Detecting Textured Objects Using Convex Hull

Kefei Lu[a,c] and Theo Pavlidis[b,c]
a. Credit Suisse Group
b. Stony Brook University
c. Part of the work discussed in this paper was performed when both authors were affiliated with Symbol Technologies

1. Introduction

Applications of computer vision technologies have benefited a wide range of industries over past decades. While some of the tasks are considered relatively easy from an academic point of view, industrial requirements on adaptability and flexibility may impose excessive difficulties on implementations. Hence, new algorithms need to be designed accordingly.

In this paper, we describe a convex hull based methodology that is helpful for detecting convex solid objects whose surfaces are not of uniform color or texture but may contain graphics and be of different colors (*internal clutter*). In addition, there may be other objects in the scene besides the ones of interest (*external clutter*). This methodology was developed as part of an industrial machine vision system, estimating the dimensions of rectangular or cylindrical boxes by a relatively inexpensive device. The application is described in detail in a recently issued US Patent [1] that also contains the details of the implementation of the method. The image analysis part is used to locate the object(s) and estimate the relative dimensions. The real dimensions are then calculated by estimating the distance of the object to the camera with an auxiliary laser beam whose absolute position is known and its parallax provides distance information.

Several systems have been developed to automatically recover known-shape objects for industrial usage, such as the one developed by Katsoulas [2] to recover piled box-like objects. However [2] uses range data to avoid the texture and illumination problems. With the growing popularity and decreasing cost of digital cameras, using intensity data is more desirable. Reconstruction of 3D objects with known generic properties from a single intensity image has been studied in recent years, for example corridor scene in [3] and polyhedron in [4]. Yet not much has been done on how to reliably locate the vertices of the object, which are critical to the reconstruction process. Various techniques have been proposed in the 3D object recognition literature. Points and line segments are still the most commonly used features. They are extracted and hypothesis of models and their poses are formed using methods such as geometric hashing [5] which uses those features as indices into pre-stored database, or the alignment method [6] which uses the features as “anchors” for initial estimation. Costa and Shapiro [7] used more sophisticated features, including pair of parallel lines, junctions, triplets, clusters of coaxial arcs and ellipses. However the number of such local features (and their combinations) grows rapidly with the complexity of the scene, especially where heavy graphics exist, and this

makes approaches based on such low level features impractical for a real-time industrial application.

The main contribution of this paper is the presentation of a case where prior knowledge about the objects of interests is used to design the low-level vision part of the detection algorithm. Because the image is captured by a hand held device that is aimed at the object(s) of interest, these objects are expected to occupy most of the image area. In addition, we expect only one object of interest and rarely two or three. The purpose of the device is to measure the dimensions of a single object and the case of more than one object is likely to be due to careless use. We emphasize the usage of one high-level feature of the object, its 2D contour. Because the target objects are of rectangular or cylindrical shapes, their outlines are expected to contain long straight-line segments and their 2D projections are expected to be convex polygons.

Taking advantage of the prior knowledge allow us to deal with the difficulty that the contrast between an object and the background might be lower than the contrast of the internal clutter. Rather than use an edge detector we apply directly a line detector (described in Section 2.1) that tends to err on the side of missing edges. Because of the prior knowledge about the object we are able to fit such gaps easily as described in Section 2.2. In contrast, traditional approaches to object detection, for example [7], emphasizes a sequence of operations: edge detection followed by line or smooth curve formation and then aggregation of the lines/curves to form outlines of objects.

The methodology we use for finding the object outlines has its roots in the earlier work on drawing beautification by one of the authors [8]. There clustering was applied to find geometrical objects that satisfied certain constraints *approximately* and then the objects were modified to satisfy the constraints *exactly*. In this work, we also detect approximate relationships (e.g. nearly parallel lines) and we use them to infer the shape of objects where the relationships are satisfied exactly. We return to this point in Section 2.3.

We present experimental results in Section 3. Conclusions and future work are discussed in Section 4.

2. The Convex Hull Approach

Using the prior knowledge of the geometric shape of the object to be detected, we design the convex hull based approach, which has three main stages: direct line detection, proximity clustering and editing of convex hull. Figure 1 shows the outputs from each of these three stages.

2.1 Direct Line Detection

With varying illuminations and backgrounds, the contrast of edge points in a gray-scale image depends on a local area. Shakunaga [3] adopted adaptive thresholding to address

the problem invoked by natural lighting conditions, which determines threshold in each local window. To deal with the varying illumination in our case, we have developed the direct line detection algorithm.

The image is divided into small square blocks (8×8 is chosen in our implementation) and for each block we test the hypothesis that there are two regions of different intensities separated by a rectilinear boundary. Within each block, the gradient is computed at each pixel (sub-sampling can be used to further reduce computation) and a pixel P is identified where the image gradient achieves its maximum value. If the value of the maximum gradient is below a preset threshold, we assume that there is no line in the block. Otherwise we divide the block into two regions by a line orthogonal to the direction of the gradient and passing through P. For each region we compute the average intensity and the root mean square error. If the errors of both regions are small, which indicates each region is relatively uniform, and the average intensities of both regions are significantly different, we accept the line segment as significant. Otherwise, we subdivide the block into four smaller blocks and repeat the process until either we identify a line segment, or the block becomes too small (4x4 in our implementation). Details of the algorithm are described below. Figure 2 gives a graphic illustration of the process.

Direct Line Detection Algorithm

```
DirectLine(Image I, BlockSize s)
{
 for each block B of size s in I{
   compute gradient value g_p and direction d_p at each pixel P;
   subdivide(B);
 }
}

subdivide(Block B)
{
 identify the pixel P_m with maximum gradient value g_Pm;
 if ( g_Pm < g_min ) return;
 derive line L which goes through P_m and has the direction vector orthogonal to d_Pm;
 divide B into two regions R_1, R_2 using L;
 compute average intensity M_i and E_i,
```

$$M_i = \frac{1}{n_i}\sum_{j=1}^{n_i} v_j, \quad E_i = \frac{1}{n_i}\sum_{j=1}^{n_i}(v_j - M_i)^2,$$

$i = 1..2$, n_i is the number of pixels in region R_i;

if ($E_i < E_t$ and $|M_1 - M_2| > \delta$)

```
   record line L;
 else{
   subdivide block B into 4 blocks B_i, i = 1…4;
   for ( i = 1 to 4 ) subdivide(B_i);
 }
}
```

Parameters used in the algorithm, the threshold of minimum gradient value g_{min}, and the threshold of regional intensity difference δ, are determined dynamically. First the gradient of each pixel in the image (or a sub sampled version) is computed and estimates of the maximum and average gradient of the image are found. These are used for the initial setting of the parameters favoring small thresholds both for determining that the gradient at a pixel is significant (g_{min}) and for determining that the difference in intensities is significant (δ). However, if the number of lines segments found is too large, both thresholds are increased and the process is repeated. At each iteration the threshold is increased by an amount proportional to the number of iterations. After the nth iteration the threshold is increased by $4n$. The maximum number of lines is set to 1000 in our implementation and at most 5 iterations are carried out. Implementation of a similar method in a different environment may require different heuristics for adjusting the parameters. The key point is we adjust the parameters to achieve a certain result because we have significant information about the object we are detecting. This is often the situation in industrial and medical applications, in contrast, say, to surveillance applications.

The direct line detection algorithm avoids the traditional two-stage edge detection and line traversal process. For each block, total number of arithmetic operations T_{ops} is:

$$T_{ops} = T_g + N_{iter}(T_m + T_l + T_{mi} + T_{ei} + T_c) \tag{1}$$

where $T_g = n^2 T_{go}$, is the number of operations for computing gradient, T_{go} is the number of operations of gradient operator; $T_m = n^2$, is the number of operations for finding P_m; $T_l = c_l$, is the number of operations for deriving line L, a constant; $T_{mi} + T_{ei} = 3n^2$, is the total number of operations for computing M_i and E_i (division ignored); $T_c = 3$, is the number of comparisons to determine whether to accept L; N_{iter} is the number of subdivisions over B. Since the blocks can only be subdivided for a finite number of times before they become too small to yield any significant line segments (in our case 2), the amount of computation per pixel is the gradient operator plus a constant overhead:

$$T_{avg} = T_{ops} / n^2 = T_{go} + N_{iter}(4 + (c_l + 3)/ n^2) = T_{go} + const. \tag{2}$$

This line detection method contains ideas from both the Hueckel [9] and Canny [10] edge detectors and could be thought of as a speed-up version of either. Our method avoids the expensive computational cost of [9] and the search along edge directions in [10]. However it also shares a non-desirable feature of the latter, leaving gaps at corners. There are two potential problems with methods of this kind: (1) missing an "edge" and (2) finding an "edge" that does not exist. An edge can be missed because of averaging of the intensities over a large neighborhood. On the other hand noise can produce extraneous edges. Most edge detectors operate over small neighborhoods (often as small as 3×3) and tend to suffer more of the second problem than the first. In contrast, our line detector operates on a large neighborhood and therefore is subject to the first problem. In particular, if there are two edges in a region, it may be classified as uniform. Because the

objects of interest occupy most of the images, their outlines (and corresponding edges) are far from each other except at corners. Therefore edges might be missed only near corners. We take advantage of the prior knowledge of the shape of the objects to deal with this issue, as described later.

The line segment thus found are grouped into longer segments by combining segments that have similar slopes and proximate endpoints. Doing that for all possible pairs requires computation of $O(n^2)$, where n is the number of line segments. We optimize the aggregation process by using prior knowledge from the direct line detection. In direct line detection, each line segment is extracted from a block (or a subdivided one), which gives us the location of the line segment. By keeping track of the block where each segment is extracted from, we can quickly determine the approximate distance between any pair of segments, which results in only $O(n)$ computational cost. A matrix is initialized such that each element of the matrix corresponds to a 4×4 block in the image (the smallest possible block size), containing indices to line segments whose end points fall inside this block. The matrix is set up during the first step of the process. Then only segments in adjacent blocks are examined for merging. The slope of the segments determines whether the merger attempt will be made. The matrix is updated every time two segments are merged so that the newly merged segment can be examined for further merging. Because the length of the line segment is an indication of reliability, we favor long line segments by giving them higher distance and slope threshold so that they are blended more aggressively.

In case of cylindrical objects, arcs and curves are approximated by a set of short line segments so that in later processing stages they are treated in the same way as their linear counterparts.

2.2. Clustering

The next step is to group together the line segments that belong to the same solid object. We solve the problem through proximity clustering. A proximity cluster is defined as follows:

> A proximity cluster is a set of line segments $\boldsymbol{L}$ that for each line segment s in $\boldsymbol{L}$, there exists at least another segment t such that s and t have at least a pair of endpoints that are near each other.

To detect proximity clusters, we construct a graph with line segments as its nodes. There is an edge connecting two nodes if and only if the two line segments corresponding to the two nodes have at least a pair of endpoints that are near each other. Thus the problem of finding proximity clusters is equivalent to finding connected components in a graph, which can be easily solved by a Breadth-First Traversal. The result of the traversal gives a set of "proximately" connected line segments, or clusters. A filter process is then applied to each cluster for further consolidation. If a cluster is enclosed in another, it is removed since it does not contribute to the object contour. Also evaluated are the average

length, length of the longest line segment, number of line segments and geometric center of the clusters to filter out those that are unlikely to be part of a solid object. The latter decision is made if:

1. The average length of a line segment in the cluster is too small (under 30 in our implementation) and the length of the longest line in the cluster is less than 60; or

2. The cluster has too few lines (less than half of the total number of lines); or

3. The geometric center of the cluster is not in the middle of the scene (we define the middle of the scene as 80 pixels from left/right boundary and 60 pixels from top/bottom boundary).

The imput image is 640 × 480 pixels. Figure 3 illustrates the results of the method. While one-to-one correspondence between proximity clusters and solid object is observed often (Figure 3(a)) it is not always the case (Figure 3(b) and 3(c)). Therefore our algorithm cannot rely on such an assumption.

At the end of this step, we also compute the "main" cluster, which has the largest summation of length of all lines segments within the cluster.

2.3 Finding and Editing the Convex Hull

Given the constraint that the object is texture-bound, contrast is not a good indicator because it may have higher values in the area of graphics than in the lines corresponding to object boundaries. Length is not reliable either because boundary lines may be broken due to low contrast or shadows. Objects cluttered around may have even longer lines than the object edges. The only reliable criterion is that the lines due to graphics should be inside the convex hull of the line segments (we assume the object is convex). Because of our assumption that all objects of interest are convex solids, the 2D contours of their projection in the image should be convex polygons. Further, given the knowledge of the geometric shape of the object, we can derive characteristics of its convex hull and use them to verify whether the convex hull has a reasonable shape.

Ideally a cluster should correspond to a single object, and the contour can be found as the convex hull of the cluster. However, this is not always the case. If parts of an object have low contrast with their surroundings the object may have more than one cluster (Figure 3(b)). Conversely if two (or more) objects are close to each other a single cluster may correspond to two (or more) objects (Figures 3(c)). Thus we need to compute the convex hull of either a single cluster or groups of them.

Let $\boldsymbol{K}$ denote the set of lines in a cluster or in a group of clusters for which we want to find the convex hull. Because many of the line segments found in the previous steps are due to noise, we found it both more efficient and more reliable to start constructing the convex hull by choosing the N longest lines in $\boldsymbol{K}$ (N can be either empirically chosen or

defined by a function proportional to number of lines in $\boldsymbol{K}$) Then a standard algorithm [11] is used to compute the convex hull of the endpoints of the N line segments. Figure 4 shows the results of applying this method. The example is challenging because the image contains three proximity clusters, only two of which belong to the object of interest.

We compute the convex hull of the main cluster as well as the convex hull of all clusters. Editing both the convex hulls is time consuming and unnecessary as only one of them contains the target object. Thus there is a need to develop criteria both for determining whether the found convex hull is indeed the outline of a single object and, if not, to try to *automatically* edit it so that it is indeed such an outline. Towards these objectives, we introduce the concepts of confidence of convex hull edges and the quality of a convex hull, and the functions used to evaluate their values.

The confidence of a convex hull edge is defined as follows:

Let $\boldsymbol{K}$ denote the set of lines in a cluster (or a group of them) from which the convex hull is computed. Let $\boldsymbol{CH_K}$ denote the convex hull of $\boldsymbol{K}$. Let L_i be a line segment in $\boldsymbol{K}$ and E_j be an edge of $\boldsymbol{CH_K}$. Then the contribution of L_i to the confidence of E_j is denoted by c_{ij} and is computed as following:

If L_i and E_j are not collinear, then $c_{ij} = 0$
If L_i and E_j are collinear and both their endpoints are close, then $c_{ij} = 1$
Otherwise $c_{ij} = \text{Length}(L_i)/\text{Length}(E_j)$

By definition of the convex hull, the value obtained by the last expression is never greater than 1 and thus $c_{ij} \leq 1$.

The confidence of E_j is denoted by C_j which is defined as the sum of contributions of all line segments in $\boldsymbol{K}$ with maximum value of 1:

$$C_j = min\ \{\ 1, \sum c_{ij}\ \} \tag{3}$$

where the summation is over all the lines in $\boldsymbol{K}$. In other words, if an edge of the convex hull is collinear with one or more of the line segments found in the earlier step, it will be given a high confidence. The confidence of a convex hull C_c is defined as the summation of the confidence of all its edges:

$$Cc = \sum_{j=1}^{n} C_j \tag{4}$$

Further refinement of the editing is based on the prior knowledge of the geometric shape of the object. The 2D projection of a rectangular box is a hexagon. Under the assumption of the orthographic model, the hexagon has the following properties: each of its internal angles is greater than 90 degree; the pair of opposite edges should have equal lengths and be parallel. These properties do not hold under perspective projection but they provide a guideline of how to determine the quality of the convex hull.

We use function *g(r)* given by equation (4) as a figure of merit indicating whether two line segments have approximately equal length:

$$g(r) = \frac{(r-al)^2(r-ar)^2}{(1-al)^2(1-ar)^2} \qquad \text{if } al < r < ar, \text{ 0 otherwise} \tag{4}$$

where r is the ratio of the lengths of the two line segments. The function *g(r)* is bell-shaped, achieving its maximum value 1 when r equals 1 provided that $al+ar=2$. al and ar are the minimum and maximum ratio values for accepting near length. In our implementation we have chosen al equal to 0.8 and ar equal to 1.2.

Approximate parallelism is estimated by function $z(l_1, l_2)$ that attempts to compute the intersection of the two line segments l_1 and l_2 . If the lines are parallel (no intersections can be computed) the return value is 2. Otherwise the distances between the intersection point and the four endpoints of the line segments are computed, each normalized by the length of the respective segment. If the smallest of the normalized distances, d_{min} is greater than 6, the lines are considered to be practically parallel and the return value is 2. If d_{min} is less than 3 the return value is 0, otherwise it is a linearly interpolated valued between 0 and 2. This is a heuristic formula and the specific parameters should be adjusted based on the physical characteristics of the camera used to capture the image. Figure 7 illustrates the significance of this heuristic.

We define the quality function C_q of a convex hull under the assumption that the convex hull is a hexagon (for cylindrical objects we locate the two vertical convex hull lines and resample the two curves to construct a hexagon so that we can use the same quality function). As we discussed earlier, the shape of 2D projection of a rectangular box is determined by two factors: the approximate equality in length and parallelism. Thus C_q is defined as

$$Cq = \sum_{i=0}^{2} g\left(\frac{l_i}{l_{i+3}}\right) \times z(l_i, l_{i+3}) \tag{5}$$

where i indicates the pairing of convex hull edges (for example, edge 0-3, 1-4, 2-5 as in Figure 5(f)). Because the maximum value of *g()* is 1 and that of *z()* is 2, the maximum value of Cq is 6. The minimum value of Cq is -1 when the convex hull is not a hexagon.

Now we define the complete editing algorithm described as below:

Step 1: Compute the quality and confidence for convex hulls of both the main cluster and all clusters, denoted as C_{qm} and C_{qa}, C_{cm} and C_{ca}.

if ($C_{qm} > C_{qa}$) use the main cluster;
else if ($C_{qm} < C_{qa}$) use all clusters;
else if ($C_{cm} > C_{ca}$) use the main cluster;

else use all clusters;

All following editings are conducted on the cluster(s) determined in this step.

Step 2: This step involves the detection of sharp angles. Projections of rectangular or cylindrical objects should not have any acute angles. Thus lines that have contribute to the creation of such an angle are removed, the convex hull is recalculated, and the process is repeated until no such angles are found.

Step 3: Compute the edge confidence for all the edges of the convex hull.

Step 4: If all edges have high confidence go to Step 6. Otherwise identify the longest sequence of low confidence edges (for example, edge 1, 2 in Figure 4© and edge 5, 6, 0 in Figure 4(d)).

Step 5: Eliminate the low confidence sequence by removing all lines that contribute to it. Re-compute the convex hull and go to Step 2.

Step 6: Locate the most distorted vertex of the convex hull by examining parallelism and edge confidence and remove lines associated with it.

Step 7: Evaluate the quality function *Cq* and if it is below a preset threshold, we exit and decide that the object cannot be located, otherwise go back to Step 2. We also keep track of the values of *Cq* and if its value decreases, we reverse a sequence of steps and exit.

Figure 5 shows an example of the editing process for a rectangular box. After a sequence of low confidence edges is removed (together with the lines that contributed to their generation) the convex hull is recomputed and the editing process is repeated. We stop the editing process if the outline has a reasonable shape and further editing introduces more distortions (it is easy to show that *Cq* achieves a maximum when the convex hull is a hexagon with opposite pairs of sides being parallel and of equal length).

From the outline, we derive the "Y", which is constituted by the three internal edges of the box, by using the vanishing points formed by the pair of corresponding external edges. Together with the three edges of the Y in dotted dash lines, these nine line segments complete the 2D outline of the box, as shown in Fig 5(f), which is then used to calculate the dimensions.

The process of editing the convex hull can also be seen as the first step in a drawing beautification process with the *Cq* being a measure of how close the outline is to that of a hexagon with pairs of opposite sides that are parallel and of equal length. Then the method used in the second step of [8] could be used here to produce a hexagon where these properties are satisfied exactly. Because the dimensions inferred from the existing shape were accurate enough (see Section 3) we did not implement that step.

3. Experimental Results

We experimented extensively with the convex hull approach on various boxes and cylindrical objects. The results are encouraging.

Figure 5 illustrates how the convex hull approach is used to locate a textured box in a gray-scale image. The original image is shown in Figure 1(a). By evaluating the quality and confidence function, we choose the convex hull of all clusters to edit since it has a more reasonable shape than the convex hull of the main cluster (the black solid cluster in Figure 1(c)).

Figure 6 gives another example of locating a textured cylindrical object. In this example, the convex hull of the main cluster (the black solid cluster in Figure 6(c)) is chosen for further editing as it has higher quality metric.

The complete system used for experiments (including the method described in this paper, the image capture system, the laser based absolute distance estimator) is described in details in [1]. The hardware consists of a handheld CCD camera with resolution of 640 × 480 and a fixed focal length at 12 inches, a 24MHz PowerPC 400 series microprocessor where the algorithm runs, and a pair of laser beams for estimating the absolute distance (one laser beam suffices, two gives more accuracy). The system performed in a satisfactory way for the intended industrial applications, such as storage planning of shipments. Figure 8 shows typical results. The error in Example 2 (Figure 8(b)) is relatively large as one of its vertex is extended by the shadow. The problem of shadows and reflections are to be studied in the future.

We have omitted several implementation details because they depend on the specific environment where our algorithm was implemented, including the computational platform and the image capturing device (also an exact replication of the method may run into problems with the Symbol Technologies patent).

4. Conclusions

We have presented a new approach to identify objects with known shape from a single gray-scale image. The novelty of our work is the usage of convex hull, a high level feature not given much attention in the image using literature. While what we have described in this paper appears to be a bottom-up image analysis method, it is in essence a top down method because we incorporate prior knowledge about the objects in the low-level image processing. The convex hull approach is designed to handle objects with rich textures on the surface, which is common in industrial environments. It can also handle background clutter. Its advantage in speed makes a real-time implementation on an embedded system possible, which is another highly constrained requirement in industrial applications. Successful experiments on various boxes and cylinders show the effectiveness and advantages of our algorithm.

Industrial applications are ideal for the integrated approach we describe here because there is significant prior knowledge about the scene and the image capturing device may be designed in a way that benefits the image analysis step. The concept of detecting approximate relationships to infer exact ones seems to be quite powerful and has been used recently by Gribov and Bodansky in cartographic applications [12, 13].

Currently either the main cluster or all of them are used to derive convex hull. Possible merging of selected clusters by their position, shape, scale etc. can simplify the editing. On the other hand, splitting of clusters may extend the algorithm to handle two or more target objects in the scene. Machine learning techniques can be used to further improve the editing process. The current system cannot deal with shadows and reflections very well since the edges created by them overwhelm the object features. It cannot deal with occlusion either. These are to be studied in the future.

References

1. T. Pavlidis, E. Joseph, D. He, E. Hatton, and K. Lu, "Measurement of dimensions of solid objects from two-dimensional image(s)" *U. S. Patent 6,995,762*, Feb 7, 2006 (Patent has been assigned to *Symbol Technologies*).

2. D. Katsoulas, "Reliable Recovery of Piled Box-like Objects via Parabolically Deformable Superquadrics", *ICCV* 2003, pp. 931-938.

3. T. Shakunaga, "3-D Corridor Scene Modeling from a Single View under natural Lighting Conditions", *IEEE Trans. PAMI,* Vol. 14, No. 2, 1992, pp. 293-298.

4. C. J. Taylor and David Jelinek, "Reconstruction of Linearly Parameterized Models from Single Images with a Camera of Unknown Focal Length", *CVPR* 1999, pp. 2346-2352.

5. Y. Lamdan and H.J. Wolfson, "Geometric hashing: A general and efficient model-based recognition scheme", *ICCV* 1988, pp. 238-249.

6. D.P. Huttenlocher and S. Ullman, "Recognizing Solid Objects by Alignment with an Image", *IJCV* 1990, pp. 195-212.

7. M.S. Costa and L.G. Shapiro, "3D Object Recognition and Pose with Relational indexing", *Computer Vision and Image Understanding* (79) No. 3, 2000, pp. 364-407.

8. T. Pavlidis and C. J. Van Wyk, ``An Automatic Beautifier for Drawings and Illustrations," *Proc. of SIGGRAPH'85*, 1985, pp. 225-234.

9. T. Pavlidis, *Structural Pattern Recognition*, Springer-Verlag, 1980, pp. 81-83.

10. J. Canny, “A Computational Approach to Edge Detection”, *IEEE Trans. PAMI*, Vol. 8, 1986, pp. 679-698.

11. R. Sedgewick, *Algorithms*, second edition, Addison-Wesley, 1988, Chapter 25, pp. 359-372.

12. A. Gribov and E. Bodansky “A New Method of Polyline Approximation,” *Lecture Notes in Computer Science*, Vol. 3138/2004, Springer, pp. 504-513.

13. A. Gribov and E. Bodansky, “Reconstruction of Orthogonal Polylines,” *Proceedings of DAS 2006*, H. Bunke and A. L. Spitz, editors, Springer-Verlag, 2006, pp. 462-473.