Presented at IS&T/SPIE EI'96, Conference 2660: Document Recognition III
pp. 160-174, Jan. 29-30, 1996, San Jose, CA.

Textured reductions for document image analysis

Dan S. Bloomberg

Xerox Palo Alto Research Center
Palo Alto, CA 94304

ABSTRACT

A particularly effective method for analyzing document images, that consist of large numbers of binary pixels, is to generate reduced images whose pixels represent enhancements of textural densities in the full-resolution image. These reduced images are generated using an integrated combination of filtering and subsampling. Previously reported methods used thresholding over a square grid, and cascaded these threshold reduction operations. Here, the approach is generalized to a sequence of arbitrary filtering/subsample operations, with emphasis on several particular filtering operations that respond to salient textural qualities of document images, such as halftones, lines or blocks of text, and horizontal or vertical rules. As with threshold reductions, these generalized "textured reductions" are performed with no regard for connected components. Consequently, the results are typically robust to noise processes that can vitiate analysis based on connected components. Examples of image analysis and segmentation operations using textured reductions are given. Some properties can be determined very quickly; for example, the existence or absence of halftone regions in a full page image can be established in about 10 milliseconds.

Keywords: textured reduction, image analysis, image segmentation, multiresolution morphology, image morphology, image reduction, threshold reduction, page segmentation

1 Introduction

Linear filters commonly used for segmentation of natural scenes, such as Gabor filters, have been used to segment grayscale document images.[7] Features derived from the filtered image, that represent characteristic document textures, are used in a classifier to determine text and image regions. On a per-pixel basis, these techniques are computationally intensive, and are most applicable for low-resolution images with highly variable lighting; e.g., captured by a camera. For scanned images with controlled illumination, nonlinear operations on binary images are most effective.

Accurate binary page segmentation systems are required to work at a number of different resolutions,

with the finest resolution typically in the range 75 to 150 ppi. The choice of resolution reflects a strong resource/accuracy trade-off, because the memory requirements scale as the square of the resolution, and for some operations (e.g., morphological) the time scales roughly as the third power.

Several systems for page segmentation of binary document images[4,2] have been described that analyze low resolution versions of the scanned images for efficiency. The analysis is more reliable if the subsampling of the full resolution image is preceeded by nonlinear binary filtering. The filter choices depend on the specifics of the analysis to be performed at low resolution. Most filters, such as morphological and rank order, are translationally invariant (*TI*). Each pixel in the image is modified by same rule involving pixels in its neighborhood. However, if the filtering is followed by subsampling, which is a translationally variant operation, only the pixels to be subsampled need to be altered by the filter. Consequently, it is much more efficient to combine the filtering and subsampling operations.

Studies on the relation between subsampling and morphological operations on binary images have been used to characterize the magnitude of errors (expressed as a pixel distance) obtained when reconstructing high resolution images from subsampled ones, and when operating in the subsampled domain instead of the high-resolution domain.[5,6] The reconstruction structuring elements used were small low-pass filters. Our concern for fidelity of the relationship between high and low resolution images is different. We are interested in filtering operations that enhance particular textures and de-emphasize others. These filters are highly distorting, and would be utterly useless in reconstruction applications. They are selected for both large enhancement and some degree of specificity.

Threshold reductions by 2x have been used to construct binary image pyramids, where a pixel at each level is computed from a threshold of the sum of ON pixels in the level above.[10,11] The threshold reduction operation is a combination of a specific rank order filter followed by subsampling, where the filter size is identical to the size of the subsampling tile. An efficient combination of filtering and subsampling was introducted as a cascade of 2x threshold reductions.[1] Such a cascade of threshold reductions provides sufficient texture-projection flexibility for most applications in document image analysis.[2] "Texture" is used here to refer to any pixel patterns established over a region that is larger than a characteristic distance of the pattern. This is to be interpreted broadly to describe all relations between image pixels, irrespective of the connectedness of the underlying components. Using threshold reductions, reduced images are generated quickly, and each pixel in the reduced image is determined by an (often large) set of neighbors. In such a cascade, about 70 percent of the computation occurs in the first 2x reduction. The speed of a 2x reduction on a Sun 10 is about 2 source image pixels per clock cycle, for any of the four thresholds.

In this paper we explore related but significantly faster filter/subsampling reductions. These are practical because many of the salient features in an image, such as halftones, lines, and lines of text, have an underlying texture that is manifest at resolutions well below 75 ppi. Because the action of the filtering/subsampling operation is to label output pixels by some textural property in the vicinity of the corresponding high resolution source pixels, we call the general class of such operations *textured reductions*. Thus, threshold reductions and their cascades are a particular class of textured reductions. The new textured reductions considered here are faster than the threshold reductions for two reasons: they use larger subsampling, and in some cases the filtering step is faster. For most of the applications, we use reductions to 40 and 20 ppi, thus gaining about an order of magnitude in speed over similar analysis using threshold reductions.

To segment page images, these scale-changing textured reductions must be combined with other texture-projecting nonlinear operations at constant scale. This is a classification problem, implemented entirely with image-based operations. Ideally, for each region of interest, a set of operations can be found that will project out all pixels of this chosen type (e.g., text lines) and no pixels from any other image regions.

1.1 Plan of the paper

In Section 2, the textured reductions are defined. The aliasing properties are described, and conditions for which no aliasing occurs are found. Examples of the action of the most simple textured reductions that produce no aliasing in at least one direction are given at different resolutions, to show their texture-projecting effects. Then in Section 3, we show how these operations can be used in conjunction with morphological operations to perform fast but coarse segmentation analysis. The trade-off here is mostly between spatial accuracy, which is limited by the degree of subsampling, and the amount of calculation required. Finally, in Section 4, we describe efficient implementations for these new operations.

2 Textured reduction operations

Textured reductions are translationally variant reductions, that are implemented efficiently using a translationally variant filter before subsampling. We consider only specific textured reductions that have the following properties:

- Operations are on binary images.
- All filtering operations are simply implemented as logical operations on pixels. This restriction is chosen for efficiency, supported by aliasing considerations.
- The subsampling is isotropic. This is not a significant limitation in the use of these reductions, because we can choose different horizontal and vertical filters.
- All filtering operations act only within each square subsampling tile. This is not a substantial restriction, as is shown below in the discussion of aliasing.
- The filters are separable. This simplifies the implementation with little loss of flexibility.
- The filters are identical for each tile. When the filter does not touch every pixel in a tile, the set of pixels can be varied to more randomly sample the image. This is usually not necessary.
- Subsampling is by power-of-2. This is chosen for computational efficiency only.

2.1 Aliasing

Any subsampling operation has the potential to cause aliasing, which can manifest itself as a low frequency pattern in the subsampled image (or, often, as the disappearance of image components with a small dimension.) Aliasing thus introduces a texture pattern into the reduced image that did not exist in the original image, and this can cause problems in segmentation when these patterns mimic textures that are characteristic of different document regions.

Aliasing can be mitigated or even entirely removed by lowpass filtering. We investigate here the conditions under which a binary image can be filtered before subsampling to prevent aliasing. We will see later that some alias-producing filters are desirable because (1) the artifacts can be filtered to prevent segmentation errors and (2) these filters have particularly efficient implementations.

Aliasing from a frequency component of a continuous signal occurs when the sampling distance is greater than half the wavelength. The analog in the fully discrete domain, where a discrete signal is subsampled at integer pixel intervals, is that aliasing will occur if there are any wavelengths less than twice this sampling distance. Conversely, if the image is filtered to remove these frequency components, aliasing will not occur. Consider filtering and subsampling in one direction, on $N \times 1$ tiles. After filtering, a pixel will be chosen for subsampling. We restrict the filter to act over each set of N pixels. What filters are guaranteed to prevent aliasing? Clearly if the filter doesn't alter the sampled pixel, aliasing will occur. Likewise, if the filter does not involve all N pixels, aliasing will be present for some set of signals. For example, suppose the filter does not use a specific pixel in each tile. Then a signal based only on those pixels would be missed. Consequently, to prevent aliasing it is necessary to sample all N pixels. Additionally, it is necessary to weight all pixels equally in the filter; otherwise, specific signals will be given inappropriate emphasis. We consider *rank order filters*, which are a particular subset of the class of filters that weight all N pixels in the tile equally. The output from a rank order filter is 1 if m or more of the N pixels are ON, where $1 \leq m \leq N$, and 0 otherwise.[8] Most rank order filters will show aliasing for arbitrary signals, as can seen from the following argument. Suppose the pixel pattern has an underlying half wavelength $L < N$, by which, in the digital domain for binary images, we mean a repeating pattern of L pixels ON followed by L OFF. Consider first the case when L is an integer. Inspection of a few cases shows that the number of ON pixels in each tile will vary with tile (i.e., with the phase of the underlying signal) unless $N \bmod L = 0$ and N/L is even. In general, the variation in number of ON pixels in a tile varies between $n_{min}(N, L)$ and $n_{max}(N, L)$, where

$$1 \leq n_{min}(N, L) \leq n_{max}(N, L) < N, \tag{1}$$

$$n_{min}(N, L) = \begin{cases} L\lfloor \frac{N}{2L} \rfloor & \lfloor N/L \rfloor even \\ L\lfloor \frac{N}{2L} \rfloor + N \bmod L & \lfloor N/L \rfloor odd \end{cases} \tag{2}$$

and for all L,

$$n_{min}(N, L) + n_{max}(N, L) = N \tag{3}$$

The variation in the number of ON pixels in a tile, which gives the strength of the aliasing signal, is then

$$v(N,L) = n_{max}(N,L) - n_{min}(N,L) = N - 2n_{min}(N,L) \tag{4}$$

Some values for this aliasing signal are given in the table below:

N	L	$v(N,L)$	N	L	$v(N,L)$	N	L	$v(N,L)$	N	L	$v(N,L)$
3	1	1	6	2	2	8	1	0	10	4	2
3	2	1	6	3	0	8	2	0	10	5	0
4	1	0	6	4	2	8	3	2	16	1	0
4	2	0	6	5	4	8	4	0	16	2	0
4	3	2	7	1	1	8	5	2	16	3	4
5	1	1	7	2	1	8	6	4	16	4	0
5	2	1	7	3	1	8	7	6	16	5	4
5	3	1	7	4	1	10	1	0	16	6	4
5	4	3	7	5	3	10	2	2	16	7	2
6	1	0	7	6	5	10	3	2	16	8	0

Note that the maximum variation, v_{max}, always occurs when $N - L = 1$, so that N is as close to L as possible, and is of magnitude

$$v_{max}(N,L) = v(N,N-1) = N - 2 \tag{5}$$

The *period* of the alias signal, $p(N,L)$, is conveniently defined in units of tiles to be the distance in tiles required for the signal to repeat itself. It is easily seen that p is related to the amplitude v of the aliasing signal by

$$p(N,L) = v(N,L) + 1 \tag{6}$$

So for a signal with no aliasing, $v(N,L) = 0$ and the value of the signal is equal in every tile.

If L is not an integer but is still less than the integer N, the signal (i.e., the number of ON pixels in the tiles) will vary in an aliased fashion, from tile to tile, for all values of L and in a range given by (1) and with a period given by (6). In the limit that L approaches the sampling distance N, the signal variation approaches N but the period goes to infinity. Now, if the number of ON pixels in a tile varies, and a rank order filter is used with a threshold larger than the minimum $n_{min}(N,L)$ but at or below the maximum $n_{max}(N,L)$, then the result for the tiles will vary with a low-frequency aliased signal.

The condition for filters that produce no aliasing for any underlying signal is now apparent. We must use a rank order filter that has a threshold outside these limits *for any possible underlying signal* L. This filter must have either have a rank value equal to the smallest possible value of $n_{min}(N,L)$ or larger than

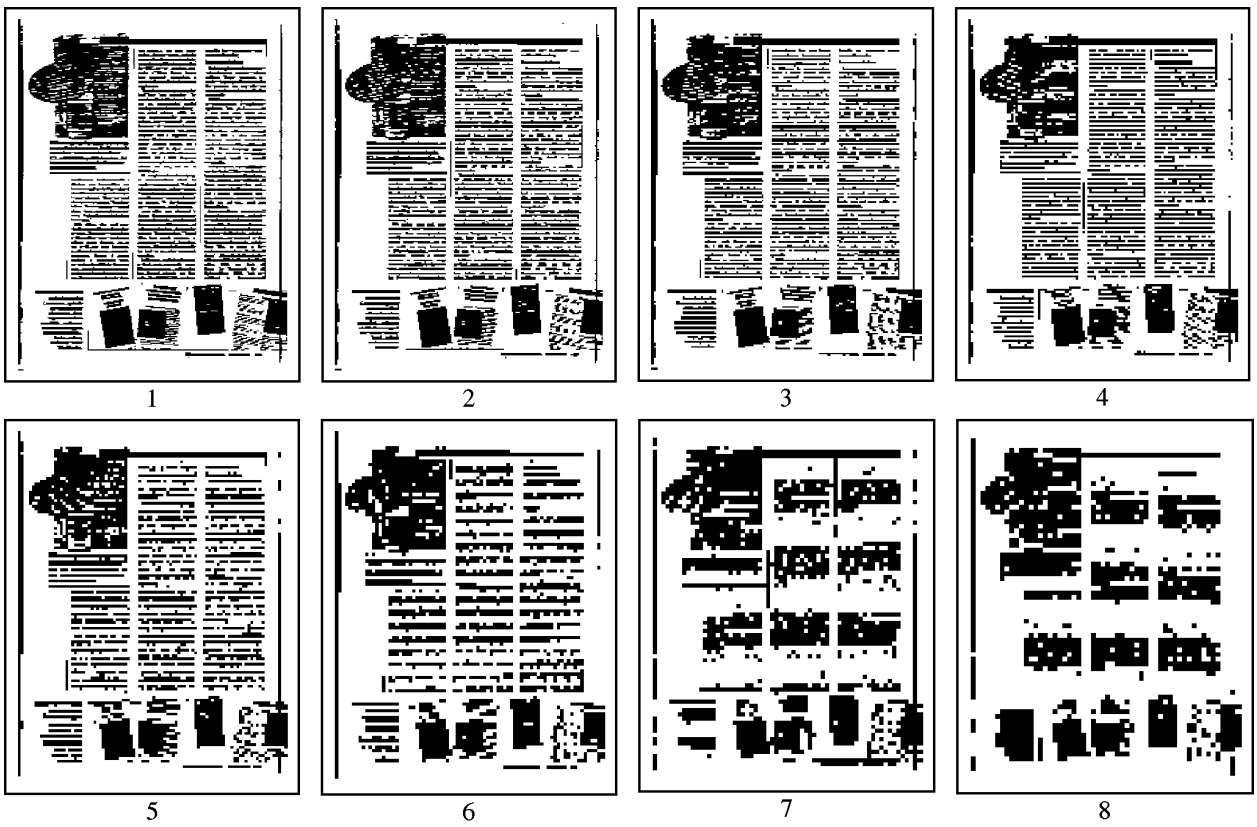

Figure 1: *Vertical textline aliasing at decreasing sampling rate.*

the largest possible value of $n_{max}(N, L)$. There are only two choices that satisfy this condition for all L: $m = 1$ and $m = N$. The threshold must thus be chosen at the extreme values, equivalent to using a dilation or erosion operator with a linear solid structuring element equal to the tile size N.

These results can be extended to two dimensions. The condition for no aliasing is when *rank filters with extreme thresholds are used over every pixel in the two-dimensional tile*. For an $N \times N$ tile, the threshold values are $m = 1$ and $m = N^2$. Additionally, for signals that vary only in the horizontal or vertical directions, separable rank filters with (possibly different) extreme threshold values in each direction are non-aliasing. This condition is applicable to document images, which tend to have the frequency distribution of the image concentrated in horizontal and vertical directions. So for example, under those image conditions a separable filter using rank value $m = 1$ in the horizontal direction and $m = N$ in the vertical direction will be non-aliasing.

To compare the performance of subsampling without filters and with non-aliasing filters, consider again a square wave signal of periodicity $2L$. Define the *Nyquist sampling rate* r_N to be when a sample is taken at every L pixels. When sampling *above* the Nyquist rate, samples are taken at intervals *less than* L pixels, and v.v. Non-aliasing filters sampled below r_N have only a d.c. frequency response: they give either black ($m = 1$) or white ($m = N$). As r_N is approached from above, there is first a loss of strict periodicity at a sampling rate of about $2r_N$, and near r_N the solid regions are extended. On the other hand, when no filtering is done, there is distortion but no loss of signal bands above r_N. Aliasing begins below r_N, but the signal first gets a significant d.c. component when the sampling rate decreases to $0.5r_N$. This is half the sampling rate when the non-aliasing filters get their (permanent) d.c. response. As the rate is lowered further, the response continues to vary.

Fig. 1 shows the effect of vertical aliasing, where only a single row of pixels is sampled in each tile. The sampling rate decreases from (1) to (8). It is well above r_N in (1), decreases to approximately r_N at (4), and $0.5r_N$ in (7). Note that the size, spacing and location of the black blocks is highly unpredictable for such very low sampling rates. However, aliasing filters are useful if we stay well above $0.5r_N$. For textlines shown here, typical vertical spacing is about 8 lines/inch, so r_N is 16/inch, and we must sample well above 8/inch to avoid highly variable results. Morphological filtering after the textured reduction, using closings and openings in the vertical direction, can be used to reduce aliasing effects. Examples are given in Sec. 2.2.

2.2 Simple filters

In this section we consider simple filters that use extreme rank filtering to avoid aliasing in at least one direction. In all cases, we consider a tile of size $N \times N$, from which a single pixel will be produced in the reduced image. Both horizontal and vertical filters can use rank of either 1 or N.

For one-dimensional filters, where only a single row or column of each tile is used, there are four possibilities. The horizontal filters of rank 1 and N that test pixels in a single row are denoted H_O, H_A, respectively. The subscripts O and A represent "OR" and "AND", because these binary logical operations between pixels in the row implement the respective rank filter. Vertical filters that test pixels in a single column are likewise denoted V_O and V_A. These are anti-aliasing in the filter direction, but aliasing occurs

in the orthogonal direction because of the subsampling.

The results from separable two-dimensional filters depend, in general, on the order of the filtering operation. There are four completely anti-aliasing two-dimensional filters, that sample the entire tile, and that do the vertical sampling before the horizontal sampling. These are denoted $D_{O,O}$, $D_{O,A}$, $D_{A,O}$ and $D_{A,A}$, where the first and subscripts indicate horizontal and vertical filtering, respectively. If it is necessary to indicate the subsampling factor (or tile size), it is placed as a superscript to the operator; e.g., $D^{16}_{O,O}$.

There are a similar set of two-dimensional filters that do horizontal subsampling first. However, $D_{O,O}$ and $D_{A,A}$ are independent of the sampling order, so there are only two additional filters, denoted $\hat{D}_{O,A}$ and $\hat{D}_{A,O}$, for which the horizontal filtering is done before the vertical filtering. Of these six two-dimensional non-aliasing filters, the first four are considerably more efficient to implement for subsampling values greater than 4, which are of most interest here. Consequently, in the examples that follow we show results of only the four D operations. The efficiency of these operations is discussed in Sec. 4.

Some of the simplest filtering operations are implicit, or nearly so, in that few or no pixel operations are actually performed. For example, H^8_O can be implemented by checking if there are any pixels on in a single row of the tile, using a machine instruction that tests if a byte has value 0.

The eight simple one-dimensional and two-dimensional textured reductions are shown in Fig. 2 for 8x, as applied to a fairly complicated page image that was scanned at 300 ppi. The reduced images are thus sampled at about 40 ppi, with reference to the original image. It is useful to consider the document image features, projected as texture patterns, that are salient in these images. In particular, different operations project vertical rules, horizontal rules, text as textblocks components, text as well-defined and connected textlines, text as small character-sized components with different degrees of separation, text as tiny dense noise, text as whitespace, halftones as solid regions, halftones as the only projected component, etc. The 2D anti-aliasing filters give particularly uniform (and different) results for the different image regions. For comparison, results sampled at about 20 ppi are shown in Fig. 3. Even at this low resolution, textures with regular characteristics and useful differences are reliably projected.

When should the one-dimensional and two-dimensional reduction operators be chosen? The one-dimensional horizontal operations are the fastest, because only a single row is filtered; all others require reading every pixel in the tile. Fortunately, the vertical aliasing caused by H_O and H_A are relatively unimportant for several important textures found in page images: halftones, text lines and vertical lines. For halftones, H_O will typically generate a (nearly) solid response when the sampling rate is below 60 ppi, regardless of the screen angle. Text lines rarely have a vertical frequency in excess of 10 ppi, so use of H_O and H_A above 20 ppi results in no appreciable vertical aliasing. Even if some vertical aliasing occurs, the result is only an inaccuracy in the observed text line frequency. The texture of the reduced image retains the appearance of text lines, along with its textural signature, given by sets of horizontal raster lines with ON pixels followed by lines without ON pixels. Finally, vertical lines that are longer than the tile size will be projected out by H_O and H_A without loss of information, compared to the more expensive use of $D_{O,A}$ or $D_{A,A}$.

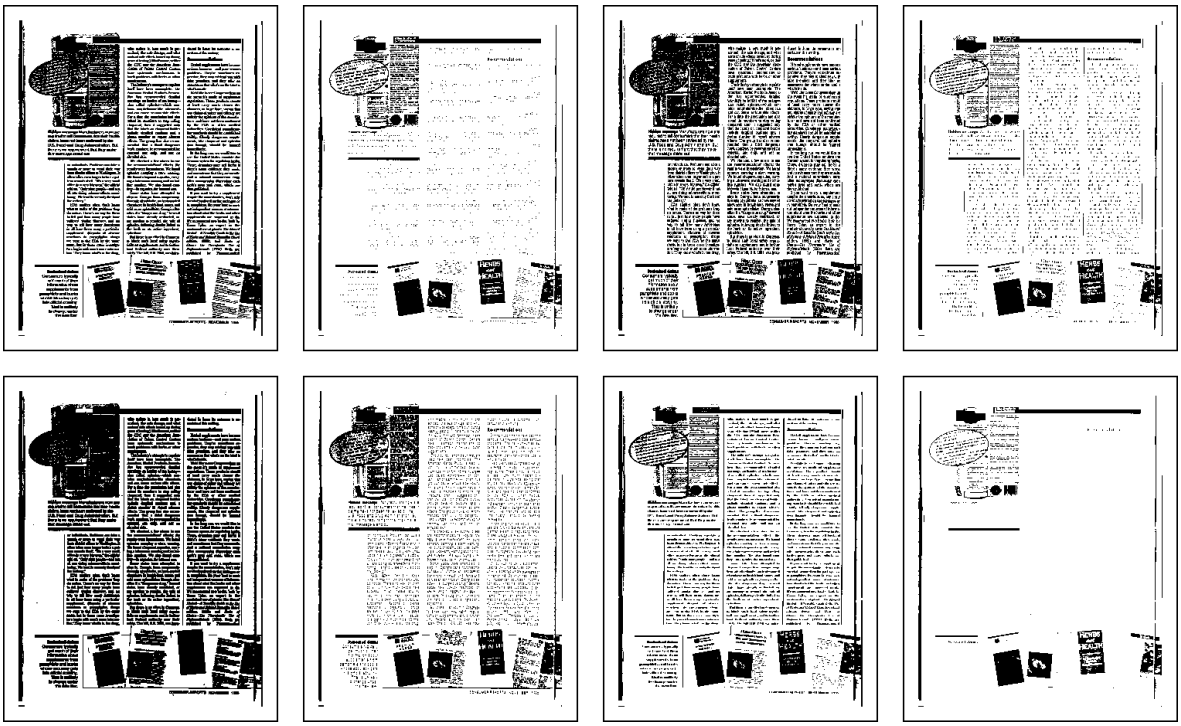

Figure 2: *The eight most simple 8x textured reductions, applied to a 300 ppi image. In the first row, H_O^8, H_A^8, V_O^8 and V_A^8. In the second row, $D_{O,O}^8$, $D_{O,A}^8$, $D_{A,O}^8$ and $D_{A,A}^8$.*

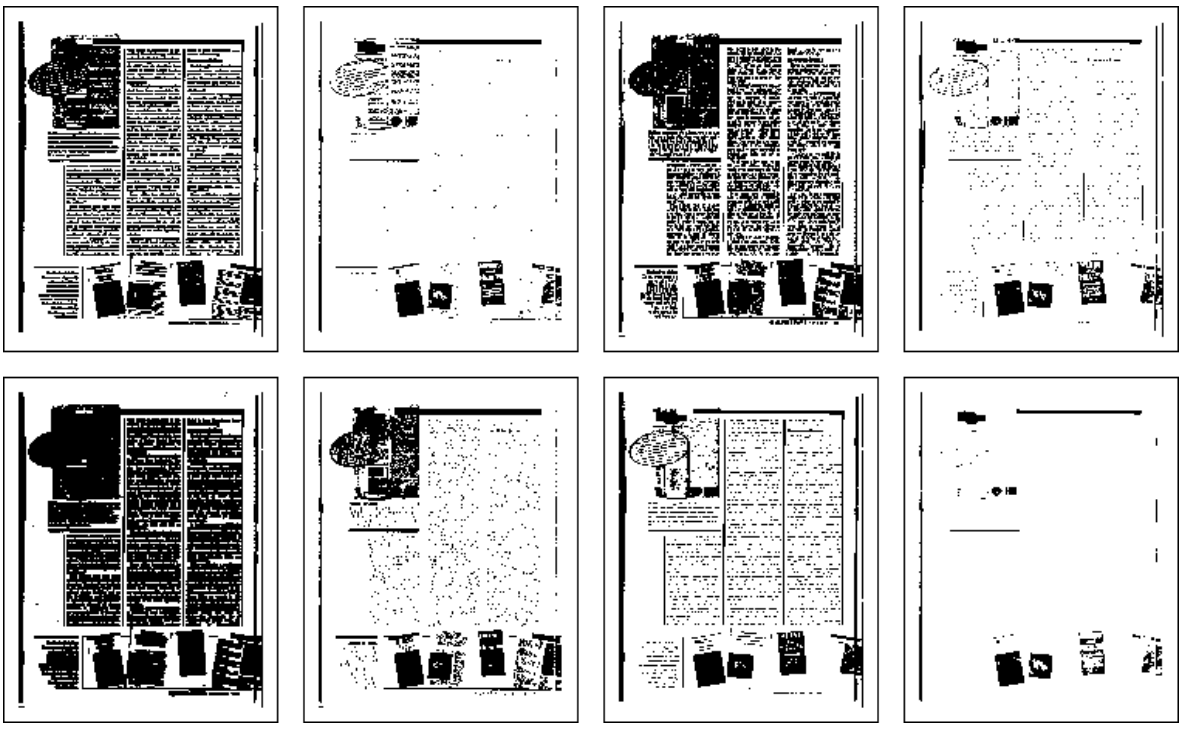

Figure 3: *The same eight textured reductions as in Fig. 2, but here at 16x reduction (about 20 ppi). The H^{16} and V^{16} operators are in the first row; the D^{16} in the second.*

3 Image analysis

In conjunction with morphological filters,[9] textured reductions can be used to segment pages into regions of different texture, such as halftones, line art, and text. In this section, examples are given for these operations. The results are not as good as with carefully cascaded threshold reductions, but here we interested in the types of qualitative information that can be obtained quickly.

3.1 Use of the simple filters

The quality of the analysis depends on the subsampling, with respect to the actual scanned image. For example, results on a 600 ppi image using a 16x textured reduction should be similar to those on a 300 ppi image using an 8x reduction, because both project out texture at approximiately 40 dpi. Consequently, in this section, we describe the results in terms of the sampling frequency with respect to the original scan in inches, rather than giving the actual reduction factor used. Also, to describe the morphological operations in a way that is independent of the original scanning resolution, the approximate size of each structuring element is given in inches on the original page. For example, a 1.0 inch SE on an image subsampled at 20 ppi is 20 pixels long.

Fig. 4 shows some analysis at 40 ppi. The images are numbered from left to right and top to bottom. (1) is a reduced representation of the original image; we produced it with a sequence of level 2 threshold reductions to give a good visual appearance. (2) is produced with V_O, followed by a large horizontal opening (1.2 inch). Likewise, (3) is produced with H_O followed by a large (1 inch) vertical opening. The halftone seed in (4) is made with $D_{A,A}$, followed by a very small square opening (0.1 inch). The halftone clipping mask in (5) is made with $D_{O,O}$. A filling operation from (4) into (5) gives the halftone regions in (6). The actual horizontal lines in (7) are found by subtracting (6) from (2), and the vertical lines in (8) are found by subtracting (6) from (3). (9) shows the vertical whitespace, found by photometrically inverting (1) and doing a large (1.2 inch) vertical opening. The rough text regions in (10) are found by subtracting (6) from H_O and closing with a moderately small (0.25 in) horizontal SE. This is cleaned up in (11) by subtracting out the halftones (6) again, as well as the horizontal and vertical lines; the result is opened with a moderately large (0.5 inch) horizontal SE, the vertical whitespace is subtracted, and then opened again with a small (0.2 inch) horizontal SE. This sequence makes reasonably nice text regions for all of the images tested. Finally, the text regions can be blocked up in (12) by opening with a tiny (0.05 inch) vertical SE to remove noise, and then doing a closing/opening with a small (0.2 inch) vertical SE.

These results can be compared with similar ones in Fig. 5 at 20 ppi. At lower resolution, there is more noise, particularly in the horizontal and vertical lines that are extracted. At 40 ppi, these lines correspond exactly to the rules in the original image, but at 20 ppi, extraneous lines and noise are introduced.

Extraction of halftone images, horizontal and vertical lines, and text lines is quite reliable at 40 ppi. Typical results for these elements are shown in Fig. 6, for images with diverse layout characteristics and using the filter sizes and sequences described for Fig. 4.

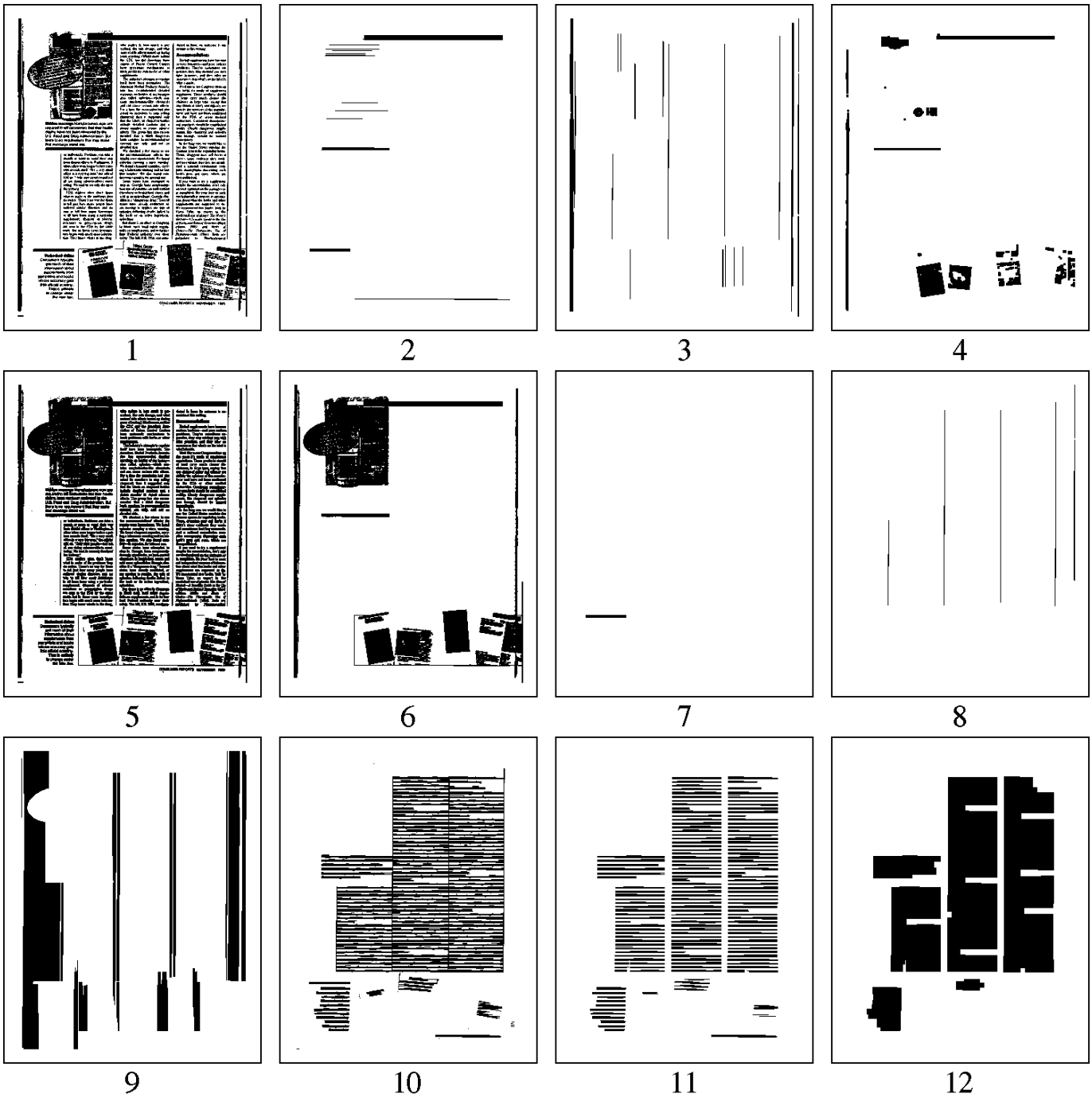

Figure 4: *12 steps in component analysis at 40 ppi*

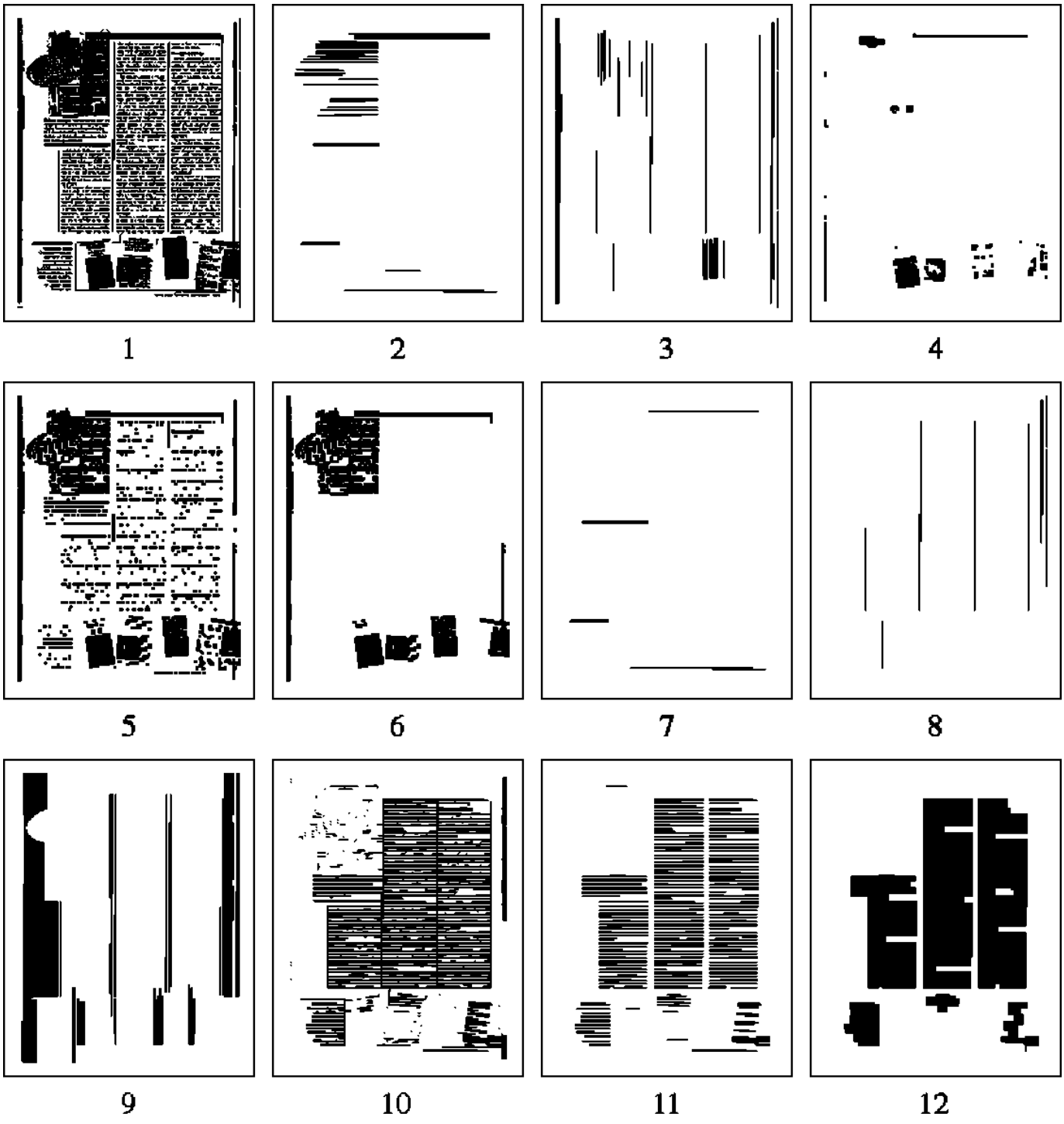

Figure 5: *12 steps in component analysis at 20 ppi*

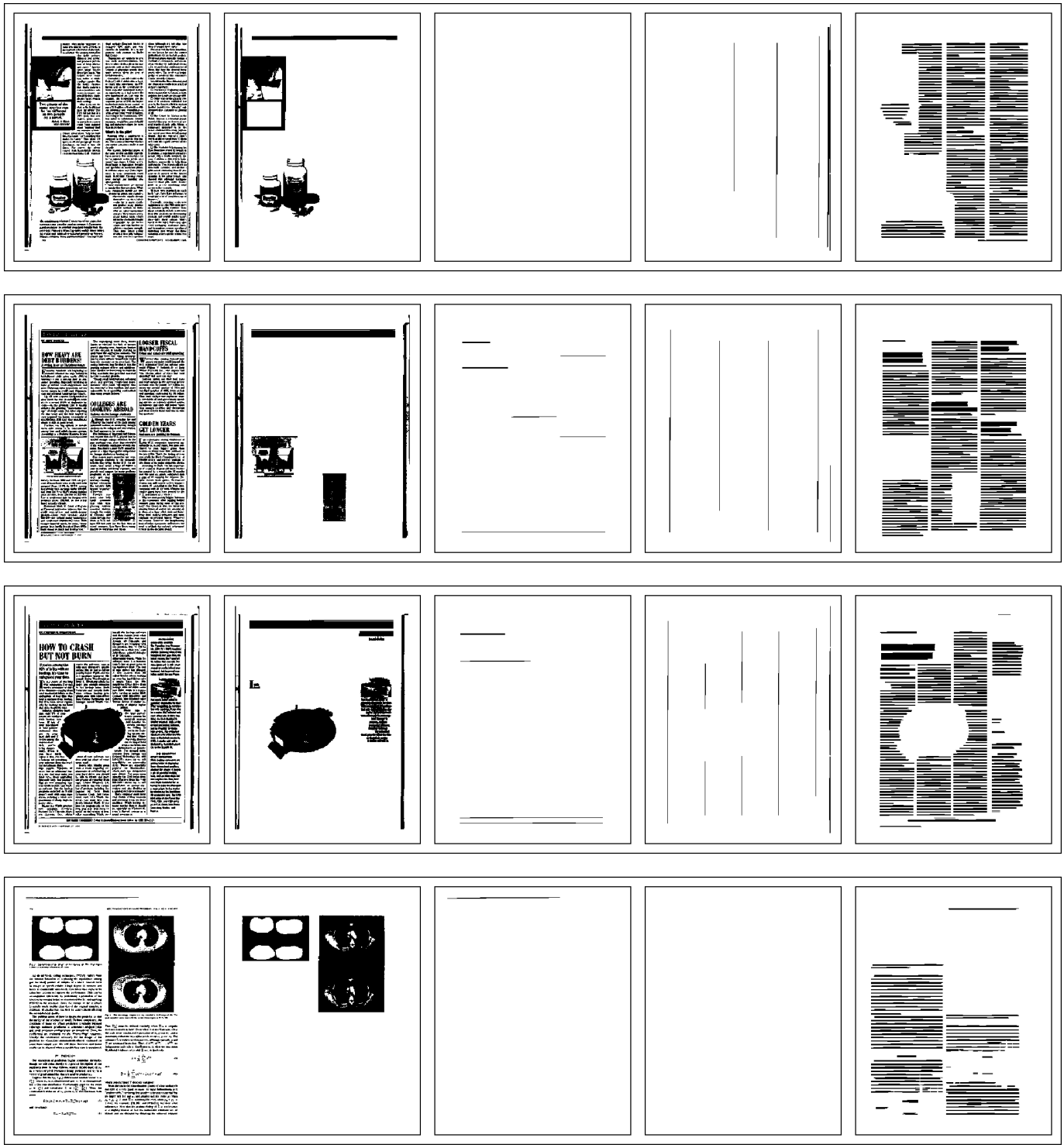

Figure 6: *Four examples from images with diverse elements and layout, at 40 ppi. For each the input image and derived halftone regions, horizontal lines, vertical lines, and text lines are shown.*

3.2 Use of the horizontal filters

The horizontal filters H_O and H_A have a special role by virtue of their speed (see Section 4 for details). Because there is no vertical filtering, they exhibit vertical aliasing from vertical structure with frequency above the Nyquist limit. Nevertheless, they are useful for a variety of tasks:

- **Halftone detection**. Halftone screens are usually tilted, and have screen frequencies of 60 ppi or greater, corresponding to Nyquist frequencies well above 100 ppi. With subsampling at or below 40 ppi (far from the Nyquist limit), aliasing can occur, but with small amplitude. As a result, halftone regions typically give a solid response. Use H_A, followed by a close/open combination to remove noise.
- **Textline detection**. Textlines typically have a vertical frequency less than 10 ppi, so no aliasing is expected for sampling above 20 ppi. See, however, the aliasing example in Section 2.2. The result of H_O is horizontal lines, that may contain inter-word gaps above 30 ppi. These gaps can be solidified by a small horizontal closing. Textlines are easily distinguished from halftone regions; because of the vertical line texture, they can be removed with a small vertical opening.
- **Vertical rule detection**. These are projected by H_O without loss, and can be distinguished from textlines with a small vertical opening.
- **Horizontal rule detection**. Rules of width greater than the sampling distance will be preserved by H_A, and preferentially projected by H_A followed by a large horizontal opening. Narrow rules can be lost due to vertical subsampling. However, if the H_A reduction is done on *different* raster rows in each tile within a horizontal row of tiles, thin horizontal rules will be projected from those tiles where the sampling coincides with the rule, resulting in a broken line.
- **Multiple column detection**. Use H_O, followed by bit inversion and a vertical opening. This will leave vertical lines in the left and right margins, as well as between any text columns in the image.
- **Text orientation**. H_O can be used to detect text of either portrait or landscape orientation. Under H_O, landscape text is rendered into broken vertical lines, where the words and often the characters are separated. A vertical opening will remove this texture, but preserve vertical rules. Alternatively, vertical runlength analysis can be used to identify this salient textural characteristic.
- **Image processing for skew detection**. When using bitmap variance statistics for analyzing skew[3] on reduced images, H_O is particularly effective for subsampling because it emphasizes the horizontal nature of the text-lines, as is apparent from the results in Fig. 2 and Fig. 3. This emphasis boosts the differential signal, allowing accurate skew detection from a smaller number of textlines.

4 Implementations

After filtering (horizontal, vertical or both), rows and columns must be subsampled. Column subsampling is the most expensive, and requires some care. Some generic and conflicting guidelines can be given

for efficient implementation of image operations on a general purpose 32-bit computer:

1. Operate on 32-bit words to the maximum degree possible. The goal is to retire source or destination words with the minimum number of operations. For power-of-2 reductions, 32-bit source words are compressed into 16 or smaller bit destination chunks.

2. Register operations are very fast. Maximize use of register shifts and register logical operations. The most common operations are mask (logical AND), shift, and OR.

3. Consider using only small tables (256 or less). Small tables can be made quickly when needed, and have a much higher cache hit rate than, for example, 16-bit tables. This can compensate for the larger number of table lookups required for smaller tables.

4. Minimize table lookups. Table lookups are relatively expensive. The number of lookups is minimized by using all bits in the table.

5. Unroll loops involving 32-bit writes to the destination. Unrolling loops reduces the number of test/branches, and can be used to minimize writes to memory.

6. Minimize writes to memory. The cost of writing to memory is highest on systems such as the Sun SPARC, that write through the cache and take 11 cycles.

Within these guidelines, several efficient implementations of column subsampling possible. We mention two different types. In both, results for at least 32 bits of destination image are always stored into a register before being written to memory.

The first method typically determines several destination pixels at a time, and is more efficient for smaller tiles, such as 2×2 or 4×4. The source words are optionally filtered using intra-tile shifts and logical operations to set the value of the pixel in each tile (or tile row) that will be subsampled. For example, H_A uses AND operations between shifted pixels, without masking. Subsampling is then done by first applying a comb mask with the subsampling periodicity to project these pixels in a source word. The word is then successively shift/ORed to make all subsampling pixels contiguous. For example, a 4x subsampling requires two sequential shifts (of 15 bits and 6 bits) to compress the 8 selected pixels in a 32-bit word into a contiguous byte. A 256-entry table lookup permutes them back into the original order. Four of these operations can be placed in-line to construct the pixels in a full destination word. Threshold reduction[1] by 2x (or greater) can be implemented in this way. The filtering part is very fast; most of the computation is in the subsampling (the extraction of the filtered pixels). Consequently, a 2x *textured* reduction is not significantly faster than a 2x *threshold* reduction. [1]

The second method uses no table lookups, and it is more efficient for non-aliasing textured reductions with tile sizes 8×8 or larger. Source words are optionally filtered, and then progressively shifted, masked, and tested. Consider the H_O and H_A operations. No explicit filtering is necessary. For H_O, the pixels in successive tiles are projected from the source word (by shift and mask). This is tested and if nonzero, the

[1] In fact, the 2x threshold reduction for levels 1 and 4 are identical to $D_{O,O}$ and $D_{A,A}$, respectively, and the level 2 and 3 threshold reductions are equivalent to $(D^2_{O,A} \cup D^2_{A,O})$ and $(D^2_{O,A} \cap D^2_{A,O})$, respectively.

corresponding pixel is turned on in the reduced destination. For H_A, the projected tiles are compared with the mask, and if equal, the destination pixel is turned on. All these operations on source and destination words take place in registers.

If filtering is required, this is done by a shift followed by a logical operation. Suppose instead of H_O we wish to check whether either of two specific pixels in each tile are ON. First shift and OR to put the result for each tile in a single pixel of that tile. Then carry out the shift/mask/test procedure given above, using a mask that covers only the result pixel, not the entire tile.

We see now why the large tile two-dimensional textured reductions are more efficient when the vertical filtering preceeds the horizontal filtering and subsampling. The reason is that the vertical operations, requiring only logical operations between 32-bit words, are much faster than the shift/mask/test horizontal ones that do the subsampling.

The speed of the H operators on a 40 MHz Sparc 10 is given in the second column of the following table. For example, an H_A^{16} reduction on an 8.5×11 inch page scanned at 300 ppi takes 8 milliseconds. This can be followed by a quick erosion of the reduced image to make a decision if any halftones existed in the original 8 million pixel image. The third column gives the number of source pixels retired in each machine cycle.

Reduction	*Source Mpixels/sec*	*Source pixels/cycle*
8x	270	7
16x	1000	25
32x	3200	80

5 Summary

A special class of binary image reduction operations, that includes filtering, has been introduced. We have derived conditions under which no aliasing occurs in at least one direction, and described the special textured reduction operators that work under these conditions. These operations can be implemented efficiently because the filtering and subsampling operations are combined. Filtering operations for which no aliasing occurs require sampling over all pixels in each tile. When sampling over the tiles is not complete, some degree of aliasing is introduced. This is often allowable, because for particular types of texture found in document page images, errors in projected texture can be removed in later processing stages. The engineering trade-off with incomplete sampling is between introduction of aliasing and faster implementations. We have shown that fast textured reductions can be used to obtain quickly page segmentation information describing the salient textural regions of page images. For more accurate page segmentation implementations, it is often useful to have information such as whether or not there are halftone images and text regions, and if there is text, if it exists in single or multiple columns. For these applications, the H_O and H_A textured reduction operators are the most interesting because of their efficient implementations.

6 REFERENCES

[1] D. S. Bloomberg, "Image analysis using threshold reduction," *SPIE Conf. on Image Algebra and Morphological Image Processing II, Vol. 1568*, San Diego, CA, July 1991, pp. 38-52.

[2] D. S. Bloomberg, "Multiresolution morphological analysis of document images", *SPIE Conf. 1818, Visual Communications and Image Processing '92*, Boston, MA, pp. 648-662, Nov 18-20, 1992.

[3] D. S. Bloomberg, G. E. Kopec and L. Dasari, "Measuring document image skew and orientation", *SPIE Conf. 2422, Document Recognition II*, San Jose, CA, pp. 302-316, Feb 6-7, 1995.

[4] P. J. Bones, T. C. Griffin, and C. M. Carey-Smith, "Segmentation of document images," *SPIE Symp. on Electronic Imaging Science and Technology, Vol. 1258*, Feb. 1990.

[5] R. M. Haralick, C. Lin, J. Lee, X. Zhuang, "Multi-resolution morphology," *Int. Conf. on Computer Vision, London*, pp. 516-520, June 1987.

[6] R. M. Haralick, X. Zhuang, C. Lin and J. Lee, "Binary Morphology: Working in the Sampled Domain," *CVPR '88, Ann Arbor, MI*, pp. 780-791, June 1988.

[7] A. K. Jain and S. Bhattacharjee, "Text segmentation using Gabor filters for automatic document processing," *Machine Vision and Appl*, Vol 5, pp. 169-184, 1992.

[8] P. Maragos and R. W. Schafer, "Morphological Filters - Part II: Their Relations to Median, Order-Statistic, and Stack Filters," *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-35, pp. 1170-1184, Aug. 1987.

[9] J. Serra, *Image Analysis and Mathematical Morphology*, Acad. Press, 1982.

[10] S. L. Tanimoto, "A hierarchical cellular logic for pyramid computers", *J. Parallel and Distributed Computing*, Vol 1, pp. 105-132, 1984.

[11] S. L. Tanimoto, "Paradigms for pyramid machine algorithms", in *Pyramidal Systems for Computer Vision*, ed. V. Cantoni and S. Levialdi, NATO ASI Series, Vol. F25, pp. 173-194, Springer Verlag, 1986.