Extracting Lines with a Reconfigurable Mesh Parallel Processor [1]

Katja Daumüller, Charles C. Weems and Alan R. Hanson

Image Understanding Architecture Laboratory
Department of Computer Science
University of Massachusetts, Amherst MA 01003

Technical Report 94-59
December, 1994

Abstract

In this paper, we present an algorithm for extracting straight lines from a grey-level image. The algorithm is region-based, where each region is the underlying structure for one line, but only pixels in the region with highest gradient magnitude contribute to the line. The algorithm is optimized for both quality and speed, and time was optimized for runs on the low-level part of the heterogeneous Image Understanding Architecture (IUA), where it operates in nearly frame rate. Line output of the algorithm was input to a performance evaluation algorithm, whose results, together with timing experiments, show that the algorithm has a high performance/quality ratio as compared with two other algorithms.

[1]This work was supported in part by Army Research Laboratory contract DAAL02-91-K-0047

Contents

1 Introduction **3**

2 Related Work **5**

2.1 The Plane Fit Algorithm 5

2.2 The Principal Axis Algorithm 6

2.3 Other Approaches 7

3 The Subregion Algorithm **10**

4 Experiments **24**

4.1 Quality Measurements 24

4.1.1 The Performance Evaluation Algorithm 24

4.1.2 Comparison of Line Output 26

4.1.3 Performance Evaluation Results 36

4.2 Time Analysis 42

4.2.1 The Sequential Implementation 42

4.2.2 The Parallel Implementation 43

4.2.3 Timing Results 44

4.3 Quality versus Timing Measurements 46

5 Analysis and Conclusions **48**

1 Introduction

Straight line extraction algorithms serve as input to other algorithms in a variety of vision applications, and the line quality largely determines the usability of these algorithms. Object recognition, for example, relies to a great extent on orientation information. Heavily distorted lines make object recognition impossible. Another problem for object recognition is false positive lines, especially if the lines are not just short lines attributable to noise. The quality of the lines not only affects the quality or success rate (in this case) of the application, but can also affect its speed. High fragmentation, for example, may increase the run-time. This is especially crucial if the run-time depends exponentially or polynomially on the number of lines. For example, the model-matcher in [1] runs in order quadratic time over the number of lines.

Because of these effects, most line extraction algorithms ([2, 3, 10, 18, 20]) are explicitly designed to produce the best quality possible, often with a sacrifice of execution speed. Slow line extraction algorithms, however, are a bottleneck for real-time applications and are also an impediment in an explorative and experimental setting. Fast line extraction algorithms, on the other hand, tend to optimize primarily for speed, but with low quality. We introduce an algorithm which runs at a speed comparable to a previous fast algorithm [8] (henceforth called Principal Axis Algorithm), but whose output quality is comparable to the algorithm in [3] (henceforth called Plane Fit Algorithm). Our straight line extraction algorithm (henceforth called the Subregion Algorithm) adopts basic principles from the Plane Fit Algorithm. For optimization of speed, some features are introduced that are similar to features in the Principal Axis Algorithm.

The Plane Fit Algorithm is a region-based algorithm, which first creates two sets of regions and then fits a plane to the slope in each region. As all pixels of one region contribute more or less to the line, these regions are called support regions.
The Principal Axis Algorithm was implemented as a fast sequential version of the Plane Fit Algorithm. It also adopts the region scheme from the Plane Fit Algorithm, but does all the computation only once and has a simpler approach for computing the line from the regions.

The Subregion Algorithm was developed as a parallel algorithm on the low-level

part of the IUA [19], and design decisions were made to optimize speed on the IUA. The IUA is a heterogeneous machine with a low-level SIMD mesh which has reconfigurable broadcast buses. The reconfigurable broadcast buses offer hardware support to broadcast a value or select the minimum/maximum value in a region quickly, which is a valuable enhancement for region-based processing. On the other hand, care must be taken to reduce the number of floating points operations as much as possible because of the lack of a floating point co-processor.

After having developed an optimized algorithm on the IUA, a sequential version was developed, which turned out to have comparable performance to the Principal Axis Algorithm on sequential machines.

The algorithm itself adopts the region scheme from the Plane Fit Algorithm, but replaces the plane fit with a least squares fit of selected points. It computes the line of the underlying region by first splitting the support region into three parts in rough gradient orientation, and then selecting points of highest gradient magnitude in each subregion. One advantage of this scheme is that the parameters of the line depend less on the shape of the region than in the Plane Fit Algorithm. In our approach, the lines are computed from only one set of support regions, in manner similar to the Principal Axis Algorithm.

We evaluate the output quality of these algorithms with the help of a performance evaluation method from the University of Washington [7, 12, 13]. Experiments were run on three diverse image sets; the groundtruth having been received from the University of Washington for one of them. For the sake of classifying the algorithm with respect to quality and timing, the same experiments were also run for the Plane Fit and the Principal Axis Algorithm, and the performance evaluation method was applied to the results. The paper next describes the related work, then introduces the Subregion Algorithm in detail, and last shows results of the performance evaluation, along with timings.

2 Related Work

2.1 The Plane Fit Algorithm

Burns' Plane Fit Algorithm [3] aims at extracting long, straight lines, not necessarily of high contrast, which might have fluctuations in the gradient magnitude along their length. Therefore, the first evidence for a line is determined by the gradient orientation. Pixels are clustered in *support regions* if they are spatially adjacent and if their gradient orientation is roughly the same. Computation of the gradient orientation was done with a 2x2 mask, which achieves more sensitivity to detail and has a symmetrical response to rotation of the line.

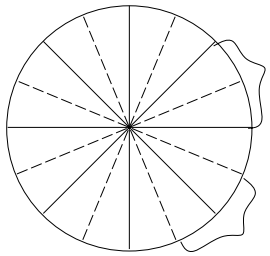

Figure 1: Region Partitioning Scheme with Overlapping Bucket Scheme.

Support regions are built according to a fixed partitioning scheme of the angle domain, see Figure 1. This has the inherent disadvantage that lines whose orientation lie on the borders of the slice in Figure 1, may have fragmented support regions. Burns avoids this by introducing the method of overlapping buckets. A second fixed partitioning scheme, which equals the first one, but rotated by half of the bucket size, creates a second set of support regions. The line parameters are computed with plane intersections of the weighted fit to the intensity values and the horizontal average pixel intensity plane, within a support region. The weight favors intensity values of pixels with high gradient magnitude. In the end, the redundant support regions are removed with a voting scheme,

where each pixel votes for one of the two associated support regions with the longer line.

Taking primarily the gradient orientation as evidence for a line and using the plane fit method, the algorithm actually extracts long, straight lines as well as shorter lines (see section 4.1 for results). In some cases, the orientation is slightly distorted, because the calculation of the line parameters takes into account every pixel in the region. The time performance (see section 4.2) is about 6 to 10 times slower than the Principal Axis Algorithm. This is likely due to processing the double SRRs and also indicates that the plane fit itself is computationally expensive, which could be caused in part by its use of 4 trigonometric functions and one square root function for each plane fit.

2.2 The Principal Axis Algorithm

Kahn and Kitchen's Principal Axis Algorithm [8] focuses primarily on speed optimization.

For the gradient magnitude and orientation, it uses a 3x3 mask because, [8] argues, the 2x2 mask produces errors that are too large in the gradient orientation [9]. The scheme of support regions and overlapping buckets is adopted from the Plane Fit Algorithm. For speed optimization, it uses a table lookup for the tangent. The algorithm merges the two SRRs obtained from the overlapping bucket scheme into one before the line extraction, so that the line extraction has to be performed only once. This is implemented according to the following rule:

A pixel is linked to an adjacent pixel if it either falls into the same bucket in the first partitioning scheme or if it falls into the same bucket in the second partitioning scheme.

The drawback of this rule is that pixels with significantly different gradient orientation can be classified into the same region, as a result of a gradual change in gradient orientation. The resulting support regions therefore may not represent the supporting pixels of an actual line.

The line extraction is implemented using the principal axis of the support region, which is less costly to compute than a plane fit. The orientation of the principal axis is computed using the small eigenvalues from a scatter matrix (which takes into account the gradient magnitude of each pixel). This provides the orientation of the line, whereas the position is determined by the centroid of the support region. The endpoints of the line are obtained by intersecting the line with the bounding box of the support region. With this scheme, the line extraction (including the computation of the endpoints) still needs three trigonometric functions and one square root function, but has overall less computation.

Results show that the quality of the Principal Axis Algorithm is not sufficiently high for e.g. tracking lines consistently or recovering good orientation estimates (see section 4), which can be either due to the merging scheme of the SRRs (results show that lines are fragmented more) or due to the computation of the principal axis (the orientation is sometimes very skewed).

2.3 Other Approaches

We review some of the more widely referenced line algorithms in this section and explain the choice of the three that are used in this study. Most other line extraction algorithms base their approaches on the linking of edge pixels from the edge extraction algorithms of either [16], [5, 7] or [10]. We briefly review these edge extractors here. The Marr-Hildreth operator of [16], for example, convolves the image with the Laplacian of a Gaussian, and then declares step edges to occur at the zero-crossings of the convolved image. The Canny operator [5] first smoothes the image by convolving it with a two-dimensional Gaussian and then finds step edges at maxima of the smoothed gradient direction. Haralick [7] uses a similar method, but finds the gradient with fits to parameterized surfaces. The Nevatia-Babu edge extraction algorithm [10] uses the output of directional edge detection, i.e. convolution with several different masks and determines the gradient orientation from the highest responding mask. A threshold of the gradient magnitude in this direction determines if the pixel is an edge pixel. After that, the edges are thinned.

The edge pixel map of these edge extractors may be precise, but in order to obtain endpoints for straight lines, further evaluation has to be done, which, in case good quality should be achieved, is equivalent to the effort after performing a convolution in the Plane Fit and Principal Axis Algorithms.

The Boldt line algorithm [2] uses zero-crossings of the Laplacian to form edges. Each zero crossing point creates a one-pixel wide line segment, whose gradient and position are determined by interpolation. After that, Boldt uses collinearity, proximity, gradient orientation and similarity of contrast as measures for merging lines. Lines are linked at different hierarchies until all parameters lie within the ranges of the measures. The final replacement is done with a least squares fit. Although the algorithm produces very good results, it is computationally expensive. The is in part caused by the zero-crossings of the Laplacian, which recover high-frequency data, but do not consistently follow the boundaries of an edge, and therefore produce fragmented boundaries that require many line combinations to be tested.

The Nevatia-Babu [10] line-finder takes the input from the Nevatia-Babu edge extractor. It links edge points according to predecessor- successor structures, and similarity in orientation. The edge segments are then approximated by a series of piecewise linear segments. One-pixel gaps in the edge segments can be bridged.

Other algorithms take Canny's edge point image as input. For example, the line extraction algorithm in [18] links the edge points of Canny's algorithm according to templates of pixel constellations of edge orientations. Bridge pixels can link fragmented segments, and special templates extend lines. The endpoints of the segments are the endpoints of the lines. In the end, a collinearity measure is applied to merge two lines into one with the two outer endpoints of the old line.

The following algorithms are two of the few examples of parallel and fast implementations of straight line extraction algorithms in the literature. The algorithm in [14] uses the Canny edge point image to link the edge points according to their difference in orientation and determine lines as endpoints of linked segments in a single pass with special-purpose hardware. The implementation on the Connection Machine [17] also takes Canny's edge point image and links boundary segments with a parallel

'pointer-jumping' algorithm. Then a parallel curve decomposition algorithm breaks the boundaries into straight lines. In the end, a least squares algorithm puts the line into its final position.

The algorithms above show that achieving lines with little fragmentation and good orientation must include special cases and remedies to deal with the edge pixel map:

- The edge points could miss one or several pixels due to noise or low contrast. If the pixels are in the middle or the ends of an edge pixel chain, bridge or extension mechanisms have to be implemented. (as in [10, 18]) This is more likely to happen with edge detectors which take into account the gradient magnitude.
- If edge pixel segments are to be merged or broken up into lines, a least squares fit (as in [2, 17]) is more accurate than just using the outer endpoints or the new endpoints at the breakpoints.

Implementing the first approach requires the use of larger windows, which could slow down the algorithm or require an extra pass over the image. Implementing the last approach also requires extra computation. Note that in the fast algorithms [14, 17], no bridging or extending of edge segments is implemented, and only in [17] is the least squares method applied.

Our approach aims at maintaining the quality of the Plane Fit Algorithm and replacing expensive methods with fast, equivalent ones. In our algorithm, as well as the Plane Fit Algorithm, line pixels which are not maxima of gradient magnitude, but have roughly the same gradient orientation, are not considered a special case. Also, in our algorithm as well as the Plane Fit Algorithm, the final line is approximated with a least squares fit method.

In order to provide for an upper and lower quality bound, our approach is compared to the Plane Fit and the Principal Axis Algorithm respectively throughout the paper. Also, as the two algorithms have the same support region scheme as our approach, reasons for differences in quality can be explained in more detail.

3 The Subregion Algorithm

The algorithm can be described in three basic parts, which are the parts structurally different from the Plane Fit Algorithm. The 'input' and 'output' to these basic parts are standard gradient magnitude and orientation computations and a line parameters, with which the endpoints are computed. There are also minor changes to achieve speed computing the input representations and continuing from the output representations. The three steps of the algorithm are:

1. Build the support regions with the overlapping bucket scheme and merge them into a single set of support regions. – The Plane Fit Algorithm is operating on two sets of support regions, the Principal Axis Algorithm on one also.

2. Split the support regions into at least three parts, taking the gradient orientation of one arbitrary pixel as the orientation of the split. – This step is not necessary in the Plane Fit Algorithm or the Principal Axis Algorithm.

3. Select n points of highest gradient magnitude in the subregions of the support regions. Compute with line parameters in a least squares fit over the positions of the 3 x n points. – The Plane Fit Algorithm computes the line with a weighted least squares fit to the intensity values of the pixels, the Principal Axis Algorithm computes the orientation with a principal axis (which is computed with a Scatter Matrix) and the position of the line with the centroid of the support region.

From the historical development point of view, the Plane Fit Algorithm set the desirable level of quality and gave the basic steps of the algorithm. The Principal Axis Algorithm provided some fast techniques. Either parts of the fast techniques of the Principal Axis Algorithm were adopted and improved, or new techniques developed, striving to be as high in quality as the Plane Fit Algorithm at the particular step in the algorithm. The notion of 'fast' was set by the speed on the low-level IUA simulator with no virtualization. In the low-level IUA with no virtualization, region communication operations

are especially fast, and floating point operations slow. In the following, it is shown for the three steps how the IUA influenced the design of the algorithm:

The first step, the Principal Axis Algorithm 'merges' the support regions before the line is computed, which saves one computation of set of lines. It is a big improvement in time and easy to implement on the low-level IUA. The decrease in quality, however, is big, and so another criteria for region merging was found.

The second step might look like an addition of time, but actually adds only one tenth (for most of the images) to the time of the whole algorithm on the IUA. Mainly highest values have to be extracted out of regions and compared to local information. This can be done with broadcasts and by selecting the minimum/maximum in a region.

For the third step, something less expensive and different from the method in the Principal Axis Algorithm had to be found. For the IUA, the trigonometric functions in the plane fit of the Plane Fit Algorithm are expensive, and therefore a method based more on region communication was developed: Searching the 3 x n points of maximal gradient magnitude takes only one tenth of the time for the whole algorithm (the 3 x n peaks were extracted sequentially, because setting up the structure for a search in parallel within the three subregions would be too expensive). Although there are no trigonometric functions for the least squares computation of the line it takes three fifth of the whole algorithm, because of its many floating point multiplications and divisions. Here is a complete description of the whole algorithm:

1. The gradient orientation and the gradient magnitude are computed.

   The gradients in x and y direction are computed with a 3x3 Prewitt mask, where the arctangent for the gradient orientation is obtained from a lookup table.

   Previous line extraction algorithms have used either a 2x2 mask, as in the Plane Fit Algorithm, or a 3x3 mask, where the 3x3 mask is either a Sobel or a Prewitt

operator. For the 3x3 mask, the author of [8] argues that it does not respond to high-frequency, 1 pixel-wide regions, or to fine details. On the other hand, [9] found that the 2x2 mask produces relatively large errors in the gradient orientation. In our experiments with non-synthetic data, the 2x2 mask sometimes breaks long, straight lines, which are important in outdoor-scene recognition. This can happen because the 3x3 mask can smooth the influence of noise. Among the 3x3 masks, the Sobel operator is slightly slower than the Prewitt operator because of the multiplication with two (or leftshift), and is not significantly better [9]. In our experiments, the Prewitt operator produced slightly better results and was slightly faster on the IUA.

The gradient magnitude is computed by the sum of squares of the gradient in the x-direction and the gradient in the y-direction. This provides slightly better results than the sum of the absolute values, which was used in the Principal Axis Algorithm.

2. The support regions are built using the overlapping orientation bucket scheme. and the two SRRs are merged

After filtering for *Magnitude*, the two SRRs are built in the same manner as in the Principle Axis and Plane Fit algorithm, and they are then merged prior to line fitting like in the Principal Axis algorithm. The merging process itself, however, is different from that of the Principal Axis algorithm. It is based on a voting scheme with each pixel voting for the SRR with the bigger support region covering the pixel. The new region is then a subregion of the voted region. The construction begins by setting a flag associated with each pixel:

Set pixel's flag to one iff its surrounding support region in the first SRR contains more pixels than its surrounding support region in the second SRR, otherwise set pixel's flag to two.

With this rule, the flag assigns a SRR to every pixel. After setting the flag, the final support regions are connected with a connected components algorithm using the following *merging criteria*:

(a) two adjacent pixels have the same flag

(b) two adjacent pixels belong to the same bucket in the SRR assigned by the flag

The application of this rule on a real image is shown in Figure 2. In the figure, the first SRR and the second SRR of a subimage are shown in the top row along with the final SRR of the Principal Axis algorithm and the Subregion algorithm in the bottom row.

The Principal Axis Algorithm creates the final SRR as a union of the first and the second SRR (see section 2.2). The Subregion algorithm, however, builds a subset of either the first or the second SRR. In the figure, pixels are at intersections of region links, and one pixel wide support regions or pixels below the threshold of gradient magnitude are not shown. The numbers to the top right of the pixels indicate the bucket number with a resolution of 0.5 in the range of 0 to 9.5.

The Figure shows that for the diagonal center region, the Subregion Algorithm picks the left SRR, and creates a subregion of it (which is identical to the region in the top left SRR). The Principal Axis Algorithm creates the smallest superset possible of the center region in the top left and right SRR. Due to its merging scheme, it links regions together which could have a very different bucket number. In the figure, the bucket numbers in the final region differs by 3 which equals 108 degrees! At this point, pixels in the region do not represent the same straight line

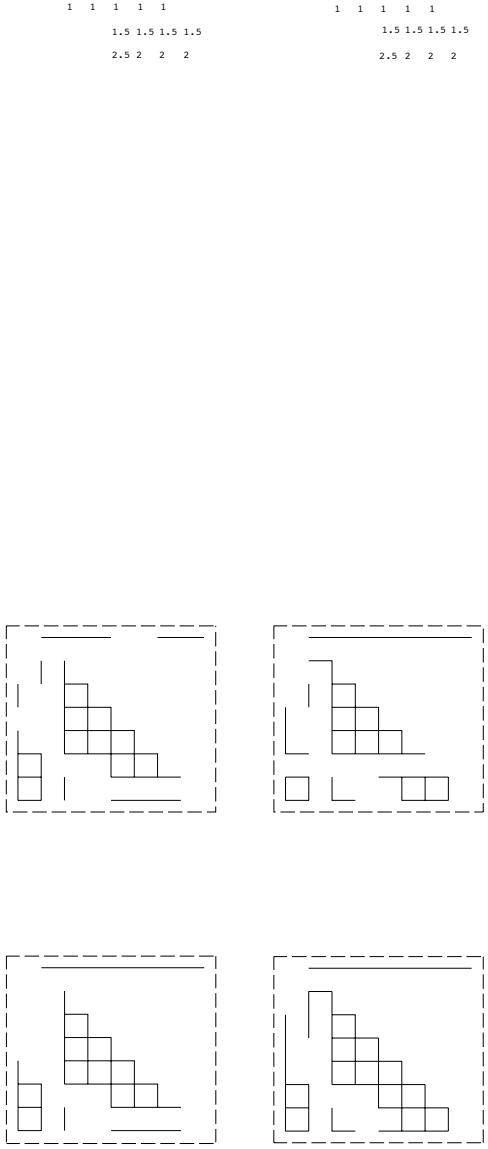

Figure 2: Merging SRR's with the Subregion Algorithm and the Principal Axis Algorithm on a subimage. Pixels are at intersections of region links, and one pixel wide support regions or pixels below the threshold of gradient magnitude are not shown. The Numbers to the top right of the pixels indicate the bucket number, rounded to 0.5 in the range of 0 to 9.5.

any more. This is one of the reasons why lines in the Principal Axis Algorithm can be distorted.

In the Subregion Algorithm the size of the support region is used as a criterion for merging the two SRRs. Using the size of the region is a good approximation of the length of the line, which is not known at this point. Usually, regions are largest when the orientation of the line lies in the middle of the bucket, and they get fragmented when the orientation of the line lies on the edges of the bucket, and some pixels associated with the line fall into another bucket. Therefore, regions are bigger at the optimal bucket partitioning and get fragmented at the suboptimal bucket partitioning.

The merging scheme maintains a spatial separation of regions, which means that each pixel has a unique region to which it is assigned. This has been done mainly for the reason that a well-defined region representation allows one processing step for the region operations; otherwise, two steps would be necessary on the IUA, and additional processing in the sequential implementation. Therefore, little time savings would be gained in comparison to the Plane Fit Algorithm.

In the Plane Fit Algorithm, lines can intersect and maintain the full support region for the better SRR. The disadvantage of having a well-defined region representation is that lines with similar orientation that are collinear (perhaps originating from the same line) or nearly parallel could 'compete' for pixels. In general, such competition happens mostly in the outskirts of a region and was observed more for irregularly shaped regions with no real underlying line. In case two lines compete for pixels around their endpoints, one line wins and gets longer, while the other one gets shorter. Between collinear line segments, however, the gap is often filled with pixels of completely different orientation, so that 'competing for pixels' does not take place.

3. Split the support regions into parts.

In order to obtain an accurate least squares straight line fit to pixels in the support region, the fit should be made using points of highest gradient magnitude more or less evenly distributed along the length of the resulting line. However, in practical applications, variations in the line contrast along the length of the line could easily result in clustering of the highest gradient values. Therefore, in order to ensure a distributed selection of points, the support regions that were constructed in the previous step are split into parts.

Here, we describe the split into *three* parts. With only two parts, the splitting line could lie directly at the location of a cluster and the line could still be distorted. As speed is a concern in the Subregion Algorithm, we chose three for the number of subregions. Also, results in practice show that there is a minimal improvement from three to four subregions.

Splitting the support region is done in the direction of the gradient of one of its arbitrarily chosen pixels. The choice of the pixel does not matter because every pixel's orientation differs by at most one Bucket Size from the real orientation of the line. As the default value for the number of buckets is 10, this would be a maximal orientation error of 36 degrees.

Usually, the bucket ID corresponds to the orientation of the shape of a support region. If a straight line is underlying the support region, the support region has an oval shape, which is split along its minor axis. We convert the gradient orientation of the arbitrarily chosen pixel into the slope m. Through every point (x_i, y_i) in the support region, we draw a line with slope m and obtain:

$$t_i = y_i - x_i \cdot m,$$

where t_i represents a value on the y-axis. Naturally, some points (x_i, y_i) have the same t_i.

tmin

The line representation of slope and intercept was preferred over the radius and angle representation because trigonometric operations are expensive on the IUA, and also on sequential machines they are costly.

After assigning a t_i to every point (x_i, y_i), we look for the t_{min} and t_{max} which are the most distant t_i from each other on the y-axis (x-axis for vertical slope). Figure 3 shows the construction of the split.

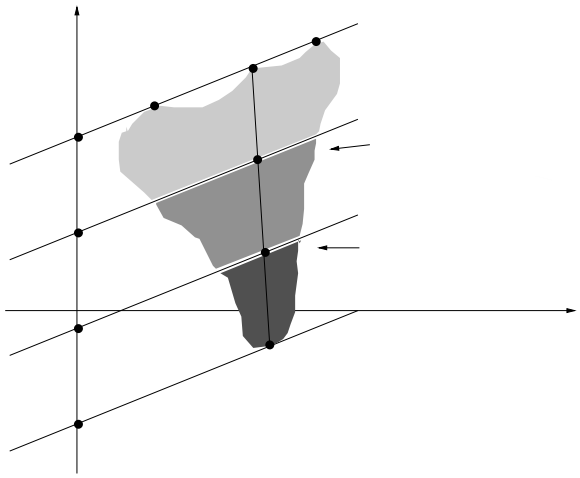

Figure 3: Split of support region (shaded region) into three parts. The three new subregions are indicated by the light gray, medium gray and dark shaded regions

The lines with slope m through t_{min} and t_{max} are only 'touching' the region and they are the most extreme lines which still intersect with region pixels in the selected orientation.

Choose two arbitrary points inside the support regions which lie on the lines through t_{min} and t_{max} and call them p_{min} and p_{max}, as shown in Figure 3. Next, the line segment from p_{min} to p_{max} is divided into three parts of equal length, with points p_1 and p_2 as splitting points. The final splitting lines l_1 and l_2 pass through p_1 and p_2 with slope m, so that they divide the region into three parts along its length.

This formulation works for values of m that are not too large, otherwise numerical instability is introduced. We found that $m > 150$ and $m < -150$ are satisfactory boundaries for considering m to be vertical, in which case t_{min} and t_{max} are the most distant *vertical* lines touching the support region.

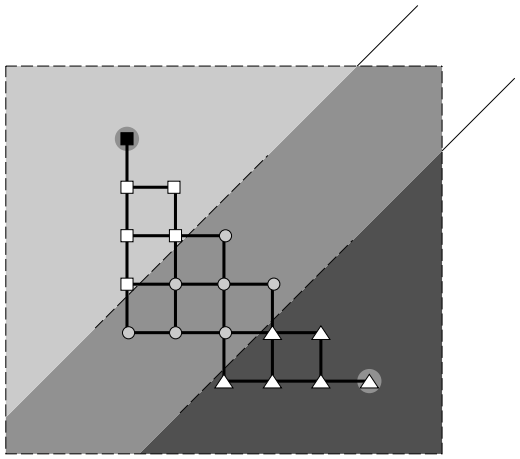

Figure 4: Split of support region into three parts. The black pixel is the arbitrarily chose pixel, which determines the slope of line l1 and l2. The square pixels lie in the light gray area, the circles in the medium gray area, and the triangles in the dark shaded regions.

Figure 4 shows the split for the final support region of the Subregion Algorithm in Figure 2 from the previous step. The support region is evenly split along its long axis, although the arbitrary pixel giving the orientation (the black pixel) is a border pixel of the support region (and, as we will see in Figure 6, also distant to the output line).

4. Select n points of highest gradient magnitude in the subregions of the support regions

n points of highest gradient magnitude are selected in each subregion, where the number of selected points is equal in each subregion. Given all the pixels in the support region, in this step, clearly the gradient magnitude is the final evidence for line pixels. Not all line pixels, however, have to be selected in order to retrieve a good approximation of the line.

The setting of the parameter n is a tradeoff between speed and accuracy. We have selected three points in every subregion for our implementation, which should prevent outliers to be responsible for a whole subregion. Large numbers of n, however, are not always favorable. In case almost all points in the subregions get selected as points of highest gradient magnitude, the least squares fit in step 5 would rather be an approximation of the orientation of the support region than of the orientation of the line.

The same support region as in the previous figure is shown in Figure 5 without shading, where the selected points are black circles, squares and triangles, depending on their subregion. Also, the ranking in gradient magnitude is shown, with one meaning the highest score. Note that the subregion with triangle points has a lower ranking for all three points than the other regions. This is an example of fluctuation of gradient magnitude along the line, and therefore shows the necessity of subregions.

5. Retrieve the position and orientation of the line with a least-squares fit method.

In this step, the line parameters are computed from the points selected in the previous step. The line is represented in slope m and intercept t. In the ideal line match, for every pixel i on the line, the following equation is true:

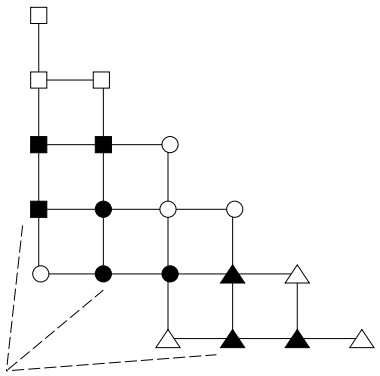

Figure 5: 9 selected points of highest gradient magnitude with triangles, circles and squares representing points in the subregions. The filled black points represent the points of highest gradient magnitude, where their numbers are equivalent to a ranking of their gradient magnitude values.

$$y_i = m \cdot x_i + t$$

The error function to be minimized consists of the differences between the y coordinates of the pixels and the virtual y coordinate of the line at the same x coordinates. This error is squared for every pixel, and the sum of the squares is the error function E. In order to prevent numerical instability for lines with vertical slopes, the x and y-coordinates are exchanged for regions in which the gradient orientation of an arbitrary pixel produces a slope above one.

The least squares method was chosen because it has a linear solution, and is therefore easy to implement and executes quickly on the IUA, where trigonometric functions and iterative arithmetic floating point functions are costly. With the least squares fit, trigonometric functions are avoided, making it faster. Furthermore, the least squares fit produces optimal results in the absence of outliers. If outliers are present, the selection of several points in each subregion smoothes the effect to some extent.

The 'distance' from each pixel in the support region to the line is used to determine if pixels are 'on the line' or not. If $|m| \leq 1$, 'distance' means the vertical distance of each pixel to the line; if $|m| > 1$, 'distance' means the horizontal distance of each pixel to the line. The pixels are labeled 'on the line' if 'distance' is ≤ 0.5.

6. Compute the endpoints of the line.

In this step, the integer coordinates of the endpoints of the lines are computed. The endpoints are the pixels in the region 'on the line' (see previous subsection) which are furthest away from each other. This is implemented by selecting the highest and lowest value obtained by concatenating the short values row and column index of the pixels 'on the line' into an integer value. In order to select the maxima and minima of this integer value for endpoint extraction, one of the row or column values has to be subtracted from the image row size or column size first, depending on the slope of the line. If the slope of the line is positive, both row and column

values have maxima or minima at the endpoints. (assuming an x-y coordinate system for the columns and rows). If the slope is negative, one of the row or column values has to be subtracted from the image row size or column size first.

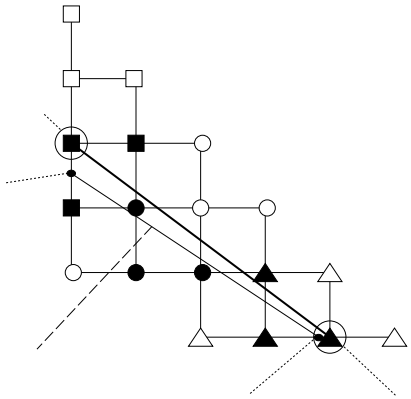

Figure 6: The final line, showing the line from the least squares fit with integer and floating point endpoints.

Integer endpoints are an approximation to floating point endpoints in small images. Figure 6 shows the final line of the example from the last steps, and also the deviation from the floating point endpoints, computed with slope and intercept from the least squares method. The figure shows that the orientation error in this example is about 5 degrees. This is the maximal orientation error, because the real endpoint was directly between two integer points, and the line has only length 5. A line with length 15, for example, would only exhibit a maximal deviation of 2 degrees.

Note in Figure 6 also the slight deviation of the orientation of the real line from the orientation of the support region. This was the reason for making the recovery of the line independent from the shape of the support region, in contrast to the Principal Axis Algorithm, where the line is dependent on the shape of the support region.

7. Apply the filtering values.

Lines may now be filtered on the basis of their features, such as length or contrast. Note that line contrast is computed as the average of the gradient magnitude of the line points. Pixels that by themselves do not exceed a certain gradient magnitude are already filtered out in the first step.

Experimental results which compare the Subregion, Principal Axis and Plane Fit Algorithm are reported in the next section.

4 Experiments

In order to verify the performance/quality ratio for the Subregion Algorithm, both line output quality and speed were measured. The same experiments were also done for the Plane Fit Algorithm and the Principal Axis Algorithm, for doing comparisons with other established algorithms. Hereby, the expectation is that the comparison to the Plane Fit Algorithm yields superior quality and worse speed for the Plane Fit Algorithm, and the comparison to the Principal Axis Algorithm yields worse quality and superior speed.

Methods of estimating the output quality of lines include comparing directly the line output of several algorithms with the measure of the eye, or comparing specific measures of a performance evaluation algorithm. Assessing the line quality with the measure of the eye determines how the lines reflect lines in the real image, but might not give accurate information about the use of the lines by other algorithms. A performance evaluation algorithm provides some performance parameters, but might provide misleading information on the global quality of the lines obtained. In this paper, line outputs from all three algorithms are compared on three image sets, and then compared in terms of the performance evaluation algorithm described in [11]. For speed comparisons, the three algorithms were run on the SPARC-10 on the three image sets and the Subregion Algorithm was run on the IUA Simulator for two image sets. The performance/quality ratio is then computed from the obtained quality and speed measures.

4.1 Quality Measurements

4.1.1 The Performance Evaluation Algorithm

The performance evaluation algorithm introduced in [11] is based on pixel-wise comparisons of groundtruth segments and recovered segments and works on integer endpoints.

First, an association list between recovered line segments and ground truth segments is created by searching a one-pixel wide stripe which is oriented orthogonal to the groundtruth line for every groundtruth pixel. The length of the stripe is limited on both sides of the groundtruth line by the truncated half of the 'interval width', which is

currently set to 6 (then the total length of the stripe is 7, also counting the groundtruth pixel). For every groundtruth pixel, the closest occurrence of a recovered line pixel is recorded. The association list indicates which recovered segments cover how much (determined by a 'vote count') of one groundtruth segment. Start and endpoints of the recovered segments are then projected to their groundtruth segments in the association list. Final assignments to the groundtruth segments are made first for the recovered segments with the highest vote count. If a projected segment which occupies less groundtruth pixels overlaps with a projected segment which occupies more groundtruth pixels (and whose final assignment is done first), the former is filtered out if the overlap is bigger or equal to the truncated half an 'operand width'. Currently, the 'operand width' is set to 5 (which means that only overlaps of up to size two are allowed). Final assignments, however, only happen if the recovered segment has less than double the pixels of the vote count. In the end, the permanently assigned segments serve as input for several statistics.

The statistics computed with this performance evaluation algorithm are the mean and variance of several performance parameters. The following is a description of the performance parameters that are used later in this paper. A '+' in the itemized description means that the performance parameter has been added for this evaluation. The (m) means that the computation of the performance measure has been modified. In the descriptions, 'assigned segment' signifies a recovered segment which has been associated with a groundtruth segment.

- **random length:** length of recovered segments which are filtered out in matching process of the algorithm
- **segment length:** length of assigned segment
- **max. segment length:** length of longest assigned segment (no record if no assigned segments)
- **+excess length:** total length for which assigned segments projected onto their groundtruth segment exceeded the endpoints of the groundtruth segment (no record

if no assigned segments)

- **gap length:** length of gaps between adjacent assigned segments projected onto their groundtruth segment; parameter is set to one if adjacent segments overlap (no record if no assigned segments or only one assigned segments)
- **(m)fragmentation:** number of recovered lines per groundtruth segment (set to zero if no assigned segments)
- **orientation error:** difference in orientation between assigned segment and groundtruth segment
- **+min. orientation error:** smallest orientation error in the assigned segments, for every groundtruth segment (no record if no assigned segments)
- **+midpoint distance to groundtruth:** perpendicular distance of the midpoint of the assigned segment to its groundtruth segment
- **+groundtruth coverage:** percentage of coverage of groundtruth segment (no record if no assigned segments)

The performance parameter fragmentation was modified to take into account the groundtruth segment with no or one assigned segment. The original version recorded the number of breaks in each groundtruth segment, but no record was made for a groundtruth segment which had zero or only one assigned segment.

4.1.2 Comparison of Line Output

Comparisons were done using three sets of images; the image sets were selected assuring variety with respect to several criteria, namely image size, contrast, brightness and number of lines present in image. Groundtruth was selected by hand for the first two image sets. Hereby, in each image lines were selected which satisfy at least one of the following criteria (a set of lines should satisfy many of them): lines essential to the image content,

lines with low contrast across the line, lines with high contrast along the line, lines with slow gradient change, lines which are long, lines which have different angles and lines which are in general not easily detectable.

The first image set was selected to test if the algorithms works well on a variety of images with completely different settings, such as contrast and brightness. The set consisted out of 10 mixed indoor/outdoor images which are 256 x 256 large and have each 15 to 35 groundtruth lines associated with them. Only lines greater or equal to length 16 were selected. With the second set, a 256 x 256 motion sequence, it was tested if lines can be tracked consistently over several frames. The image set contains 7 images, and in each image 27 to 35 groundtruth lines were selected by hand from corresponding cones and other corresponding objects. The third image set tested how well the algorithms extract lines in aerial images. It is an aerial motion sequence, taken from an artificially built outdoor model scenery (the RADIUS modelboard) with camera parameters different from the other two sequences, and the images slightly larger than 1024 x 1024 pixels. The groundtruth in this image set was taken from the University of Washington [11] with 958 to 2266 groundtruth lines in each of the eight images.

Figures 7 to 13 show the real image, the groundtruth lines, and the line outputs of the three algorithms. The following paragraphs explain the line output for the three algorithms on the three image sets.

The First Image Set

Figure 7 shows three indoor and one aerial image of the first image set. The first image (numbering from left to right), taken from a motion sequence, displays lines with high contrast across the line, low contrast along the line and relatively fast change in gradient magnitude; basically it contains lines ideal for line extraction algorithms. The second image as well as the third image contain different lightening conditions. In the third image, some lines are high contrast, and some dark lines and not clearly discernible with the eye. The fourth image contains many lines with low contrast and lines which are difficult to classify as lines.

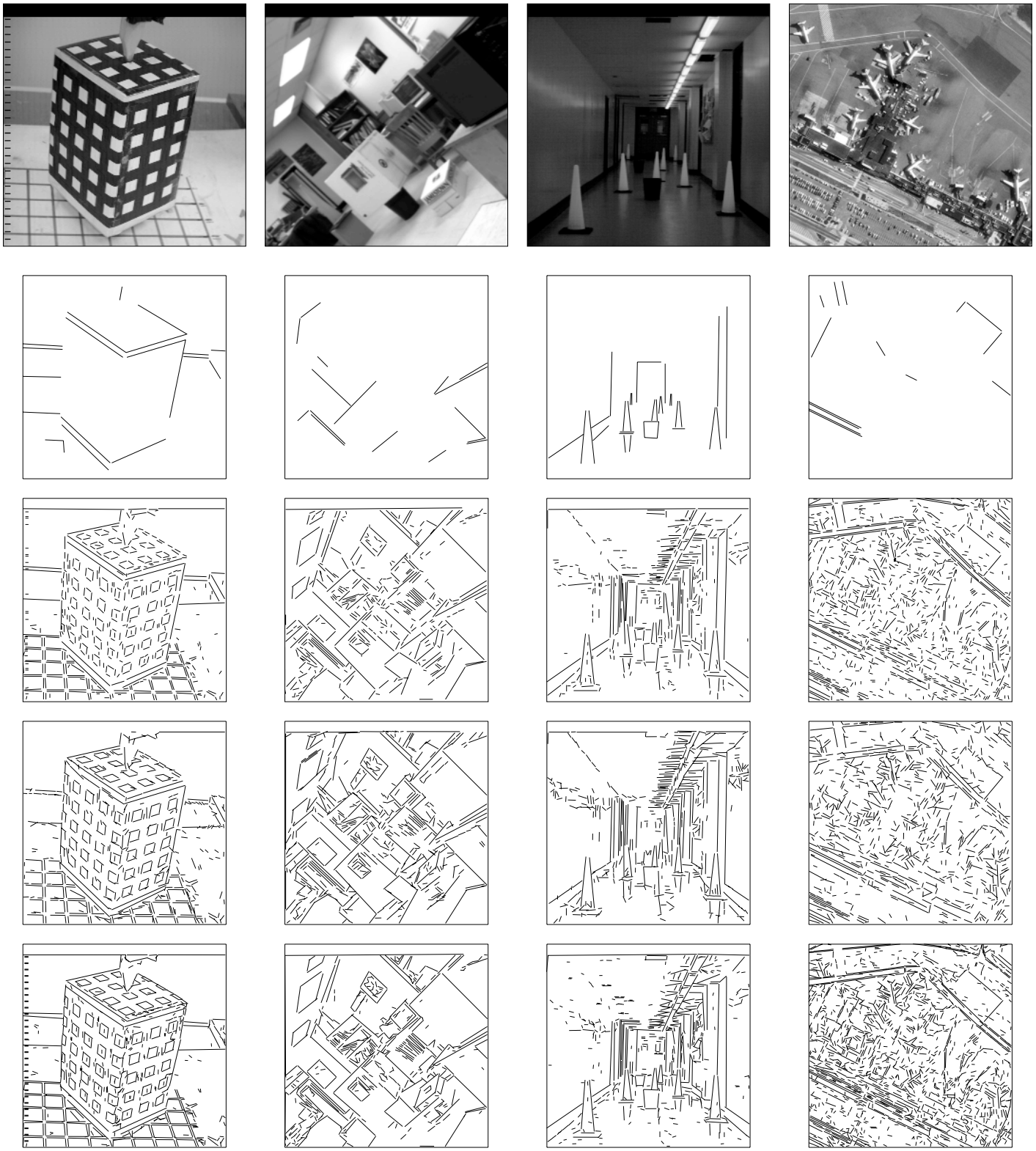

Figure 7: First four images of first image set. The top row shows the real image and the second row the groundtruth. The third, fourth and fifth rows show the output for the Subregion, the Principal Axis and the Plane Fit Algorithm respectively.

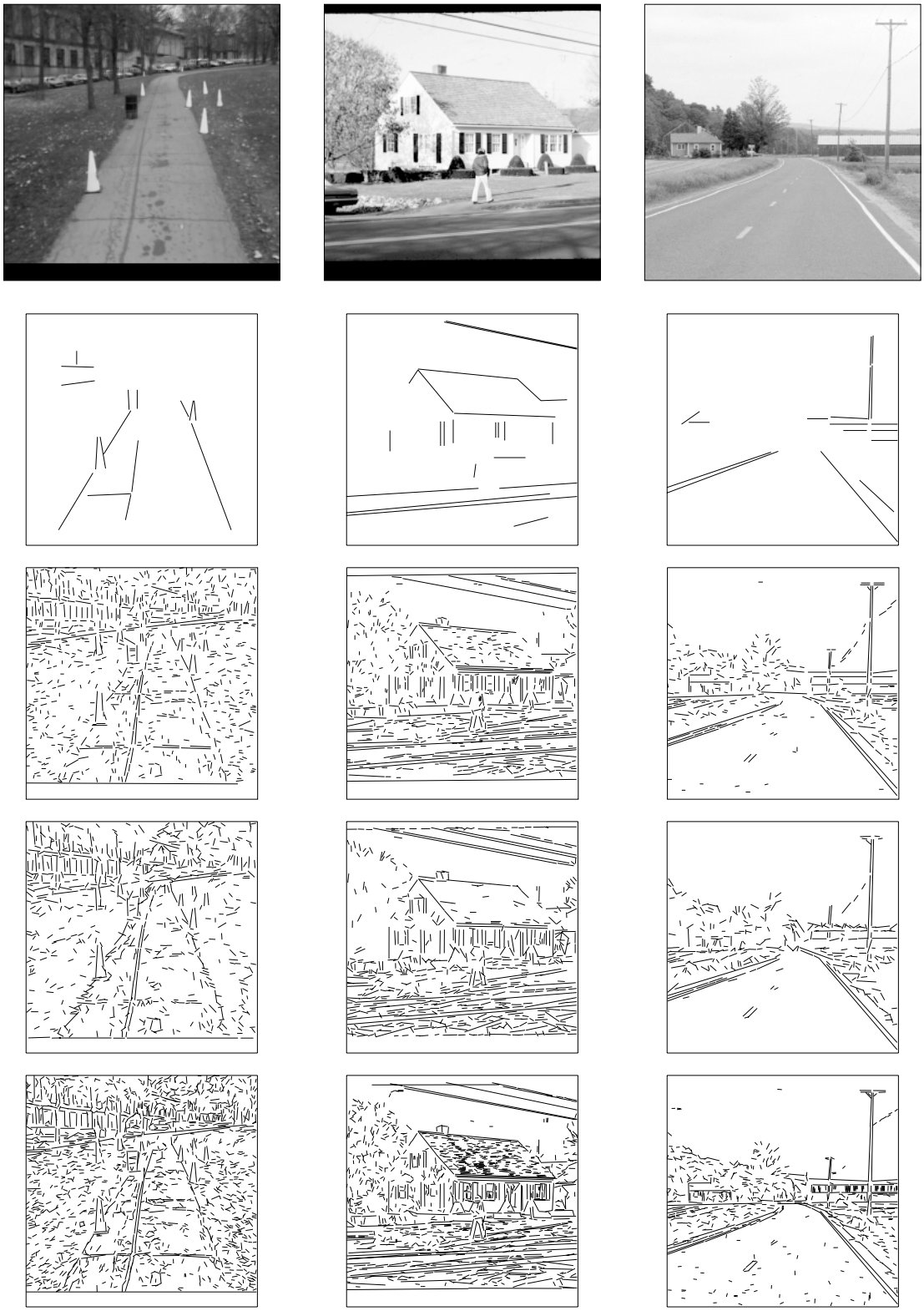

Figure 8: Fifth to seventh image of the first image set. The top row shows the real image and the second row the groundtruth. The third, fourth and fifth rows show the output for the Subregion, the Principal Axis and the Plane Fit Algorithm respectively.

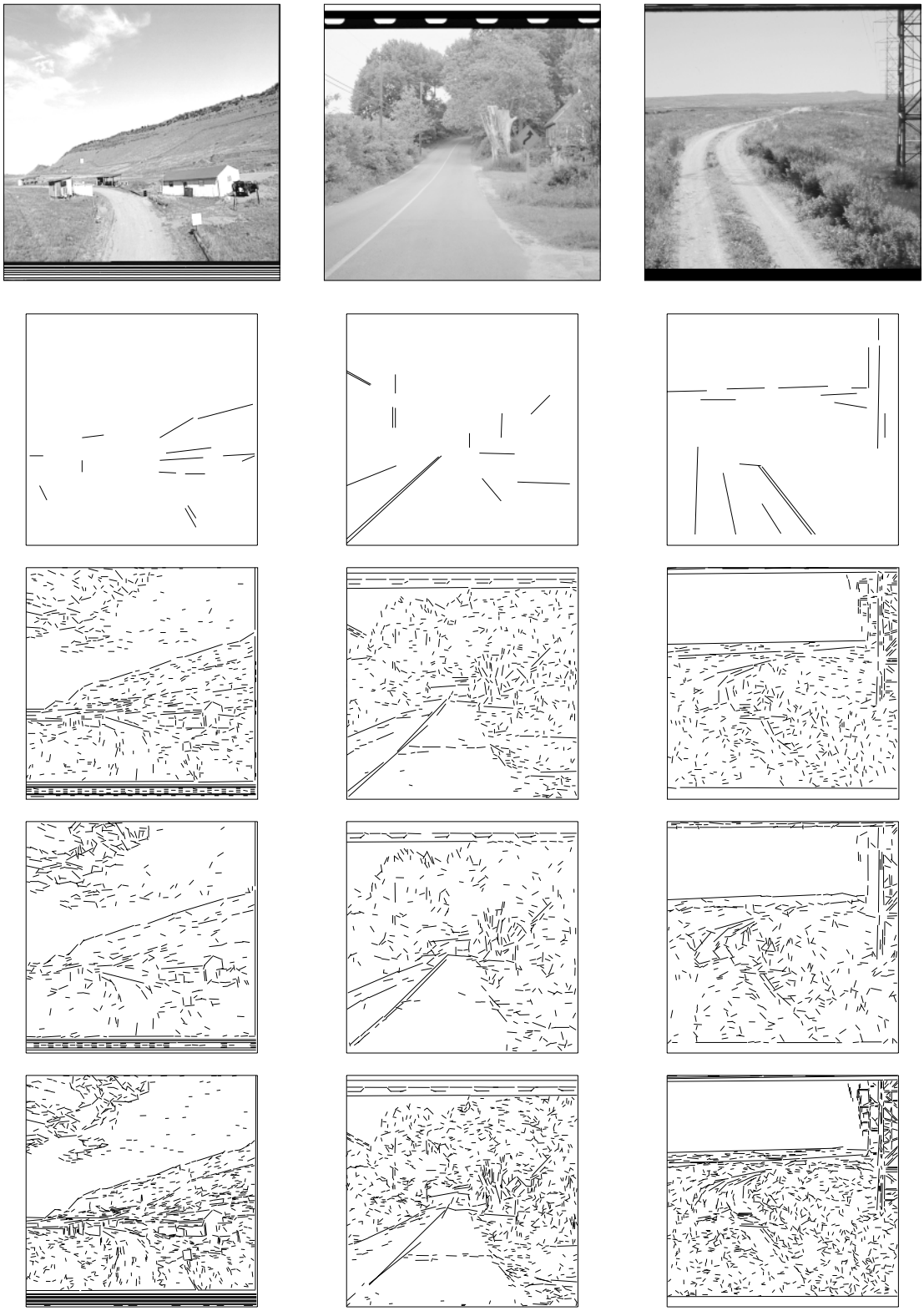

Figure 9: Eighth to Tenth image of the first image set. The top row shows the real image and the second row the groundtruth. The third, fourth and fifth rows show the output for the Subregion, the Principal Axis and the Plane Fit Algorithm respectively.

Throughout the four images, the quality of all algorithms does not differ greatly. Closer examination of the line output, however, reveals that the Plane Fit Algorithm detects detail very well - even the scale on the left border of the first image is detected. The Subregion Algorithm detects some low contrast and blurred lines well (the right ceiling border in the second image), but does not always extend lines until corners (the posters on the wall of the second image), and the Principal Axis Algorithm sometimes fragments lines into smaller lines with high orientation error (see horizontal line to the right of the box in image one, border lines of the floor in image two or dark rectangle in top right of image four). These fragmented lines are not always low-contrast lines; the line in the first image one has high contrast across the line.

Figure 8 shows three outdoor scenes which have some straight lines with high contrast and fast gradient change, but also lines which have no clear-cut boundary and still are discernible as lines for the observer. Examples for the first kind come mostly from man-made objects, whereas examples for the latter kind are lines in nature like the horizon line and border lines between occluded man-made objects and lines in nature, like the border between the road and grass. In the images, most lines of the first kind are extracted well by the three algorithms, for example, the white stripe on the right border of the street in the third image (numbering from left). The roof of the house in the right side of the same image, however, is not extracted by the Principal Axis Algorithm and the window is only recovered by the Plane Fit Algorithm which is again best at recovering details in small objects. Concerning border lines of man-made objects to nature, there are examples where every one of the three algorithms works best. The Principal Axis Algorithm, for example, recovers the three wires in the top of the second image best. It misses, however, essential parts of the house in the left side of image three and does not recover short lines to represent the tree. Also, it fragments the boundary of grass to pavement in images one and three. The Subregion Algorithm recovers the hidden roof in left side of the third image best (see groundtruth) and has mostly lines with the correct orientation in the surrounding of the groundtruth line. The Plane Fit Algorithm recovers the grass to pavement border line in image three best, but fragments the bottom right border in the first image.

In Figure 9, some outdoor images with many straight lines from nature and bound-

aries to occluded man-made objects (like roads) are shown. The Subregion Algorithm works best in detecting the two horizontal blurred lines in the bottom right of the second image. The Plane Fit Algorithm recovers the roof of the house in the first image well, and shows superiority in recovering detail with the cottage in the first image, the traffic sign in the second image and the metal stand in the third image. The Principal Axis Algorithm recovers the line on the left side of the road in the second image best, but is worst on recovering the roof of the house and structure of the cottage in the first image. It also does not recover any detail of the metal stand in the third image. The Principal Axis Algorithm, however, is the only algorithm that does not merge the horizon line in image three.

In summary, the following observations can be recorded from the first image set, along with the explanation for their cause:

- The Plane Fit Algorithm is best at recovering detail. The reason is that it operates with a 2 x 2 mask, which was not chosen for the Subregion Algorithm because of bad experimental results.

- The Principal Axis Algorithm has high fragmentation, also with high contrast, fast changing gradient lines. This can be due to the implementation of the arctangent used for computing the gradient. In the table lookup, pixels which have slightly different approximate horizontal orientation fall into a different bucket with both region representations. This can also be the case with diagonal orientation.

- The Principal Axis Algorithm has a high orientation error with some lines. This can be due to the merging scheme of the two SRRs into the final SRR, as well as the computation of the line as a Principal Axis. The continuous linking of pixels with other pixels which have slightly different orientation can change the shape of the support region. Computation of the final line as a principal axis, however, uses the shape of the support region.
  The final SRR in the Principal Axis Algorithm can also be the cause for a lack of lines, for example in the second image of Figure 8.

- The Plane Fit Algorithm and the Subregion Algorithm sometimes merge groundtruth lines or do not recover lines because they are merged with other groundtruth lines. Examples of this phenomenon are the wires in the second image of Figure 8 or the horizon line in the third image of Figure 9. The problem could be adjusted by increasing the bucket size, at the price of increased fragmentation.

The Second Image Set

The second set of images tests how well line correspondences are recovered out of a motion sequence. The groundtruth lines are the cones, the two wastebaskets and the door frame. Figure 10 shows image one to three (numbering from the left), and Figure 11 shows image four to seven, where a noisy image has been taken out between images four and five.

In the line output, the cones are recovered unfragmented by all algorithms. Merging the cones with the vertical wall structures, however, is a problem for all the algorithms. The same cones are merged, except for the sixth image, where the Plane Fit Algorithm does not merge the left cone. The difference between the Subregion Algorithm and the other algorithms is that instead of recovering a line with slight slope for the cone as well as the wall, it only approximates the line on the wall. The reason for this is that the points of highest gradient magnitude lie at the top of the cone for the first and second subregion, and further up on the wall for the third subregion, so the resulting line approximates more the wall line and does not lie in the support region at the level of the cone.

The bottom plates of the cones are tracked consistently by the Plane Fit and the Subregion Algorithm, but they are only recovered in fragments by the Principal Axis Algorithm, being the fragmentations at different locations within the plate lines. The same is true for the Principal Axis Algorithm with the smaller wastebasket. The door frame and the bigger wastebasket are also fragmented at different locations in the image sequence, but there are consistent pairs of fragmentation patterns. In general, although it does not recover the merged lines to the full length of the cone, the Subregion Algorithm

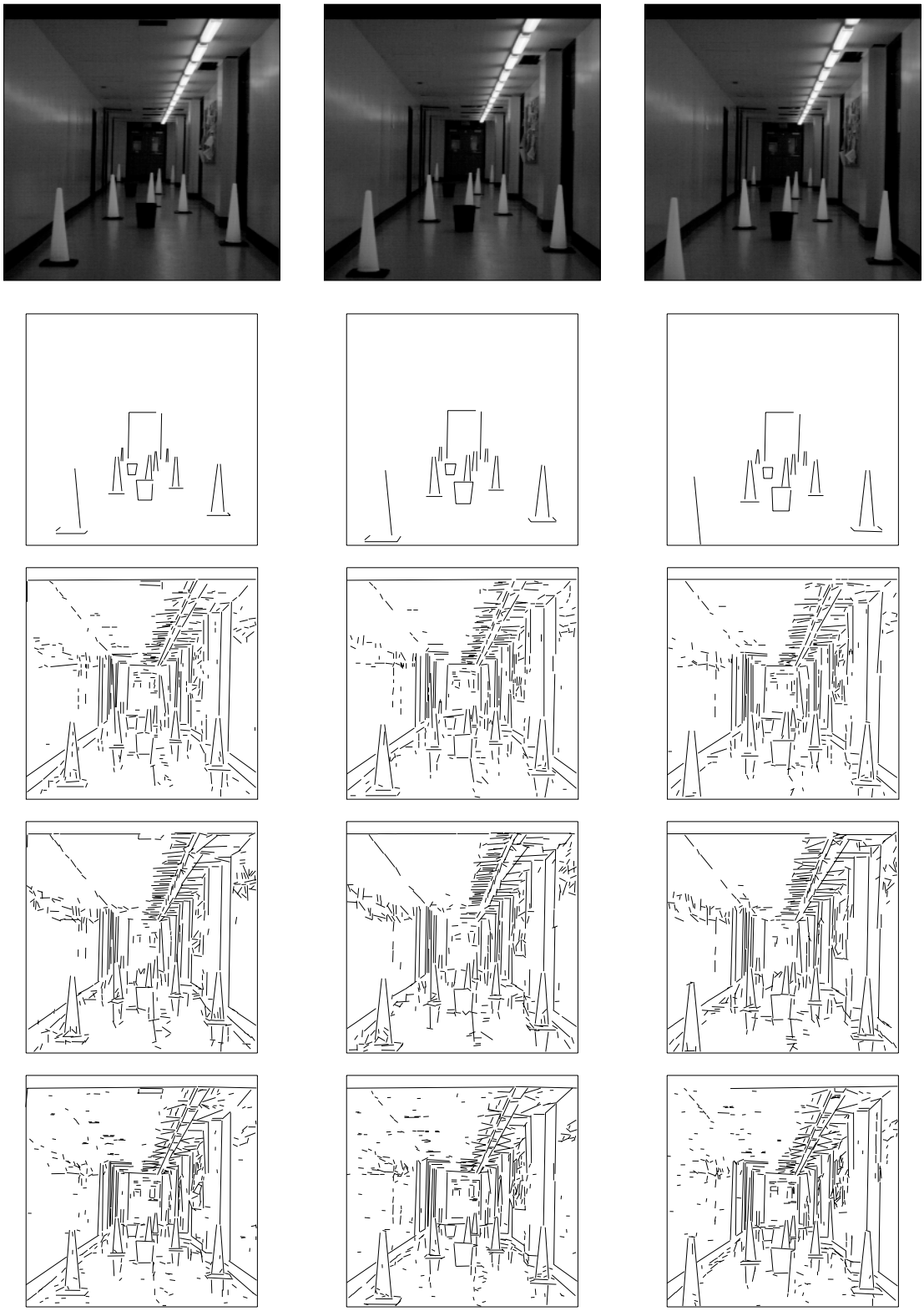

Figure 10: First three images of the second image set. The top row shows the real image and the second row the groundtruth. The third, fourth and fifth rows show the output for the Subregion, the Principal Axis and the Plane Fit Algorithm respectively.

Figure 11: Last four images of the second image set. The top row shows the real image and the second row the groundtruth. The third, fourth and fifth rows show the output for the Subregion, the Principal Axis and the Plane Fit Algorithm respectively.

tracks lines more consistently than the Principal Axis Algorithm.

All the itemized observations recorded for the first image set were also made in the second image set.

The Third Image Set

The last image set tests how the algorithms work on aerial images, as shown in Figure 12 and Figure 13, which are the second and fourth of the eight images in the aerial image sequence. The line output shows that the Principal Axis Algorithm produces highly fragmented lines, but that the fragmentation is mostly on the horizontal lines. It also produces lines where no evidence of a line is visible with the eye (see the two diagonal lines in the top left of Figure 13), and sometimes has high orientation errors (see the line distorted by 90 degrees in box with stripes in the bottom right corner of Figure 12 or the two large squares in the middle bottom of Figure 13). The Subregion Algorithm and Plane Fit Algorithm do not produce the same fragmentation or orientation errors, and their distinction is that the Plane Fit Algorithm recovers more detail. With some long lines, however, the Subregion Algorithm produces better orientation estimates, for example the square in the bottom right of Figure 13.

In the third image set, all the observations from the first and second image set can be repeated except for merging support regions with different underlying lines. The list of observations can be extended by one for the third image set:

- The Principal Axis Algorithm sometimes recovers lines with no line association in the image. The reason is that originally small regions could expand in the region merging step due to gradual gradient change, which puts pixels with completely different gradients in one region.

4.1.3 Performance Evaluation Results

Table 1 compares the performance parameters for the three algorithms on the three image sets. Hereby, the evaluation results are for integer endpoints, and the endpoints of

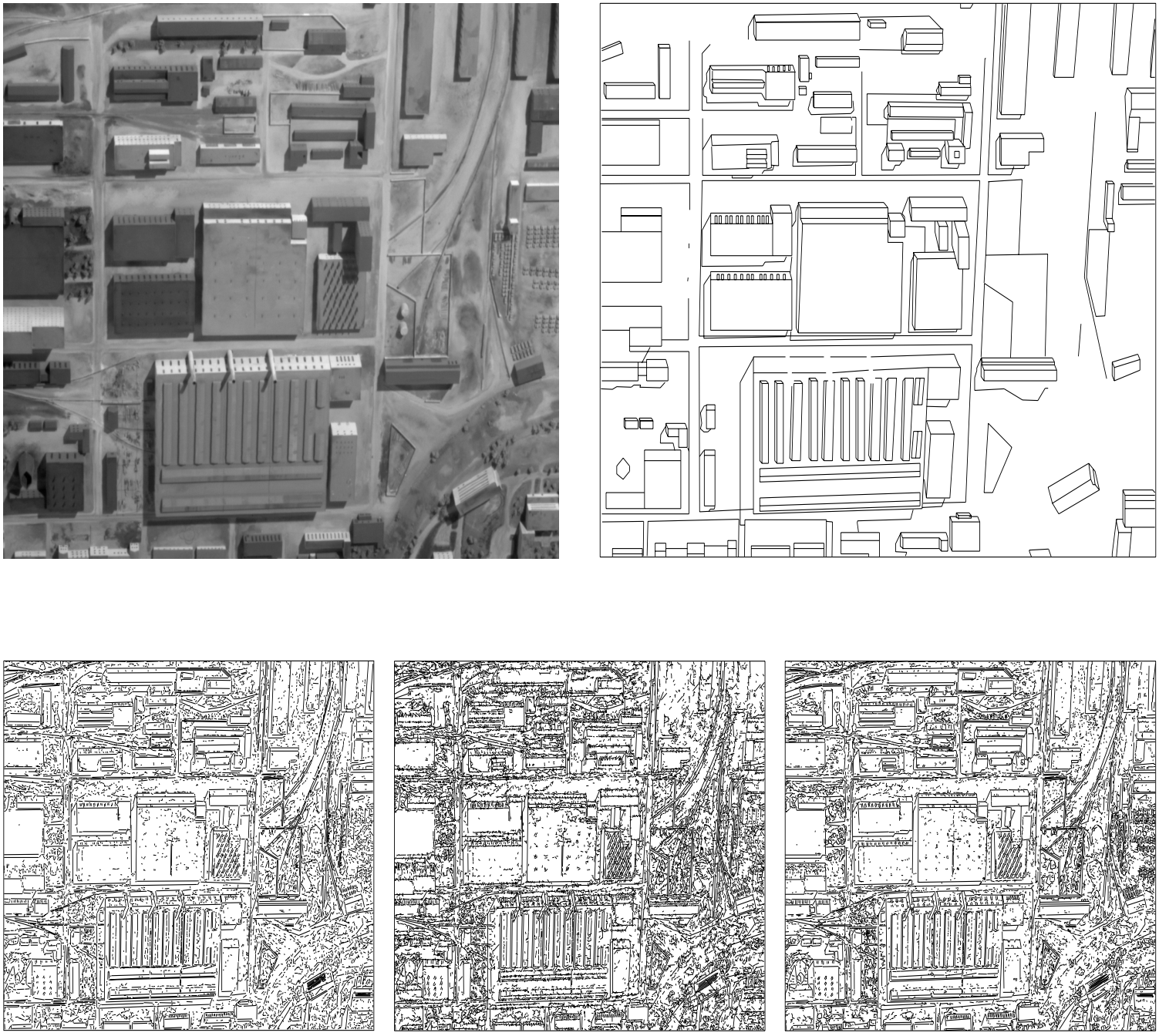

Figure 12: Second image of the third image set. The top row displays the real image and the groundtruth image, and the bottom row the Subregion Algorithm (left), the Principal Axis Algorithm (middle) and the Plane Fit Algorithm (right).