1-1-2003

Speedup of Optical Scanner Characterization Subsystem

Roger D. Clements
Boise State University

Elisa H. Barney Smith
Boise State University

 DOI: 10.1117/12.476060

Speedup of Optical Scanner Characterization Subsystem

Roger D. Clements and Elisa H. Barney Smith
Electrical and Computer Engineering Department
Boise State University
Boise, ID 83725-2075 USA

ABSTRACT

Small image deformations such as those introduced by optical scanners significantly reduce the accuracy rate of optical character recognition (OCR) software. Characterization of the scanner used in the OCR process may diminish the impact on recognition rates. Theoretical methods have been developed to characterize a scanner based on the bi-level image resulting from scanning a high contrast document. Two bottlenecks in the naïve implementation of these algorithms have been identified, and techniques were developed to improve the execution time of the software. The algorithms are described and analyzed. Since approximations are used in one of the techniques, the error of the approximations is examined.

Keywords: Image degradation, Point spread function, Intensity binarization, Scanner characterization, Synthetic character.

1. INTRODUCTION

In a series of studies commissioned by the U. S. Department of Energy, the University of Nevada, Las Vegas (UNLV) tested several OCR software packages and found that their accuracy rate falls abruptly when the scanned image quality degrades even "slightly," as perceived by humans[1]. In an effort to stimulate research in recognition problems, H. S. Baird developed a model containing ten imaging defects and deformations introduced by printing and scanning machines that affect OCR accuracy[2]. Of those described by Baird, three deformations have the most significant influence on OCR misclassifications [6]. Two of these are estimated deterministically using a characterization system developed by E. H. Barney Smith[4, 5]: the width w of the point spread function (PSF), and the binarization threshold θ. The model using these parameters may be used by OCR software to improve recognition rates or to predict how a document image will look after being subjected to the appropriate printing and scanning processes and, therefore, predict system performance.

The parameters for the model are estimated by observing the degradation in black and white corners that have been extracted from the scanned document. Because a great deal of noise is present in a bi-level image, primarily from intensity and spatial quantization, many corners must be analyzed to find (w_{est},θ_{est}). One of the problems with this approach is that it requires a prohibitive amount of computational resources. Two sources of complexity have been identified. In the remainder of this paper we will describe the naïve characterization subsystem and the modifications employed to reduce execution time.

2. CHARACTERIZATION ALGORITHM

The deformation of a document corner is illustrated in Figure 1. The solid black lines delimit the original document corner, and the grey area is the resulting greyscale scanned corner. Note that the corresponding lines describing the degraded corners' edges are parallel to the original edge when sufficiently far from the apex of the corner due to edge spread [3], and so the angle measurements of both corners are the same. The corner erosion distance of a black corner, d_b, is the distance from the apex of the extrapolated edge lines (dashed lines intersecting at p_1) to the degraded corner (p_2) [4]. It may expressed mathematically as

$$d_b = \frac{-wESF^{-1}(\theta)}{\sin\left(\frac{\phi}{2}\right)} + f_b^{-1}(\theta; w, \phi) \tag{1}$$

where ESF (Edge Spread Function) is the cumulative marginal of the PSF, and f_b is the greyscale profile of the blurred corner along the bisector, defined as

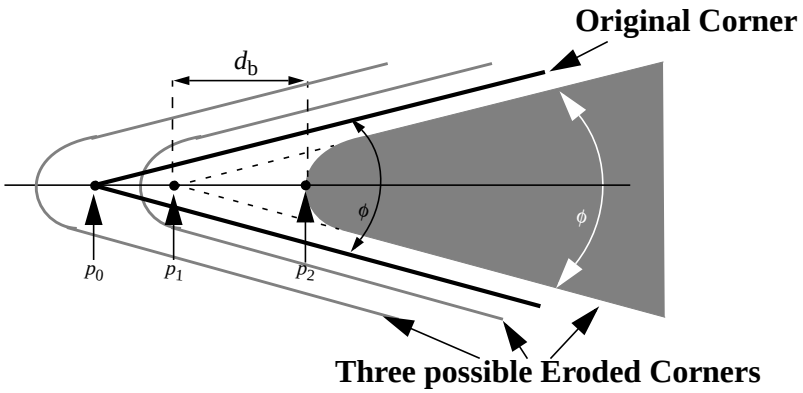

Figure 1: Three possible ways in which a blurred wedge (grey area) may be displaced from the original position (black lines) showing measurements after blurring and thresholding.

$$f_b(d_{0b};w,\phi d) = \int_{x=0}^{\infty} \int_{y=x\tan\frac{\phi}{2}}^{x\tan\frac{\phi}{2}} PSF(x-d_{0b})dydx \ . \tag{2}$$

Equation 1 expresses the corner erosion distance as a 3D surface above the w-θ plane for a constant ϕ. Corner erosion surfaces for white corners, d_w, are symmetric with those for black surfaces about the $\theta = \frac{1}{2}$ line. A closed form for the erosion distance is not typically available, but can be numerically computed for fixed ϕ, w and θ; thus an erosion surface for a fixed corner angle may be sampled at discrete points (w_i, θ_j), $w_i \in W$ and $\theta_j \in \Theta$, for some set of PSF widths W and threshold values Θ.

When a scanned corner is located in a document, its edge line positions are estimated and the angle ϕ_s and erosion distance d_{bs} are measured. An erosion distance surface for the angle ϕ_s is then sampled and a level curve extracted corresponding to the corner erosion value of d_{bs} measured from the scanned image. This level curve describes a locus of candidate scanner parameters (w, θ) that could have produced the measurements of the scanned corner. The point at which loci from a black and white corner intersect is the single value (w_{est}, θ_{est}) that could have produced both corners. To minimize the effects of noise, many black and white corners (N_b and N_w) must be measured, and for each one, an erosion surface generated and a level curve extracted. The multiple intersection points are found and used to estimate a single scanner characterization (w_{est}, θ_{est}).

To generate an erosion surface, at each sampling point (w, θ) a synthetic corner is constructed at high resolution and convolved with the PSF with width w and the erosion distance is calculated at each threshold θ. Synthetic corner construction and convolution are computationally expensive - usually five to twenty minutes are required for each surface on the current system, depending on the required sampling resolution (750MHz PIII, 384MB SDRAM running MATLAB on Windows 2000). Then, after the level curves are extracted, the erosion surface is discarded. Furthermore, the N_b black curves and N_w white curves may have up to N_bN_w intersection points that must be found. Each curve is represented by sample points that delimit line segments. If there are L_b line segments belonging to the N_b black curves and L_w line segments belonging to the N_w white curves, then potentially $L_bL_w >> N_bN_w$ intersection checks must be made to find all the intersections. Not only must the location of the intersection between each line segment be found, but it must also be determined whether the pair of line segments intersect between the endpoints. This is also an expensive operation, typically requiring hours or days to complete for a large dataset. The operations of 1) level curve extraction, and 2) the location of intersections, constitute the bulk of the time spent by the subsystem to characterize the scanner, and methods to reduce these execution times will be described in this paper.

3. CONSTRUCTING EROSION DISTANCE SURFACES

The angle measurement of corners in a document image are not known a priori, so they must be estimated as the corner is analyzed. This leads to potentially different measurements for each corner angle. For each corner measure, a specific erosion distance surface is needed, from which level curves are extracted. Two techniques are proposed to quickly approximate the level curves represented by corner measurements, and thus reduce execution time required to find (w_{est}, θ_{est}): Surface Interpolation, and Erosion Distance Offset. Each technique requires a discrete set of erosion distance surfaces S,

one for each of the (pre-determined) discrete corner angles $\phi_i \in \Phi$, that will be used for all scanned corners.

3.1 Surface Interpolation.

The Erosion Surface Interpolation technique uses two surfaces from S to linearly interpolate another between them. More specifically, given the set of corner erosion surfaces S, and a scanned corner of angle ϕ_s, $\phi_j \le \phi_s \le \phi_{j+1}$, for some ϕ_j, $\phi_{j+1} \in \Phi$, then for each $w_k \in W$ and $\theta_m \in \Theta$ a new surface s_a may be generated approximating the actual surface for angle ϕ_s using the formula

$$s_a(w_k, \theta_m) = s_j(w_k, \theta_m) + \frac{\phi_s - \phi_j}{\phi_{j+1} - \phi_j}(s_{j+1}(w_k, \theta_m) - s_j(w_k, \theta_m)) \tag{3}$$

The surface s_a is then used to generate the level curve at the value d_{bs}, the corner erosion distance measured from the scanned corner.

3.2 Erosion Distance Offset

Level curves are represented as sample points which delimit the line segments between them. At each of these points, for a given erosion surface and erosion distance, the partial $\Delta d_b/\Delta\phi$ has been found to be very nearly constant, for a wide range of corner measurements d_b and ϕ. Thus, a close approximation to a level curve extracted from one surface s_1, corresponding to angle ϕ_1, at a value of d_{b1} may be obtained from a nearby surface s_2, corresponding to angle ϕ_2, at a value d_{b2} given by

$$d_{b2} = d_{b1} + (\phi_2 - \phi_1)\left.\frac{\Delta d_b}{\Delta\phi}\right|_{d_{b2},\,\phi_2} \tag{4}$$

The Erosion Distance Offset technique calculates the value of $\Delta d_b/\Delta\phi$ for each of the pre-generated erosion surfaces at each d_{bi} in a given set of erosion distances D; these are recorded in a table indexed by ϕ and d_b. Then, for a scanned corner of angle ϕ_s, $\phi_j \le \phi_s \le \phi_{j+1}$, for some ϕ_j, $\phi_{j+1} \in \Phi$, and erosion distance d_{bs}, $d_{bi} \le d_{bs} \le d_{bi+1}$, for some d_{bi}, $d_{bi+1} \in D$, an approximate level curve is found from surface s_j corresponding to ϕ_j using an erosion distance

$$d_j = d_{bs} + (\phi_s - \phi_j)\left.\frac{\Delta d_b}{\Delta\phi}\right|_{d_{bs},\,\phi_s}, \tag{5}$$

where the partial is interpolated from the pre-generated table.

4. FINDING INTERSECTIONS

On a page of text, one may find several hundred corners that may be used to estimate characterization parameters, and each one corresponds to a level curve. We must assume that all level curves intersect, and so we must make N_b x N_w intersection checks. On a typical page of text, about N_b = 200 black and N_w = 300 white corners appropriate for use in this estimation procedure may be found. Then N_b x N_w = 60,000 curve intersection checks must be made. However, assuming each curve has 12 line segments, if each level curve has 12 line segments, then up to L_b L_w = 8,640,000 line segment intersection checks can be made. Calculating if and where two line segments intersect is the most time-consuming operation in this part of the subsystem, and assuming that its execution time may not be reduced, nor may the operation be approximated, the number of intersection checks must be reduced.

One may make use of the observation that a black and white curve intersect at most one time, and so once an intersection is found, the rest of the line segments in those curves do not need to be checked - we can then continue with another pair of curves. But assuming that most curves intersect between their middle line segments, this method will only reduce the average number of intersection checks by about half. The algorithms discussed here reduce the execution time of finding intersections still further.

For an algorithm that finds intersections between level curves, a computational complexity of order $O(N_b \text{ x } N_w)$ is very tight - that is, *at least* N_b x N_w intersection checks must be made. An algorithm that approaches this complexity is desirable to one that has a complexity of order $O(L_b \text{ x } L_w)$. Four methods for replacing the naïve algorithm with one approaching order $O(N_b \text{ x } N_w)$ are proposed. The first two partition the space to reduce the number of elements involved in the search. The second two are enhancements that make the partitioning more efficient. The enhancements can be used individually or jointly and with either partitioning algorithm.

4.1 Space Partitioning.

One source of inefficiency is the checking of whether or not two line segments intersect when they are "far apart," that is, when there is no possibility that they will intersect. We can reduce these unneeded checks by partitioning the (w,θ) space into rectangular partitions (containers), and, for each line segment in N_w, place the line segment into all partitions that contain any piece of it. Then we find intersections only between lines belonging to the same partition.

The parameter space is initially partitioned by placing all white line segments into partitions that contain any part of the segment (the partitions are said to be "covered" by the line segment). During the intersection-finding stage, a line segment from a black curve is considered and the covered partitions found. The black segment is then checked for intersection with all the (white) segments in all the covered partitions. This continues until the list of black segments is exhausted.

When determining the partition boundaries, there is a trade-off between partition size and execution time. The choice of partition size is dependent on the density, length and orientation of the line segments. Specifically, the partitioning should minimize unneeded intersection checks. This is not an easy goal to achieve because there are at least two subtleties that must be considered. We begin by restating the motivation behind the investigation of space partitioning: by using small partitions, we limit intersection checks to only those lines belonging to a small spatial locality. We note that as the number of partitions increase from unity, the number of intersection checks made between line segments that are "far apart" decreases. Thus, to decrease execution time, we are motivated to decrease partition sizes.

The first subtlety exists during the partitioning stage, *to wit*, finding the partitions covered by a given line segment. The current implementation finds the intersections between the line segment and the vertical and horizontal boundaries of the partitions, finds the midpoints of segments joining consecutive intersections, and compares the coordinates of the midpoint with the partition boundaries. That entire white line segment is added into each partition that contains a midpoint. Thus, if a line segment spans an m x n rectangular region of partitions, $(m-1)+(n-1)=m+n-2$ intersections must be found. This represents a significant overhead that increases execution time as the number of spanned partitions increases, or equivalently, as partition size decreases. This contrasts with the trend mentioned in the last paragraph.

A second subtlety exists during the intersection-finding stage. We begin by noting that a black line segment may (and usually does) cover several of the same partitions as a white line segment. This causes two problems: 1) an intersection check will be made for each shared partition per line that shares more than one partition, and 2) if the two line segments intersect, each intersection check beyond the first one will record an additional intersection point that is eventually used for determining the final (w_{est}, θ_{est}). Both of these problems may be solved by assigning each white line segment a unique identifier during partitioning. A dictionary data structure containing white line identifiers is used during the intersection-finding stage: before a white line segment is checked for intersection, the dictionary is consulted. If the segment's tag is not in the dictionary, the check is made and the white segment's tag is entered into the dictionary; otherwise, no additional intersection check is made, and the next white line is examined. The dictionary is re-initialized with each new black line segment. A dictionary operation is much quicker than an intersection check, but many operations are performed for each set of level curves for which Space Partitioning is called upon to process.

For the algorithm described in this paper, the partition boundaries were determined by aggregating all the coordinates from all the level curves and dividing parameter space into bins containing about the same number of coordinates. Experiments suggest that, for the level curves used, a 40 x 40 partition grid is near optimum. These values were used in the timing experiments, the results of which are reported in Table 2.

4.2 Level Curve Partitioning.

An alternative method to partitioning the (w, θ) space is to partition the level curves. A level curve found by MATLAB is a set of coordinate pairs that may be connected by straight lines. The Level Curve Partitioning algorithm partitions the curves into mutually exclusive subsets of contiguous line segments. The maximum and minimum abscissas and ordinates of all coordinate pairs in a partition are found, and are used to determine the spatial extent of the partition (the "bounding box"). During the intersection-finding stage, a black-curve partition is checked against every white-curve partition for spatial overlap between the bounding boxes. If there is an overlap, each line segment of one partition is checked for intersection with each segment of the other; otherwise, no intersection checks are made, and the search continues with the next white partition. This continues until all pairs of black and white partitions have been examined.

Similar to in the Space Partitioning method, there is a trade-off between partition size (given in line segments per partition) and execution time, which varies depending on the number of segments per level curve, the size of the partitions (measured in spatial units), the time required to determine overlap compared to the time required to check for intersec-

tions, and so on. Also, one may change the size of a given partition depending on its location in the curve - that is, since intersections are less likely at the extreme ends of the curve, different sized partitions may be defined there compared to the middle portion. The goal, here, is to minimize the spatial area of the partitions and thus reduce the chances that two bounding boxes will overlap, and so forcing us to check for line segment intersections. For the Space Partitioning algorithm, an average of four line segmant partitions were used.

In addition to the partitioning algorithms, two enhancements were developed: Level Curve Rotation and Level Curve Trimming. These can be used with the previous speedups individually or jointly. These will be introduced next.

4.3 Level Curve Rotation.

We can find a rotation angle that makes the line segments of one color mostly horizontal and the line segments of the other color mostly vertical, Figure 2b. By rotating the line segments in such a way, we can reduce the number of spatial partitions to which a line segment belongs. Also, we can reduce the spatial extent of bounding boxes and thus reduce the number of boxes that overlap. Finding a good rotation angle and rotating the line segments incurs an initial performance penalty, but total execution time of both Space Partitioning and Level Curve Partitioning, including the time to rotate the line segments, is decreased.

4.4 Level Curve Trimming

When the line segments are rotated so that they are either mostly horizontal or vertical, the segment endpoint with the smallest ordinate of the horizontal curves is found and any segments of the vertical curves that fall entirely below this value may be discarded. This may be done 4 times, once for each of the northwest, northeast, southwest and southeast areas that can be trimmed (see Figure 2c). Again, this incurs an initial penalty, but the total execution time for Space Partitioning and Level Curve Partitioning is decreased.

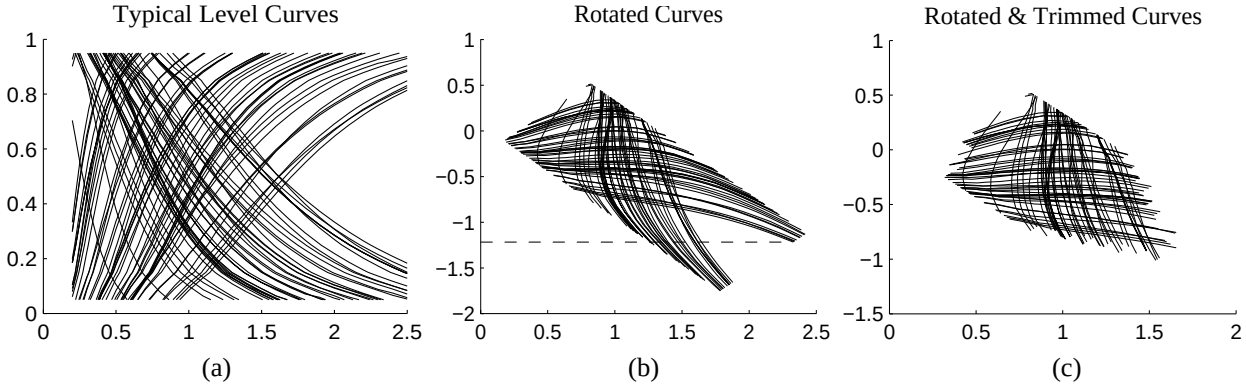

Figure 2: (a) A typical set of Level Curves. (b) The same curves rotated with the minimum ordinate of the horizontal curves indicated by a horizontal line at -1.2. (c) The rotated and trimmed results.

5. RESULTS

To test the modifications to the naïve subsystem, two different types of experiments were made. The first type compares the Erosion Distance Offset and Surface Interpolation algorithms. These algorithms use approximations to find level curves from corner measurements, and so the difference between results obtained from the naïve algorithm and those obtained with these new ones must be measured. The time required to complete the calculations must also be measured.

The second type of experiment compares the execution times of the Space Partitioning and Level Curve Partitioning algorithms. Also the benefit of including the trimming and level curve rotation is investigated. Since they do not use approximations, no error analysis is required.

5.1 Approximation Algorithms

The objective of the first experiment was to compare the results of generating an erosion contour using an approximation algorithm based on neighboring angle measurements with the erosion contours that would result from generating the same contour from a surface corresponding to that exact angle measure. Recall that the algorithms that find level curves from corner measurements use corner erosion surfaces for angles near the measured one. Figure 3 illustrates graphically how

closely approximated level curves match the actual ones. Figure 3a shows two sets of approximations. Each set consists of 4 level curves. Three of them are level curves generated using a constant d_b value from three surfaces corresponding to three consecutive angles from Φ. The fourth level curve is represented with dots rather than lines, and is derived from a surface interpolated from the outer two, using the same d_b value. Specifically, for the leftmost set, d_b=3 and ϕ=12°, 13°, 14°; the dotted line comes from using d_b=3 on a surface interpolated from the 12° and 14° surfaces. The rightmost set uses the same surfaces, but with d_b=8.

Figure 3b shows 2 sets of 3 curves. In each set, the solid curve comes from a surface of angle 11° at a constant d_b, the dashed curve from an angle 12° at the same d_b, and the dots are the curve extracted from the 12° surface at a value of d_b+ offset, as calculated in Section 3.2. The leftmost set uses d_b=3, and the other set uses d_b=8.

Note that the difference between contours extracted from surfaces only one degree apart and using the same value for d_b is not negligible. Second, note that the difference between the curve being approximated and the approximation is barely noticeable.

To test how the approximation algorithms perform in practice, angle measurements from [4,5] were obtained and used to find a set of (w_{est}, θ_{est}) pairs for each of the Erosion Distance Offset and Surface Interpolation algorithms. Each set contains 17 pairs, one for each threshold made from the greyscale images. These two resultant sets are compared against the results from [4,5] and the error is observed.

To test these approximation algorithms, *S*, *D* and Φ were generated. Φ contained each of the integral corner angles 4-75°, and for each angle in Φ, a corresponding surface in S was generated using the Gaussian PSF. For each of these surfaces $s_j \in S$, the sampling points were $W \in$ [0.2, 2.5] in increments of 0.1, and $\Theta \in$ [0.05, 0.95] in increments of 0.1. The set D={0.5, 0.75, 1.0, 1.25, 1.5 1.75, 2.0, 2.25, 2.5, 2.75, 3.0, 3.5, 4.0, 4.5, 5, 6, 8,10, 15, 20, 25, 30, 40, 50}. The unequal spacing is due to the behavior of $(\Delta d_b)/(\Delta\phi)$, and is beyond the scope of this article.

The measurement dataset from [4,5] was obtained by scanning a phototypeset page of very large black and white corners. The corners ranged from 5 degrees to 60 degrees in 5-degree increments, making a total of 24 corners per page. The page was scanned 5 times, yielding 120 corner images. This was repeated at 17 threshold levels.

The results of the comparison between the approximations and the naïve subsystem are given in Table 1. A comparison between w_{est} and θ_{est} from these approximations and the values from [4, 5] are shown. The errors listed are the means and standard deviations of the absolute value of the differences between corresponding (w_{est}, θ_{est}) pairs of the approximation datasets and the resultant dataset from [4,5].

Of the two approximations, the Surface Interpolation method compared most favorably with the results from [4], with an error less than 5%. The Erosion Surface Offset technique yielded results with an error of about 10%.

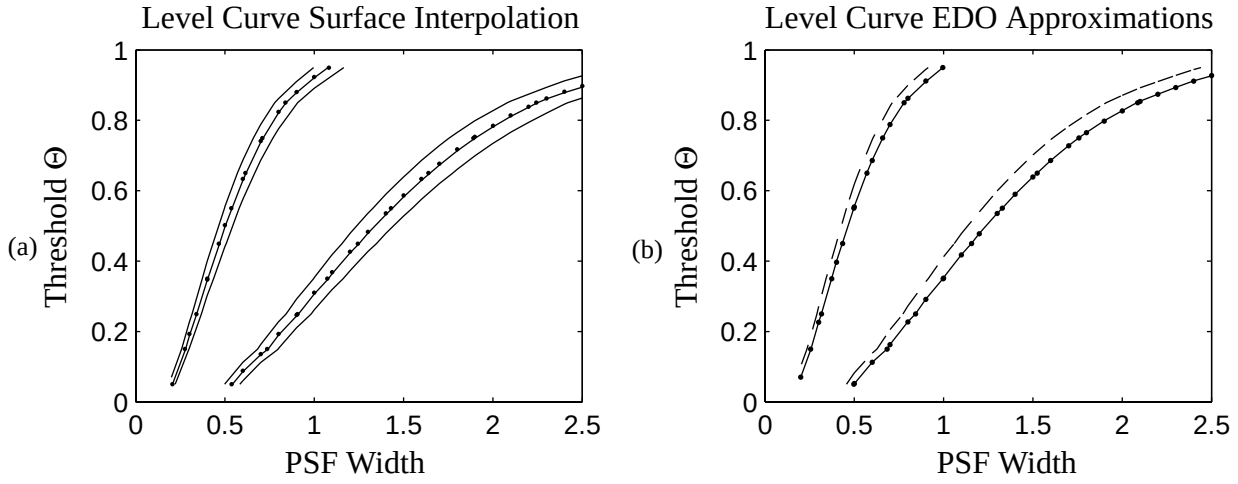

Figure 3: Comparison between actual erosion contours and approximations from (a) interpolation and (b) Erosion Distance Offset. Approximations are shown with dots that overlie the actual contour. Notice that the curves generated at even one degree difference have a significant difference.

However the surface generation is not the only difference between these experiments an the original ones. In the original experiments, to make the estimation process proceed in a reasonable amount of time, the same corner angle measurement was used for all angles that were known to have originated from a scan of the same angle, and the surface was generated with the angle measurement from a priori knowledge. Also not every combination of black and white d_b and d_w level curves were compared. The values of w_{est} and θ_{est} were calculated from the 25 intersections of all five measurements at each of the 12 angle measures, and these 12 values were averaged to produce the final estimates. The results from all these experiments closely match the values of w_{est} and θ_{est} estimated from grey level scans using the knife edge estimation technique and calibrated grey density step charts. This simplification could not be used for an arbitrary input where the angle measurements are not known a priori.

Table 1: Comparison of estimation results after implementing level curve approximations

	Surface Interpolation	Erosion Distance Offset
mean $\|\Delta w\|$	0.0358	0.0946
std $\|\Delta w\|$	0.0118	0.0267
mean $\|\Delta\Theta\|$	0.0074	0.0108
std $\|\Delta\Theta\|$	0.0044	0.0082

To compare the execution times between the naïve algorithm that finds level curves from corner measurements, each measurement in the above dataset had a level curve generated for it. For all 12x2x5x17=2040 measurements, the process completed in 91 seconds for the Surface Interpolation algorithm, and 106 seconds for the Erosion Distance Offset algorithm. Using the naïve approach, assuming 10 minutes per surface generation (about 20 minutes per surface were required to generate S), generating level curves for the same 2040 corner measurements would have taken almost 2 weeks.

5.2 Intersection Finding Algorithms

The dataset used above is too small for accurate and practical timing measurements, so another dataset was used which was derived from a phototypeset page of scanned 12-point Arial characters. Each page was scanned at 600 dpi, 8-bit greyscale. At each of 17 thresholds, corners were located in the characters and were then measured by the computer for color, angle ϕ and corner erosion distance d_b, and the results recorded.

Each partitioning algorithm may be used by itself, or with any combination of the enhancements. To observe the speedups that can be realized with each combination thereof, experiments were conducted to time each configuration. Two subsets of the measurement dataset were created, level curves generated for each measurement, and their intersections found. Two quantities of angle measurements were used to show how the speed changes with the quantity of erosion contours. In the first set of timng measurements, 97 angles were used, consisting of 52 black corners and 45 corners. The second set of timing measurements contained 487 corner measurements, consisting of 257 black and 230 white corners. The results are listed in Table 2. Times given do not include generation of level curves.

The Naïve algorithm systematically takes a line segment from a black level curve and tries to find an intersection with a line segment from a white level curve. If an intersection is found, no more line segments between those level curves are checked for intersection. The times found using this algorithm form the baseline for the rest of the timing measurements.

For the purposes of our discussion here, we limit consideration to the dataset of 487 elements. Note that the worst execution time of all the speedups presented here is still 4.6 times shorter than the best of the Naïve times (Naïve + trim). Note also that the Curve Partitioning algorithm consistently executed in less time than the Space Partitioning algorithm. Lastly, note that execution times did not change appreciably for the Space Partitioning algorithm, no matter which enhancements were used. Specifically, there is only about a 34 second difference between the best and worst times, compared to the Curve Partitioning algorithm which improved by 178 seconds after adding the enhancements. Clearly, the enhancements have a greater impact on the latter than the former.

6. CONCLUSION

Two techniques have been developed that significantly reduce the time required to find level curves and intersections from

Table 2: Timing measurements, in seconds, for intersection-finding algorithms and enhancements.

Algorithm	97 Angles	487 Angles
Naïve	111	3374
Naïve + trim	67	2134
Curve Partitioning	16	448
Curve Partitioning + trim	13	376
Curve Partitioning + rotate	13	344
Curve Partitioning + trim + rotate	9	270
Space Partitioning	11	462
Space Partitioning + trim	9	456
Space Partitioning + rotate	9	439
Space Partitioning + trim + rotate	7	428

scanned corner measurements. The Surface Interpolation algorithm required slightly less time than the Erosion Distance Offset algorithm. It is also more robust, since it relies on simpler math concepts. However, the time required to interpolate an erosion surface is very sensitive to the sampling density of the surface, determined by W and Θ, while the time required to calculate a new erosion distance relies primarily on the sampling density of the table containing the partials, determined by D and Φ. Specifically, if the Surface Interpolation method is used, one must consider the sampling density of the erosion surfaces used for the interpolation in order to minimize execution time. However, if the Erosion Distance Offset method is used, the density of the table containing offset values is an added consideration.

Two algorithms and two enhancements have been presented that reduce the time required to find intersection points between level curves. The Space Partitioning algorithm might have slightly smaller execution times than the Level Curve Partitioning algorithm, but requires much more overhead: it must fill each partition with line segments (an operation requiring multiple additional intersection checks between each line segment and its support partition boundaries), and redundant checks between line segments sharing more than one partition must be avoided. Also, optimum partition sizes are more difficult to find, DRAM consumption is larger, and it is a more complex algorithm, making it more difficult to analyze and maintain.

Using these algorithms, execution time is reduced by 3 orders of magnitude, making the subsystem practical for large numbers of corners. Further research in scanner characterization may now be more easily conducted with timely access to experimental results. Errors arising from the approximations used in the level curve finding algorithms may be abated using one of the following two methods. The simplest is to increase the density of corner surfaces and, in the case of the Erosion Distance Offset technique, the density of the sampling points in D and Φ. Additionally, or instead of the former, splines may be used (a) to interpolate the level curves between sample points and (b) in the interpolation process that finds a new erosion surface from which a level curve is found. Because level curves are quite smooth, a low-order spline should suffice to interpolate them - no larger than a cubic. The trade-off of using splines is a slight increase in execution time.

7. REFERENCES

1. S. V. Rice, J. Kanai, and T. A. Nartker, “A Report on the Accuracy of OCR Devices,” ISRI Technical Report TR-92-02, Univ. Nevada Las Vegas, Las Vegas, NV, 1992.
2. H. S. Baird, “Document Image Defect Models,” Proc. IAPR Workshop on Syntactic and Structural Pattern Recognition in Murray Hill, NJ, 13-15 June 1990.
3. E. H. Barney Smith, “Characterization of Image Degradation Caused by Scanning,” Pattern Recognition Letters, Volume 19, Number 13, 1998, pp. 1191-1197.

4. E. H. Barney Smith, “Estimating Scanner Characteristics from Corners in Bi-Level Images,” Proc. SPIE Document Recognition and Retrieval VIII, San Jose, CA, 21-26 January 2001, pp.176-183.
5. E. H. Barney Smith, *Optical Scanner Characterization Methods Using Bilevel Scans*, Doctoral Thesis, Rensselaer Polytechnic Institute, December, 1998.
6. Tin Kam Ho and Henry S. Baird, “Large-Scale Simulation Studies in Image Pattern Recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 10, October 1997, pp. 1067-1079.