Nevatia, R., Evaluation of a Simplified Hueckel Edge-Line Detector, Computer Graphics and Image Processing, 6, (1977), 582-588.
- [NEVACE77] Nevatia R., A Color Edge Detector and Its Use in Scene Segmentation, IEEE Transactions on Systems, Man, and Cybernetics, SMC-7, 11, (Nov., 1977), 820-826.
- [NEVLFE80] Nevatia R., and Babu, Ramesh K., Linear Feature Extraction and Description, Computer Graphics and Image Processing, 13, (1980), 257-269.

- [NEVMP82] Nevatia, R., Machine Perception, Prentice-Hall, Inc., (1982).
- [PETASO82] Pete, T., and Malah, D., A Study of edge detection algorithms, Computer Graphics and Image Processing, 20, (1982), 1-21.
- [PRADIP78] Pratt, W.K., Digital Image Processing, John Wiley & Sons, (1978).
- [PREOEA70] Prewitt, J.M.S., Object Enhancement and Extraction, Picture Processing and Psychopictorics, Academic Press, (1970).
- [ROSDIP81] Rosenfeld, A., and Kak, A., Digital Image Processing, Academic Press, (1981).
- [ROSEAC70] Rosenfeld, A., and Thurston, M., Edge and Curve Detection for Visual Scene Analysis, IEEE Transactions on Computers, C-20, (May, 1970), 562-569.
- [STEETD90] Stein, Fridtjot, and Medioni, Gerard, Efficient Two Dimensional. Object Recognition, 10th International Conference on Pattern Recognition, I, (Jun., 1990), 13-17.

APPENDIXES

APPENDIX A

TAG IMAGE FILE FORMAT

The following is the TIFF file format (Revision 5.0). Only relevant portions from [ACTIF90] are reproduced here. More details can be obtained from [ACTIT90] or by contacting at either of the following addresses.

Developer's Desk	Windows Marketing Group
Aldus Corporation	Microsoft Corporation
411 First Ave. South	16011 NE 36th Way
Suite 200	Box 97017
Seattle, WA 98104	Redmond, WA 98073-9717

The largest possible TIFF file is 2**32 bytes in length. The recommended file extension for MS DOS, UNIX, and OS/2 is ".TIF" and for Macintosh it is ".TIFF".

Structure

In, TIFF, individual fields are identified with a unique tag. This allows particular fields to be present or absent from the file as required by the application.

A TIFF file begins with an 8-byte "image file header" that points to one or more "image directories". The image file directories contain information about the images, as

well as pointers to the actual image data. See fig. 1 (page 15).

We will describe these structures in more detail.

Image File Header

A TIFF file begins with an 8-byte image file header, containing the following information:

Bytes 0-1: The first word of the file specifies the byte order used within the file. Legal values are:

"II" (hex 4949)

"MM" (hex 4D4D)

In the "II" format, byte order is always from the least significant to most significant, for both 16-bit and 32-bit integers. In the "MM" format, byte order is always from most significant to least significant, for both 16-bit and 32-bit integers. In both formats, characters are stored into sequential byte locations. All TIFF readers should support both byte orders.

Bytes 2-3: The second word of the file is the TIFF "version number". This number, 42 (2A in hex), is not be equated with the current Revision of the TIFF specification. In fact,

the TIFF has never changed, and probably never will. If it ever does, it means that TIFF has changed in some way so radical that a TIFF reader should give up immediately. The number 42 should be used as an additional check that it is indeed a TIFF file. A TIFF file does not have a real version/revision number.

Bytes 4-7: This long word contains the offset (in bytes) of the first Image File Directory. The directory may be at any location in the file after the header but must begin on a word boundary. In particular, an Image File Directory may follow the image data it describes. Readers should simply follow the pointers, wherever they lead. (The term "byte offset" is always used in this document to refer to a location with respect to the beginning of the file. The first byte of the file has an offset of 0.)

Image File Directory

An Image File Directory (IFD) consists of a 2-byte count of the number of entries (i.e., the number of fields), followed by a sequence of 12-byte field entries, followed by

a 4-byte offset of the next IFD (or 0 if none). Do not forget to write the 4 bytes of 0 after the last IFD

Each 12-byte IFD entry has the following format:

Bytes 0-1 contain the Tag for the field.

Bytes 2-3 contain the field Type.

Byte 4-7 contain the Length (or Count) of the field.

Byte 8-11 contain the Value Offset, the file offset (in bytes) of the Value for the field. The Value is expected to begin on a word boundary; the corresponding Value Offset will thus be an even number. This file offset may point to anywhere in the file, including after the image data.

The entries in an IFD must be sorted in ascending order by Tag. Note that this is not the order in which the fields are described in this document.

In order to save time and space, the Value Offset is interpreted to contain the Value instead of pointing to the Value if the Value fits into 4 bytes. If the Value is less than 4 bytes, it is left justified within the 4 byte Value Offset, i.e., stored in the lower-numbered bytes. Whether or not the Value Offset fits within 4 bytes is determined by looking at the Type and Length of the field.

The Length is specified in terms of the data type, not the total number of bytes. A single 16-bit word (SHORT) has a length of 1, not 2, for example. The data types and their lengths are described below:

- 1 = BYTE An 8-bit unsigned integer.
- 2 = ASCII 8-bit bytes that store ASCII codes; the last byte must be null.
- 3 = SHORT A 16-bit (2-byte) unsigned integer.
- 4 = LONG A 32-bit (4-byte) unsigned integer.
- 5 = RATIONAL Two LONG's; the first represents the numerator of a fraction, the second the denominator.

The value of the Length part of an ASCII field entry includes the null. If padding is necessary, the Length does not include the pad byte. Note that there is no "count byte", as there is in Pascal-type strings. The Length part of the field takes care of that. The null is not strictly necessary, but may make things slightly simpler for C programmers.

The reader should check the type to ensure that it is what he expects. TIFF currently allows more than 1 valid type for some fields. For example, ImageWidth and ImageLength were specified having type SHORT. Very large images with more than 64K rows or columns are possible with some devices even now. Rather than add parallel LONG tags

for these tags, it is cleaner to allow both SHORT and LONG for ImageWidth and similar fields.

Note that there may be more than one IFD. Each IFD is said to define a "subfile". One potential use of subsequent subfiles is to describe a "sub-image" that is somehow related to the main image, such as a reduced resolution version of the image.

The Fields

This section describes the fields defined in this version of TIFF. The documentation for each field contains the name of the field, the Tag value, the field Type, the Number of Values (N) expected, comments describing the field, and the default value, if any. Readers must assume the default value if the field does not exist.

BitsPerSample

Tag = 258 (102)

Type = SHORT

N = SamplesPerPixel

Number of bits per sample.

Default = 1.

Compression

Tag = 259 (103)

Type = SHORT

N = 1

1 = No compression, but pack data into bytes as

tightly as possible.

2 = CCITT Group 3 1-dimensional Modified Huffman run length encoding. BitsPerSample must be 1, since this type of compression is bilevel images.

32773 = PackBits compression, a simple byte oriented run length scheme for 1-bit images. Data compression applies to only raster image data, as pointed to by StripOffsets. All other TIFF information remains unaffected.

Default = 1.

ImageLength

Tag = 257 (101)

type = SHORT or LONG

N = 1

The image's length (height) in pixels (Y: vertical).

The number of rows (sometimes described as 'scan lines') in the image.

No default.

ImageWidth

Tag = 256 (100)

Type = SHORT or LONG

N = 1

The image's width in pixels (X: horizontal). The number of columns in the image.

No default.

PhotometricInterpretation

Tag = 262 (106)

Type = SHORT

N = 1

0 : For bilevel and grayscale images : 0 is imaged as white. $2^{**}\text{BitsPerSample}-1$ is imaged as black.

1 : For bilevel and grayscale images : 0 is imaged as black. $2^{**}\text{BitsPerSample}-1$ is imaged as white.

RowsPerStrip

Tag = 278 (116)

Type = SHORT

N = 1

The number of rows per strip. The image data is organized into strips for fast access to individual rows when the data is compressed-though this field is valid even if the data is not compressed.

RowsPerStrip and ImageLength together tell us the number of strips in the entire image. The equation is
$$\text{StripsPerImage} = (\text{ImageLength} + \text{RowsPerStrip} - 1) / \text{RowsPerStrip}.$$

Note that either SHORT or LONG values can be used to specify RowsPerStrip. SHORT values may be used for small TIFF files.

SamplesPerPixel

Tag = 277 (115)

Type = SHORT

N = 1

The number of samples per pixel. SamplesPerPixel is 1 for bilevel, grayscale, and palette color images.

SamplesPerPixel is 3 for RGB images.

Default = 1.

StripBytecounts

Tag = 277 (115)

Type = SHORT

N = StripsPerImage for PlanarConfiguration equal to 1, SamplesPerPixel * StripsPerImage for PlanarConfiguration equal to 2.

For each strip, the number of bytes in that strip.

No default.

StripOffsets

Tag = 272 (111)

Type = SHORT or LONG

N = StripsPerImage for planar configuration equal to 1, and SamplesPerPixel * StripsPerImage for PlanarConfiguration equal to 2.

For each strip, the byte offset of that strip with respect to the beginning of the TIFF file.

No default.

XResolution

Tag = 282 (11A)

Type = RATIONAL

N = 1

The number of pixels per ResolutionUnit in the X direction.

No default.

YResolution

Tag = 283 (11B)

Type = RATIONAL

N = 1

The number of pixels per ResolutionUnit in the Y direction.

No default.

Apart from these fields, information fields like Artist (Tag = 315), DateTime (306), Software that created the image (305) etc., are also available.

APPENDIX B

DESIGN CAD DRAWING FILE FORMAT

The Design CAD file format is described here. Only relevant portions are reproduced from [ASBDCD91].

The Design CAD drawing file is an ASCII file, with the data present in character format. Each "record" in the file is actually a line in a text file. There are four types of records, or lines, in the file:

1. Header line
2. Entity line
3. Point line
4. String line

The header line tells, among other things, how many entity lines are to follow. The entity line tells what type of drawing entity and how many points and strings to follow. The Point and String lines contain point and strings, respectively.

THE HEADER LINE

The Header Line contains 5 numbers:

1. The minimum X coordinate in the file.
2. The minimum Y coordinate in the file.

3. The horizontal length of the drawing.
4. The vertical height of the drawing.
5. This number is no longer used. A zero remains here for compatibility.

The coordinates are in arbitrary units. The Drawing Units Size (entity type 40) can be used to determine the "real world" coordinates or the size of one Drawing Unit.

The Entity Line

The entity line contains 6 or 8 numbers:

1. The type of entity:

- 0 = Handles
- 1 = Line
- 2 = Oval
- 3 = Text
- 4 = Curve
- 5 = Arc
- 6 = Paint-Old Format
- 15 = Attribute
- 16 = Circle, Circular arc
- 17 = Hatch
- 18 = Dimension, Linear
- 19 = Dimension, Angular
- 20 = System Parameters - Old Format
- 21 = New Layer
- 22 = Text Arc

- 23 = Layer Names
- 24 = Arrow
- 40 = Drawing Unit Size and Cursor Step Size
- 41 = System Parameters
- 70 = Point Mark
- 72 = Diameter or Radius Dimension
- 73 = Dimension coordinate (Hor/Ver Distance to
a Reference Point)

2. The Number of points to follow. (0-200)
(or the layer number, if the entity = 21)
3. The line pattern scale. (1 = normal)
4. The line width.
5. The line type. (0-8)
6. The color (1-16)
7. Reserved (Optional to Dimension Entities)
8. Dimension Format (Optional to Dimension Entities)

Following the entity line are the point lines and the string lines. There is one point line, and one string line per line. Strings are found on Text, Text Arc, and Hatch entities.

If the entity type is 20, then the drawing parameters will follow, terminated by an asterisk (*). If the entity type is 40 or 41, then the number of lines that follow on that entity. Each line may have more than own value, and the values do not represent points.

Earlier versions of the program have the line type and color combined in the same value. This format is still compatible. If there are only 5 values on the entity line, the color and line type are assumed to be combined in the fifth value.

THE POINT LINE

The point lines follow the entity line. Each line contains two numbers: the X and Y coordinates of the point on the screen. These values will change when the drawing is retrieved according to the size, location, and angle of the Retrieve command.

There can be no more than 200 points per entity. A line or curve entity with more than 200 points may be used by splitting it into two or more entities.

The point line can contain an X, Y, and Z coordinate, but the Z coordinate is ignored in Design CAD 2-D.

THE STRING LINE

The string line contains a string of text. It follows the point lines. The string line is used only with Text, Text Arc, Attribute, and Hatch entities. Be sure that the string is terminated with a carriage return character, because trailing blanks will be used in determining the string length and may cause undesirable results. The maximum string length is 80 characters.

APPENDIX C

SCANNING ALGORITHM CODE

```
void scan()
{
    int i = 0, j, listn, midx, midy, pmidx=0, pmidy=0,
        linestart = 0, cnt=0, succ, count=0, dumx[200],
        dummy[200], index = 0, ptr[1000], seg = 0;
    float cx, cy, r, edjs, lins, delx, dely;
    BYTE input[DISKSIZE];
    char str[20];

    ll = 0; ptr[seg++] = -1;
    init_h(); // initialise the h[][] array.
    for(iy = 4; iy+5 < isymax; iy += 4){ /* move
        horizontally on the image */
        for(ix = 4; ix+5 < isxmax; ix += 4){
            linestart = 0;
            itry = 1;
            midx = ix;
            midy = iy;
            while(!check_visit(midx, midy) && itry){ /* check if
                the point was visited already */
```

```

if((pmidx == midx) && (pmidy == midy))
    break;
pmidx = midx; pmidy = midy;
if(read_disk(midx,midy,input)){ /* fill and check
                                if disk has all 0's/1's */
//if not, apply edge detector
hoper(input, &succ, &cx, &cy, &r, &edjs,&lins);
if(itry){
    if(succ){ /* an edge/line is found.
              if(fabs(r) < 0.11/max(fabs(cx),fabs(cy))){
                  setpoint(midx,midy);
                  if(linestart)
                      mark_vect(l1); /* mark the points on
                                      the vector as visited */
              else
                  linestart = 1;
              if(cx < 0){ /* adjust the line/edge
                          parameters for the image */
                  cx = - cx;
                  cy = -cy;
              }
              dumx[count] = midx;
              dumy[count++] = midy;
              if(fabs(cy) > 0.95) {
                  midx += 4 * fabs(cy) + 0.5;

```

```

        midy -= (cy <= 0)?(4 * cx + 0.5):- (4 *
                                                cx + 0.5);
    }
    else{
        midx += 4 * cy + 0.5;
        midy += 4 * cx + 0.5;
    }
} // end if(abs)
else{
    setpoint(midx,midy);
    midx += ((delx = r*cx/0.21) < 0) ? (delx
        - 0.5) : (delx+0.5);/*reposition
        operator to try again */
    midy -= ((dely = r*cy/0.21) < 0)?(dely-
        0.5):(dely+0.5);
} // end else
} // end if(succ)
else{
    setpoint(midx,midy);
    midx += ((delx = r*cx/0.21) < 0) ? (delx-
        0.5): (delx+0.5); /* repos.
        operator to retry */
    midy -= ((dely = r*cy/0.21) < 0) ? (dely-
        0.5): (dely+0.5);
    linestart = 0;
    if(!delx && !dely) /* if this is the same

```

```

        position as before, abort next trial*/
        break;
    } // end else
    if((midx-5 < 0) || (midx+5 > isxmax) ||
        (midy-5 < 0) || (midy+5 > isymax))
        break; /* check for boundary of the image
                buffer */
    } // end if(itry)
} // end if(read_disk())
else
    itry = 0;
}/* while*/
if(count){
    if(count-1){
        link_vect(dumx,dumy,&index,count); /* link
                vectors */
        ptr[seg] = index-1;
        seg++;
    }
    count = 0;
} // end if(count)
if(index >= 3000){ // output vectors
    putindc2(seg,ptr);
    index = 0;
    seg = 0;
    ptr[0] = -1;
}

```

```
    }  
  }// for  
}// for  
if(index < 3000) // output remaining vectors  
  putindc2(seg,ptr);  
} // end of scan.
```


VITA ²

Reddy V. V. S. Mallidi

Candidate for the Degree of
Master of Science

Thesis: USING A DISK OPERATOR TO CONVERT RASTER IMAGES OF
ENGINEERING DRAWINGS TO VECTOR IMAGES

Major Field: Computer Science

Biographical:

Personal Data: Born in Pallakadim, A.P, India, March
16, 1963, the son of M.V.S. Reddy and Sathyavathi.

Education: Received Bachelor of Engineering degree
(Civil Engineering) from Andhra University,
Waltair, India, in May, 1985 and Master of
Technology degree (Civil Engineering), from Indian
Institute of Technology, Kharagpur, India, in
August, 1990; completed requirements for the
Master of Science degree, at Oklahoma State
University, in May, 1992.

Professional Experience: Research Assistant, Department
of Computer Science, Oklahoma State University,
August, 1991, to January, 1992; Programmer,
American Small Business Computers, Pryor, OK, May,
1991, to August, 1991; Programmer, Department of
Psychology, Oklahoma State University, January,
1991, to May, 1991; Para-professional, Help Desk,
University Computer Center, Oklahoma State
University, January, 1991, to May, 1991;
Programmer, Oklahoma Resources Integrated General
Information Network Systems, Business School,
June, 1990, to December, 1990; Assistant Executive
Engineer, Neyveli Lignite Corporation, Neyveli,
India, July, 1986, to November 1989.