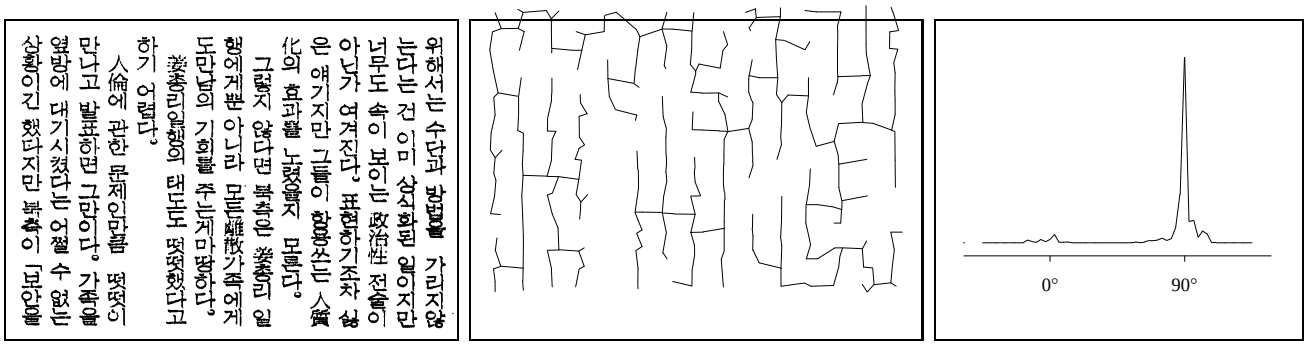

Figure 3: Left: second block from bottom on right side of Figure 2. Center: minimum spanning tree connecting centers of components. Right: Energy in coarse histogram of MST edge orientations - 90% of energy is near 90°.

5 Orientation of Text Lines in a Block

After block segmentation, text-line orientation is inferred for each block separately. Our algorithm [8] relies on the universal convention that characters are printed more tightly within a text line than between text lines.

Given the black components in the block, we proceed to (1) set aside very small and very large components, (2) idealize the remaining components as points, (3) construct the minimum spanning tree (MST) of the undirected graph defined by connecting all pairs of points, (4) analyze the distribution of MST edges to decide *horizontal*, *vertical*, or possibly *uncertain*.

The centers of bounding boxes of components surviving the size filter become the vertices in a fully-connected undirected graph. Fortunately, all its edges need not be explicitly constructed, since we can first compute the Delaunay triangulation of the vertices, which is a planar graph containing the MST. The Delaunay triangulation can be computed in $O(n \log n)$ time, and the MST can be extracted from it in time $O(n)$, where n is the number of vertices (components).

Most edges of the MST lie within, rather than between, text lines. We thus analyze a histogram of edges' orientations: if the mode of the histogram is near 0°, then we decide that the text is horizontally oriented, and of course a mode near 90° implies a vertical orientation.

In trials on over 100 pages representing a dozen writing systems, the algorithm inferred the correct orientation in 99% of text blocks; mistakes can occur on noise and tabular data. Runtime averaged 0.1 CPU s, with a maximum of 0.5 s.

Figure 3 shows the algorithm running on a block of Korean text. Korean is the most challenging, for this task, of the writing systems that we have experimented with, due to special properties of its Hangul symbol set. Hangul has 24 letters which are combined in a two-dimensional fashion to form composite syllabic symbols. Individual letters in a syllable are often disconnected. Furthermore, words are short and so there are, compared to other writing systems, a relatively large number of inter-word gaps, many of which are as large as inter-line gaps. In spite of these difficulties, we have been able to tune the heuristic to perform reliably on Hangul.

6 Isolating Text Lines in a Block

We next segment an isolated skew- and shear- corrected block of known orientation into text lines. An early version of the method, developed on the Latin alphabet, is described in [2]. We will describe it operating on horizontally oriented blocks; on vertical blocks, we have implemented an analogous method, rotated 90°. We first compute the horizontal projection, P, defined over the height of the block. In many published projection methods, P_i is the number of black pixels at height i, but we project components abstracted as rectangular boxes of the same center and area, slightly ($\times 0.75$) shrunken in height. This reduces sensitivity to writing-system-dependent details of symbol shape. Some prior methods apparently depend on the existence of gaps of white space between lines and can fail due to tight spacing, frequent diacritical marks, or small defects in the image. We have largely, but not entirely, overcome such difficulties by applying digital signal processing to P.

First we estimate the dominant line-spacing: D is the local derivative of P, which is companded by taking square roots. The autocorrelation A of D is then computed; the local maxima in A are multiplied by an exponentially-decaying factor. The surviving maxima s_{aut} is chosen as the dominant line-spacing. This complex heuristic is motivated by the great diversity of text profiles we have encountered: derivatives increase the sensitivity of the autocorrelation on short lines, companding moderates the effect of variance in line length, and decay suppresses higher-order harmonics resulting from missing lines.

S is then produced from P by convolving with a Gaussian kernel of standard-error $s_{aut}/8.0$. W is then produced by suppressing in S local non-maxima. Non-zero values of W almost always lie near the centers of maximum density of text lines, one per line. The last step is to partition the block's height. For this the original profile P is used: within each gap between adjacent pairs of line-centers, we find minima in the moving average function of P using a window 1/10 the gap height. These minima partition the block into text lines. Figure 4 illustrates the method on Thai text.

Each component, including those set aside earlier, is assigned to the text-line region in which the majority of its area lies. The line's connected components are sorted according to the reading order of the language. Components which overlap (in vertical projection) by more than 0.7 of the width of either are combined into a single symbol.

This method required 1.3 CPU s on the image of Figure 4, most of which is consumed in autocorrelation and smoothing. The method is reliable on most writing systems when the ratio of maximum to minimum text sizes within a block is ≤ 3. Mistakes typically occur in four ways: (a) diacrits may be split into a separate line or assigned to the wrong line; (b) short lines sandwiched between two long lines of the same text size may merge with one of them; (c) short lines of large text close to a block of smaller text may be fragmented; and (d) lines of small text embedded in a block of larger text may be merged. Mistake (a) is most pronounced on Thai text, which is heavily diacrited both above and below, and uses double-height diacrits.

Digital signal processing, although often complex and heuristic in practice, still seems to us the best broadly writing-system-independent method of attack on the text-line segmentation problem.