A new algorithm to group lines converging to vanishing points in perspective sketches of polyhedra

Pedro Company, Peter A.C. Varley
Institute of New Imaging Technology
Universitat Jaume I
Castellón de la Plana, Spain
e-mail: (pcompany, varley)@uji.es

Raquel Plumed
Dept. of Mechanical Engineering and Construction
Universitat Jaume I
Castellón de la Plana, Spain
e-mail: plumed@emc.uji.es

Abstract—**We seek to detect the vanishing points implied by design sketches of engineering products. Adapting previous approaches, developed in computer vision for analysis of vectorised photographic images, is unsatisfactory, as they do not allow for the inherent imperfection of sketches. Human perception seems not to be disturbed by such imperfections. Hence, we have designed and implemented a vanishing point detection algorithm which mimics the human perception process and tested it with perspective line drawings derived from engineering sketches of polyhedral objects. The algorithm returns the approximate locations of the main vanishing points and identifies those groups of lines in 2D which correspond to groups of parallel edges in the 3D object.**

Keywords— Sketches; Perspective; Vanishing points

I. INTRODUCTION

Our area of interest is creating computer-based tools to help design engineers during conceptual design (the first stage of the design process). For sketch-based modelling (SBM) systems to become a valid alternative to both current WIMP-based CAD systems and traditional paper and pencil sketching, they must cope with the full range of conceptual design sketches. Although most such sketches are done in orthographic projection style [1], it is also important to allow for perspective projection.

As explained in section II, some of the most popular vanishing point detection algorithms are compatible with human interpretation and may be tuned to mimic human perception [1-3], but none of them copes satisfactorily with the inherent imperfection of sketches. Hence, we have designed and implemented a new algorithm, specifically aimed at finding vanishing points (VPs) in sketches of engineering design products.

Section III describes our algorithm.

Section IV presents our test results. Section V presents conclusions and recommendations for future work.

II. RELATED WORK

The input to our algorithm will be a set of lines. In discussing related work, we only consider approaches which takes lines as input (we exclude those such as Barnard [6] and Magee and Aggarwal [7] which require bitmaps). We also exclude those such as Varley [8] which use (or attempt to deduce) higher semantic level information.

We note that most methods for detecting VPs are intended for 2D camera images. The errors they deal with (lens imperfections and noise in line segment extraction) are much smaller than typical sketching errors. Of these methods, the clustering approach of McLean and Kotturi [MK95] is most tolerant to noisy data.

Tardif [4] is interesting as it deals with one of the problems we consider here: its input is a set of *N* sparse edges, and its output is a set of VPs and a classification for each edge (assigned to a VP or marked as an outlier). It also includes a clustering strategy which improves on McLean and Kotturi [2]. However, it uses the J-Linkage algorithm, which is a (a) computationally expensive and (b) non-deterministic, so only produces “probable” rather than “definite” results.

Rother [3] is a recent and representative example of a group of approaches which explore all candidate VPs, a reasonable choice when the aim is to detect VPs in a line drawing with a small number of strokes. This has the major drawback that its accumulation step only works well for geometrically perfect inputs—errors in geometry would result in a set of neighbouring cells each being visited only once, resulting in a set of non-dominant cells instead of the single dominant cell which the algorithm requires to predict the location of the dominant VP. A further problem is that it only works well for the “Manhattan world” of normalons, and cannot reliably find oblique VPs.

A recent study by Plumed et al. [5] gives criteria and metrics for implementing algorithms which mimic human perception in detecting vanishing points in design sketches. Although these have proved useful during the design and implementation of our algorithm, we must highlight a substantial difference between [5] and the approach we propose here. Since the interviewed people were aware of the nature of the depicted object, they first perceived the object (as a "step", a "house", …), then spontaneously grouped edges which they know to be parallel in 3D, and finally checked whether the corresponding 2D lines were parallel or converged to a VP. However, our algorithm begins with a set of unclassified 2D lines—there is no high semantic level information about the object—and determining groups of parallel edges is one intended output of the algorithm.

III. Algorithm

The input for Sketch-Based Modelling approaches is a sketch and the output is a 3D model. We assume that the sketch depicts a single object. This paper deals with one intermediate stage of the process, in which a line drawing is parsed to get higher semantic-level information. The input for this stage is a list of lines (where a line is defined by two endpoints, each of which is an (x-y) coordinate pair). The output is a set of groups of lines in 2D which correspond to groups of parallel edges in the 3D object; each group has either one vanishing point (perspective projection) or none (parallel projection).

We have followed the idea used by Rother [3] of exploring all candidate VPs, and the idea of clustering present in McLean and Kotturi [2] and Tardif [4]. Two key ideas make this algorithm different from previous approaches. (1) We cluster candidate vanishing points instead of clustering lines. This prevents the need for repeatedly measuring complex distances (as in the voting distance in [3] or the vanishing point estimator in [4]), and allows clustering in a way more tolerant of imperfections in the sketched lines. (2) We use a polar coordinate system whose origin is the image centroid, and measure cluster similarity as difference in the orientation (angle), regardless of the distance (radius)—sketching imperfections produce far more uncertainty in distances than in orientation [5]. Thus we deal in a natural way with discrepancies in the VP position estimated for a group of strokes.

There are four stages to the algorithm: (1) enumerating all possible candidate vanishing points; (2) forming clusters of vanishing points; (3) filtering intersecting edges; (4) filtering repeated edges.

A. *Enumerating Candidate Vanishing Points*

A candidate VP is created wherever the extensions of two lines cross. The data stored for each VP are the two lines and the position, calculated in polar coordinates R_{VP} and θ_{VP} relative to the drawing centroid. R_{VP} is scaled to the size of the drawing ($R_{VP}=1$ is the largest distance between the centroid and any line endpoint).

According to [5], humans generally only perceive candidate VPs where R_{VP} is between 1.5 and 5. Thus, for the purpose of detecting VPs, only candidate VPs within this ring should be output from this stage and should be passed on to the clustering process.

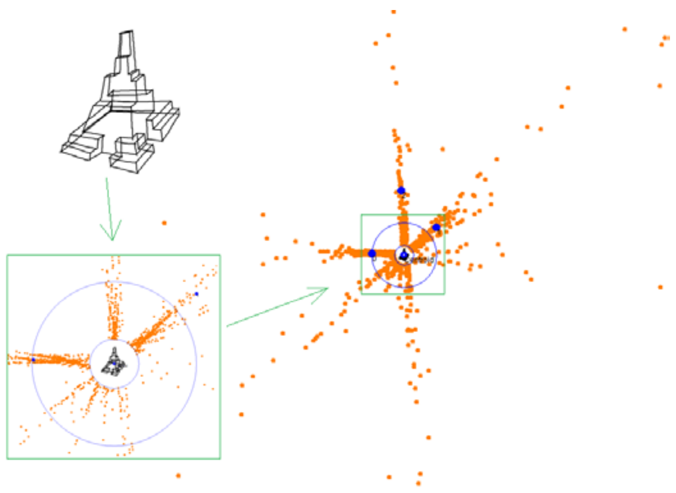
However, we also wish to identify non-convergent groups of lines. In practice, groups of (nearly) parallel lines also produce clusters of candidate VPs, but the clouds of such clusters are longer and their centroids are typically more distant than those of convergent lines. In order to include these, we do not use an outer radius limit.

B. *Clustering*

A cluster of VPs is a list of one or more VPs, together with a mean orientation angle θ_C.

Initially, each VP is its own cluster; larger clusters are grown by merger of two smaller clusters. Iteratively, we identify the closest pair of nearest-neighbour clusters and merge them, until the smallest difference between any pair of neighbouring θ_C is greater than a threshold. We set this threshold to 12°, as suggested in [5].

Fig. 1. Drawing, candidate VPs and clustering

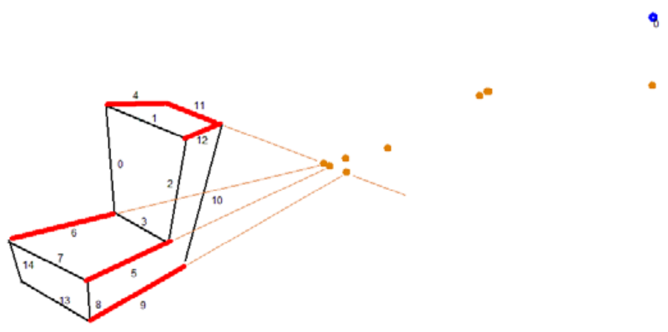


C. *Filtering intersecting lines*

Since lines are stored for each VP, a cluster is not only a list of VPs but also a list of lines. All of these lines are presumed to correspond to parallel edges of the 3D object. Clearly, any pair of lines which cross within the object cannot be parallel in 3D and thus should not appear in the same cluster. We use a filtering process to remove lines (and VPs) from clusters when this rule is breached.

Figure 2 shows an example where this filter is required: the intersection points of line 11 with lines 4 and 12 are internal, but the intersections of 11 with 5, 6 and 9 meet the criteria for valid candidate VPs and have been clustered with the intersections of 4, 5, 6, 9 and 12. Line 11 must be removed from this cluster.

Fig. 2. Vanishing points from intersecting lines



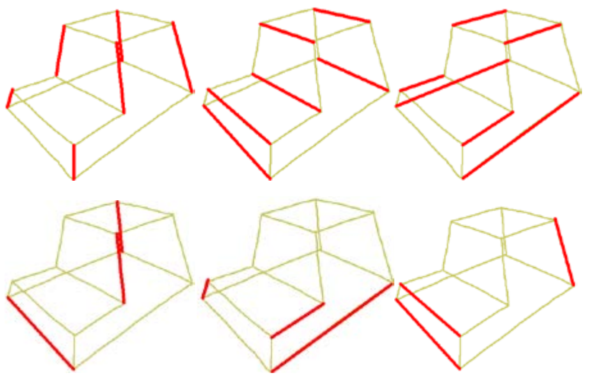
Detecting intersecting lines is straightforward (we have already done this while enumerating candidate VPs).

Deciding which line to exclude is more complicated. Where a conflict exists: 1) the line contributing to more VPs in the cluster is retained; 2) where the conflicting lines contribute to the same number of VPs, the line with VPs closer to the centroid is retained.

D. Filtering repeated lines

Most lines produce more than one candidate VP, and clustering VPs often results in some lines appearing simultaneously in different clusters. See, for example, Fig 3.

Fig. 3. Clustering output, with lines in more than one cluster



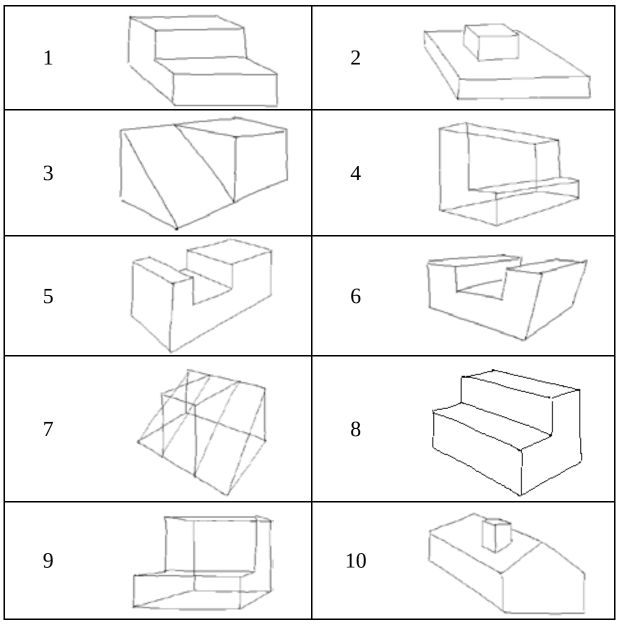
Since each cluster corresponds to a group of 3D-parallel edges, each line may belong to no more than one group. We add a second filter to remove repeated lines.

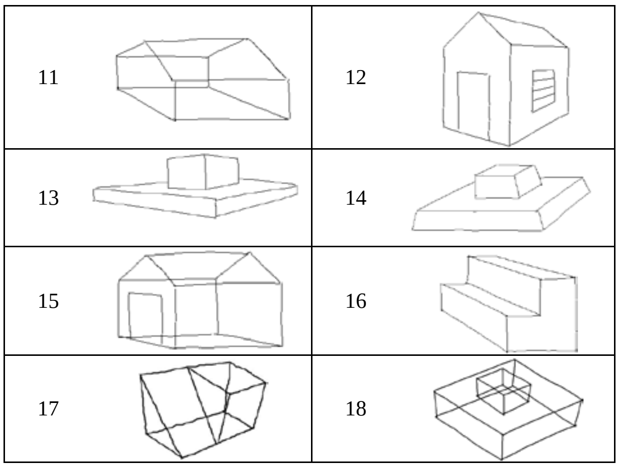
We order clusters in descending order of (number of VPs + number of lines), following the heuristic that the most populous clusters are the best perceived ones [5]. For each cluster after the first, we remove any lines which also appear in any more populous cluster. Where this results in a cluster with fewer than two lines, the cluster itself is deleted. Thus, in the example in Fig. 8, only the first three clusters would remain.

IV. ANALYSIS

We have tested our approach using the 18 examples in [5] (figure 4). It can be seen that some of these are natural line drawings, while others are wireframes—the algorithm should (and does) work equally well for both.

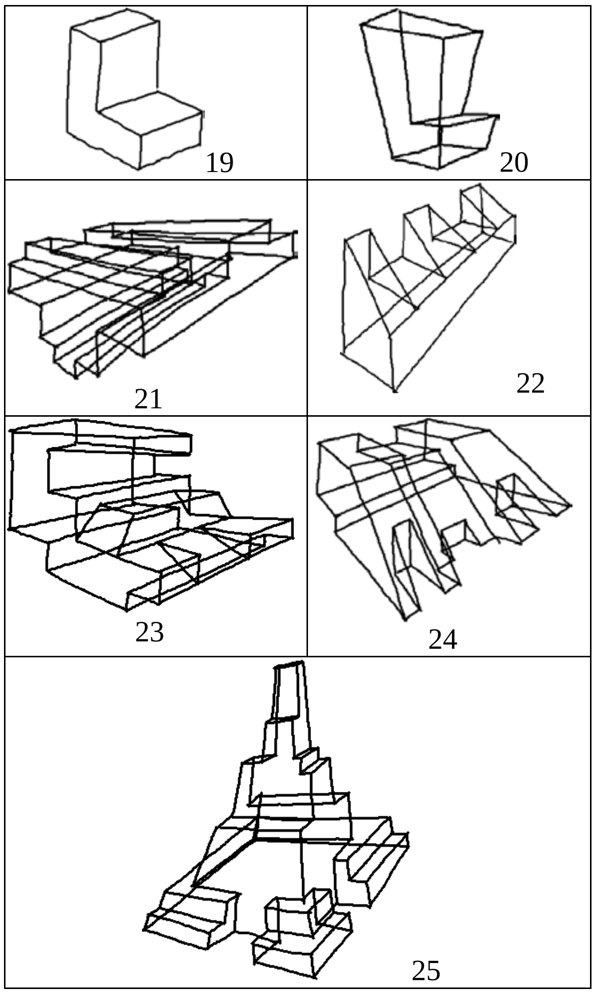
Fig. 4. Test examples





To this, we have added tested two variants of an angle bracket, with respectively zero and three perceptual VPs (19 and 20 in Figure 5), and five more complex drawings (21 to 25 in Figure 5).

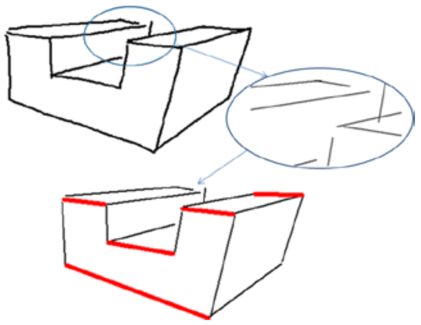
Fig. 5. Additional test drawings



For most test drawings, the results are the same as human perception. The detailed results are available in the annex.

For drawing 6, very poor quality sketching prevents vectorisation from creating a valid line drawing, and as a result our algorithm fails to group one edge correctly (Figure 6). The same problem also occurs with drawing 13. In both cases, a slightly better sketch results in the correct grouping. We note that humans can group correctly even these poor quality sketches, but this is because they use high semantic level information (candidate edges are collinear with other edges in the group, or are opposite other edges in the group in the same quadrilateral face) which is not available to the algorithm—the algorithm is designed to be independent of other perceptual cues (such as collinearity or faces) as it may precede detection of such cues.

Fig. 6. Poor quality sketching results in incomplete grouping



Poor sketch quality is even more critical for nearly-parallel lines, such as the vertical set of lines in examples 7, 8, 9 and 16, which humans perceive as non-convergent. Because of sketching errors, the orientations of such lines are all close to, but not exactly, the intended angle alpha, producing some candidate VPs located around alpha and others located around alpha+180°. This "complementary convergence" can be seen in even in Figure 1, but for clearly convergent lines it does not affect the results as the superfluous clusters will be removed by the second filtering stage.

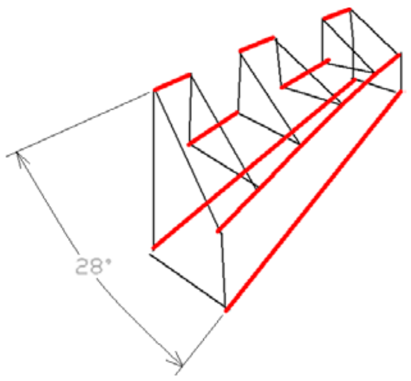
Apart from very poor quality sketches and complementary convergence, the algorithm always detects the correct three groups of lines for normalon ("Manhattan-like") shapes where all edges and face normals are aligned with one of three main perpendicular axes. The algorithm succeeds irrespective of the number of actual vanishing points—it also succeeds when lines represent edges intended to be parallel. For example, the different representations of the angle bracket have been perceived by humans as having one (examples 1, 9, 16), two (4 and 8), three (20) and zero (19) VPs.

At first sight, these results appear similar to those reported for other approaches for grouping parallel lines currently used in SBM. However, such approaches fail when the 2D lines corresponding to parallel edges span angles as big as 28°, as in Figure 7. Our new algorithm processes this and similar drawings correctly.

Sensitivity analysis shows that our algorithm is, in general, robust, but there are three exceptions: examples 15, 23 and 24. In these examples, combinations of sketching errors, large convergence angles, and complementary convergence from nearly-parallel lines, produce results which are very sensitive to tuning parameters such as the inner radius.

Counting loops shows that the algorithm is O(n^4) in theory. Its practical time complexity has not been determined: even with example 25, a figure at the limit of anything anyone is ever likely to sketch, the algorithm takes less than 50 ms. We can conclude that timing is never going to be a problem.

Fig. 7. Grouping parallel edges represented by converging lines



V. Conclusions

We have presented a new approach for finding vanishing points, tailored to sketches of polyhedral objects. Our approach improves on previous approaches in two ways: (1) it allows for inherent sketching errors, which defeat approaches designed for nearly-perfect line segments extracted from cameras images from cameras; (2) it derives from a study of human perception rather than arbitrary geometric criteria.

Our preliminary tests show that the algorithm is reasonably successful in matching human interpretation (where humans do better, it is by making use of high semantic level information). More work is still required to automatically detect complementary convergence and perfectly-parallel lines.

Acknowledgment

The Spanish Ministry of Science and Education and the European Union (Project DPI2007-66755-C02-01), and the Ramon y Cajal Scholarship Programme are acknowledged with gratitude.

References

[1] Plumed R., Company P., Piquer A. and Varley P.A.C. "Do engineers use convergence to a vanishing point when sketching?" Proc. Int. Symposium on Distributed Computing and Artificial Intelligence 2010. (DCAI'10), pp.241-250.

[2] McLean G.F. and Kotturi D. "Vanishing point detection by line clustering". IEEE Transactions on Pattern Analysis and Machine Intelligence, 17(11), 1995, 1090 - 1095.

[3] Rother. C. "A new approach to vanishing point detection in architectural environments". Image and Vision Computing, 20(9-10), 2002, 647–655.

[4] Tardif, J.P. "Non-Iterative Approach for Fast and Accurate Vanishing Point Detection". 12th International Conference on Computer Vision, 2009 IEEE, pp. 1250-1257.

[5] Plumed, R., Company, P., Varley, P.A.C. "Metrics of human perception of vanishing points in perspective sketches". 21st International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision WSCG 2013, submitted. Unpublished.

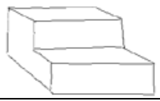
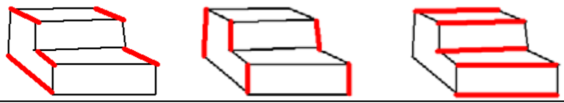
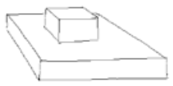
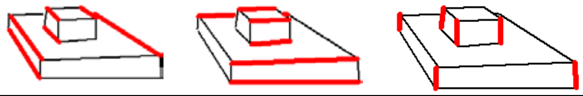
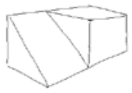
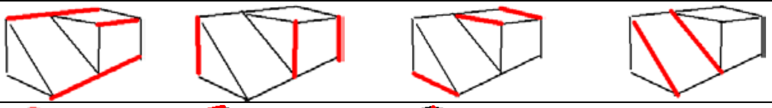
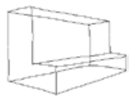
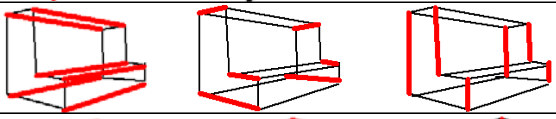
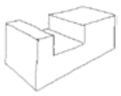
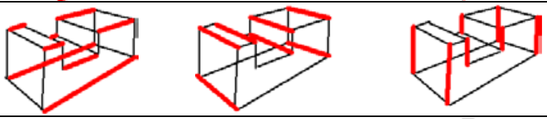
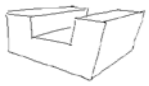
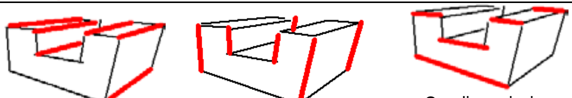
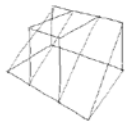
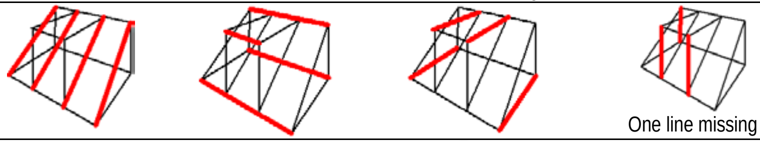
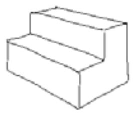
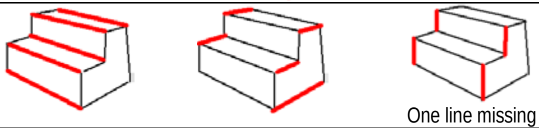
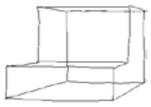
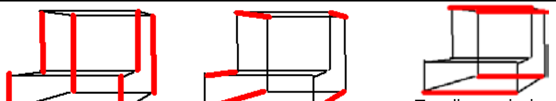
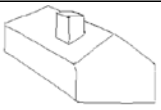
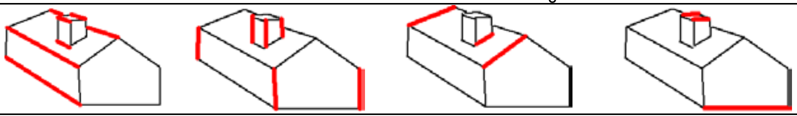
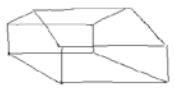
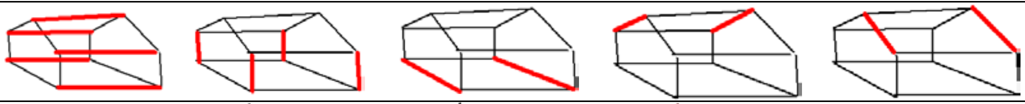
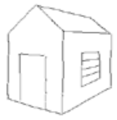
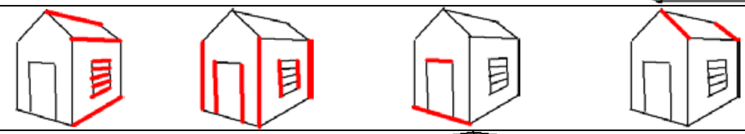
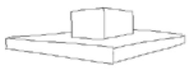
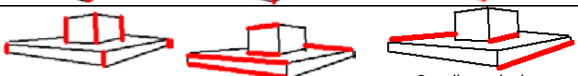
[6] Barnard S.T "Interpreting perspective images". Artificial Intelligence, 21(4), 1983, 435–462.

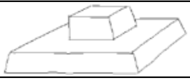
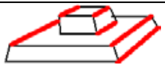
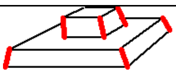
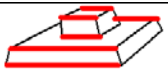
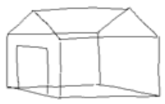
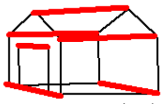
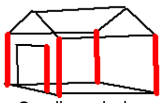
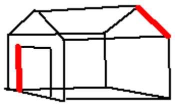
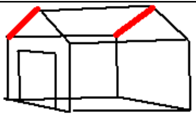
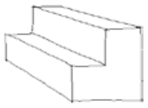
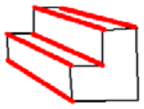
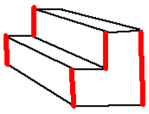
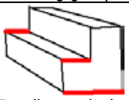
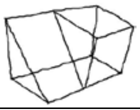
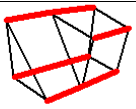
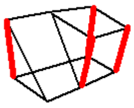
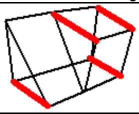
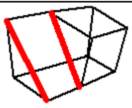
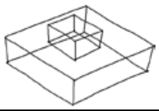
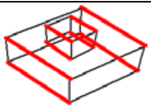
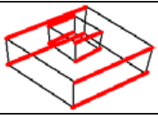
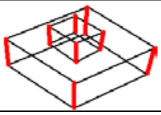
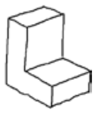
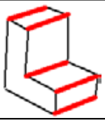
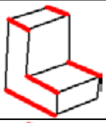
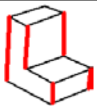
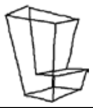
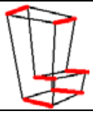
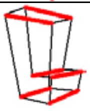
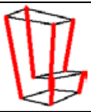
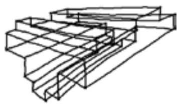
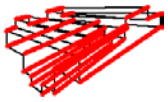
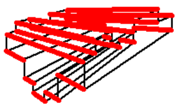
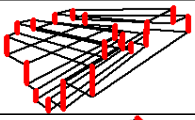
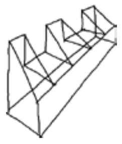
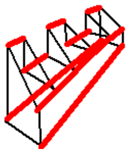
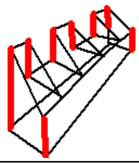
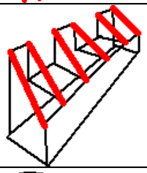
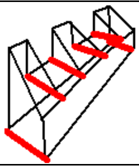
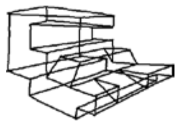
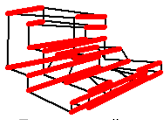
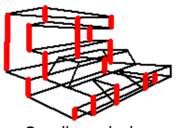
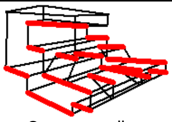
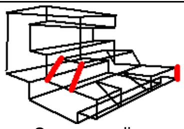
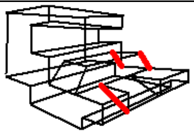
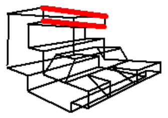
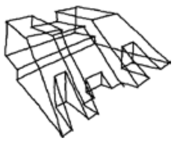
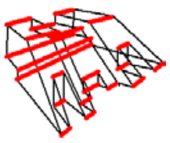
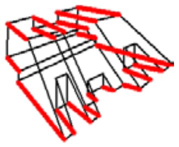
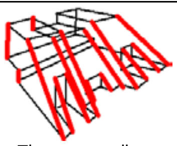
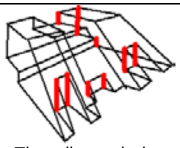
[7] M.J. Magee, J.K. Aggarwal "Determining vanishing points from perspective images", Computer Vision, Graphics and Image Processing 26(2), 1984, 256–267.

[8] Varley P.A.C. Automatic Creation of Boundary-Representation Models from Single Line Drawings. Ph.D. thesis, Cardiff University, 2003.

ANNEX

Groups of parallel edges detected by the algorithm for the 18 examples in reference [1], plus the 7 examples added to further test it, are marked in red thick lines:

1		
2		
3		
4		
5		
6		 One line missing
7		 One line missing
8		 One line missing
9		 Two lines missing
10		
11		
12		
13		 One lines missing

14						
15		 Two groups mixed	 One line missing	 Wrong group		
16				 Two lines missing		
17						
18						
19						
20						
21						
22						
23		 Four wrong lines	 One line missing	 One wrong line Four lines missing	 One wrong line	 One wrong line
		 Sub-group of the first group	Two oblique lines missing (central ramp)			
24				 Three wrong lines	 Three lines missing	
25