An algorithm for grouping lines which converge to vanishing points in perspective sketches of polyhedra

Pedro Company[1], Peter A.C. Varley[1], Raquel Plumed[2]

[1] Institute of New Imaging Technology, Universitat Jaume I, Castellón, Spain
e-mail: (pcompany, varley)@uji.es
[2] Dept. of Mechanical Engineering and Construction, Universitat Jaume I, Castellón, 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 new algorithm is fast, easily-implemented, 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; Parallel edges

1 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 2, 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. Our algorithm clusters candidate vanishing points instead of clustering lines, and measures cluster similarity by angular difference in orientation. This deals naturally with sketching errors, as, when judging where VPs should be located, people are far more tolerant of discrepancies in distance than discrepancies in orientation [4].

Section 3 describes our algorithm. Section 4 presents our test results. Section 5 presents conclusions and recommendations for future work.

2 Related Work

The input to our algorithm is a set of lines. In discussing related work, we only consider approaches which takes lines as input (we exclude those such as Barnard [5] and Magee and Aggarwal [6] which require bitmaps). We also exclude those such as Varley [7] 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 [2] is most tolerant to noisy data.

Tardif [8] 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) 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. Figure 1 illustrates that a coarse grid only returns "the vanishing point is somewhere around here", while a fine grid does not work at all—each vote is in a different box.
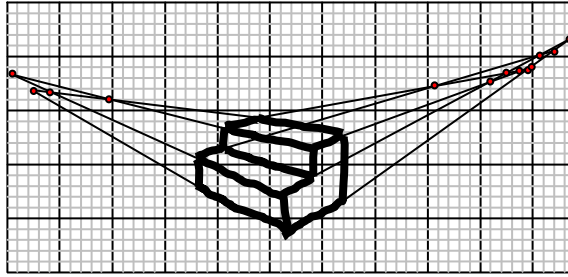
Fig. 1. Grid cannot cope with sketching imperfections

A further problem is that it only works well for the "Manhattan world" of normalons, and cannot reliably find oblique VPs. Nevertheless, [3] is representative of approaches which consider all pairwise intersections of the detected line segments. This technique is impractical for online analysis of photographic images, as it is very time consuming, but it may be useful for sketches, which contain fewer lines.

A recent study by Plumed et al. [4] 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 [4] 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.

We note that first grouping the lines and then finding VPs could be a valid alternative, although it is not as straightforward as it seems. Approaches for grouping 2D lines that represent edges parallel in 3D have been reported in the literature. However, the problem has proved difficult as no general solution has been developed so far. For instance, a naïve algorithm may group lines A and D—instead of A, B and C—in figure 2 (Varley [7]).
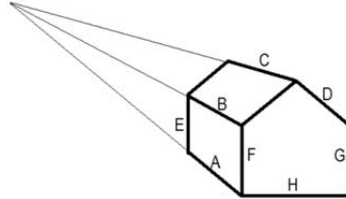

Fig. 2. Which lines should be parallel?

3 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 do not discuss *vectorisation*, the conversion of strokes (time-stamped lists of 2D points) (figure 3 left) into lines (figure 3 centre). But we note that vectorisation does not correct geometrical imperfections inherent in sketching. Vectorisation strategies also affect our clustering algorithm: segmented edges (upper red lines in figure 1 centre) produce more candidate VPs (and more "votes") than their unsegmented equivalent (lower red line). Our algorithm uses minimal vectorisation—for example, it does not use junctions (figure 3 right), because fitting line endpoints into a common junction produces undesired line rotations which corrupt the parallelism intended in the sketch.
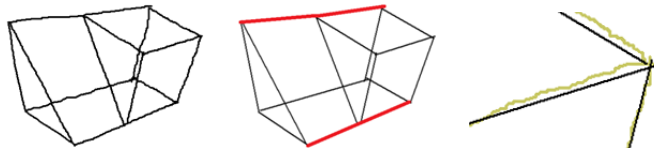

Fig. 3. Strokes (left), collinear edges (centre), and detail of endpoints (right)

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 [8]. Two key ideas make our algorithm different from previous approaches. (1) We cluster candidate vanishing points, instead of clustering lines as in [9]. This prevents the need for repeatedly measuring complex distances (as in the voting distance in [3] or the vanishing point estimator in [8]), and allows clustering in a way more tolerant of imperfections in the

2

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 [4]. Thus we deal in a natural way with discrepancies in the VP position estimated for a group of strokes. Finally, since some wrong clustering still happens, we add various filters to detect and correct mistakes in the groups of edges.

There are eight 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; (5) filtering groups of parallel edges; (6) filtering anti-convergent lines, (7) filtering opposite convergence and (8) resampling isolated edges.

3.1   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).
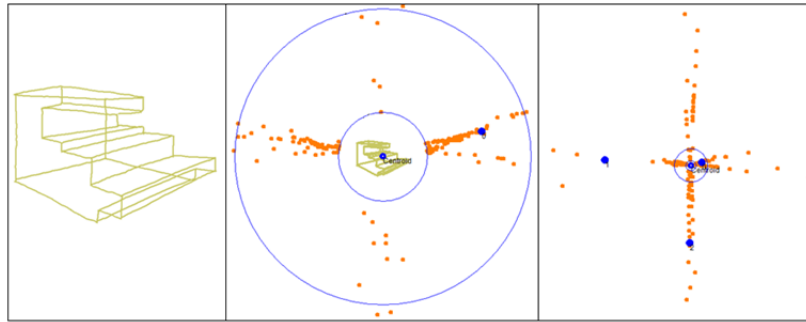

Fig. 4. Sketch (left), ring with candidate VPs (orange dots) in two zooms (centre and right)

According to [4], humans generally only perceive candidate VPs where R_{VP} is between an *inner ring* (Ri) of 1.5 and an *outer ring* (Ro) of 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 (figure 4 shows an example sketch and its ring in two different zooms).

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 to the object than those of convergent lines. In order to allow for these, we first calculate all VPs, regardless of outer radius, and only later classify them as true VPs or spurious VPs (groups of parallel lines).

3.2   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 cutting-distance threshold "*CD*".
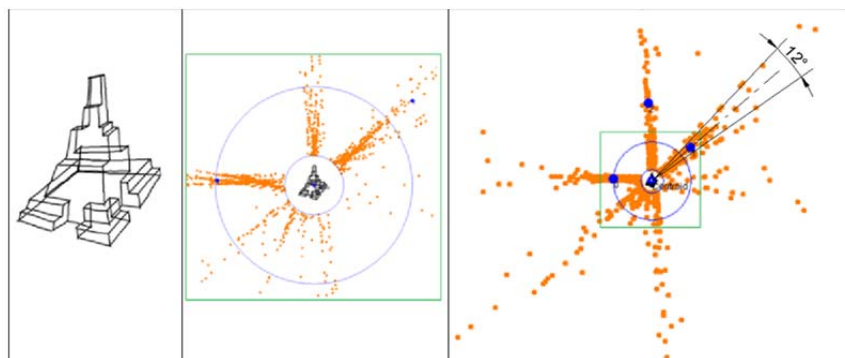

Fig. 5. Sketch (left), candidate VPs and clustering (centre and right)

The pseudo-code for the clustering algorithm is as follows:
1.   Get the θ angles for each of the NumCandidateVP candidate VPs
2.   Create two paired lists (whose initial length is NumCandidateVP):
   •   Orientations list of θ angles

- Cluster list, which is a meta-list where each item is a list which starts having just one candidate VP (the candidate VP whose angle is stored in the paired member of Orientations list)
3. Sort the paired lists in ascending order of θ
4. Define NumClusters= NumCandidateVP;
5. Repeat while NumClusters >1:
   a) Find the position of the smallest MinGap (i.e. min(Gap[i]= Orientation[i+1]- Orientation[i]) ∀i)
   b) Finish if smallest gap (MinGap) is bigger than CD
   c) Add all the candidate VPs from Cluster[i+1] to Cluster[i]
   d) Recalculate Orientation[i] as the angle of the centroid of all candidate VPs in Cluster[i]
   e) Remove Cluster[i+1] and Orientation[i+1] from the list
   f) NumClusters= NumClusters-1

Figure 6 displays the clusters for a simple drawing, and shows that the algorithm does not produce a full dendrogram, but a "truncated" one. A suitable CD parameter guarantees that Mean VPs are not clustered together, as would happen with a full clustering process. Note that the dendrogram is out of scale in the figure to highlight the cutting-distance. We set CD to 12° (figure 5), as suggested in [4].
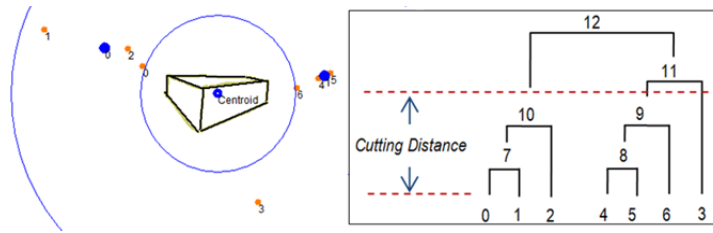

Fig. 6. Candidate and Mean VPs (left) and dendrogram of the clustering process (right)

Mean VPs are calculated as the centroids of their respective groups. Outliers are excluded from this calculation, although they are maintained in the group. Points are considered outliers if they meet Chauvenet's Criterion [10].

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.

3.3 Filtering Intersecting Lines

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 7 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.

When detecting intersecting lines, we must consider sketching imperfections. Our algorithm marks two lines as crossing within the object if they are not collinear and: (a) their endpoints are close to one other (up and down lines in figure 3 right), or (b) their endpoints are close to their intersection (up and left lines in figure 3 right). We implement both criteria as thresholds: the distances (between the endpoints or between the intersection and either endpoint) are less than 0.1 times the length of the original edges (parameter SnapD).
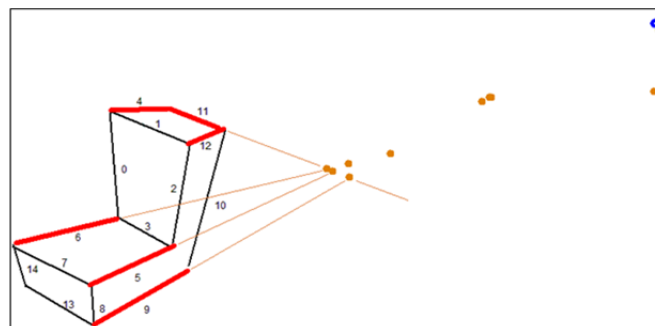

Fig. 7. Vanishing points from intersecting lines

Similarly, we have included a tolerance angle for accepting two nearly-parallel lines sharing an endpoint as collinear (collinear lines may meet at a K-junction [7]), since they are not always sketched as perfectly parallel (figure 1 centre). We set this TolCol tolerance to 5°.

4

These tolerances are appropriate for our test examples, but will require adjustment for better-quality or poorer-quality sketches.

When deciding which line to exclude, we apply two rules: 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.

Lines intersecting the extrapolations of other lines are also removed (unless *nearly* parallel), as the edges they depict cannot be parallel to the rest of the group (for example, the edge depicted by line 10 could never be parallel to the edges depicted by lines 5 or 6 in figure 7).

3.4 Filtering Repeated Lines

Most lines produce more than one candidate VP, and clustering VPs often results in some lines appearing in more than one cluster. See, for example, figure 8.
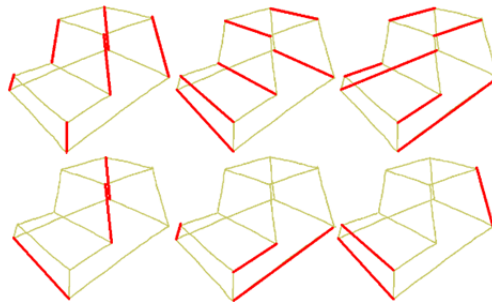

Fig. 8. Clustering output, with lines in more than one cluster

Since each cluster corresponds to a group of 3D-parallel edges, each line may belong to no more than one group. We add a 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 [4]. 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 figure 5, only the first three clusters would remain.

In some cases, retaining the most populous groups is insufficient. For example, the five groups with two lines in example 17 were ordered such that the only group perceived by people (the "ramp") is removed because it is placed after, and shares its two edges with, other equally-populated groups. Worse, these four groups are later discarded, since they share edges with other more populous groups. A second criterion, discriminating among equally-populated groups to give priority to the more useful groups, is required. Hence we apply the Gestalt Law of Similarity and rearrange equally populated groups by assigning the highest priority to those with edges most similar in size (figure 9).
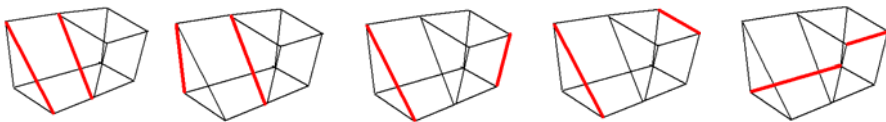

Fig. 9. Equally sized groups rearranged in decreasing order of similarity between lengths of lines

3.5 Filtering parallel edges

The outer ring was ignored in section 3.1, so as to detect groups of parallel edges. Once all groups have been detected, groups of parallel edges must be labelled as such. Two criteria must be met for labelling genuinely parallel groups: (i) if the centroid of the VPs is within the outer radius, the group is convergent; if the centroid of the VPs is beyond the outer radius, the group might be parallel; (ii) parallel groups are disperse; hence, the group might be parallel if the standard deviation of distances between each VP and the centroid of the cloud is greater than 15 times the radius of the bounding box of the drawing (parameter MinDisp). Groups with only two lines are considered to depict parallel edges whenever their VPs are outside the outer ring.

3.6 Filtering Anticonvergent Lines

It is only after filtering parallel edges that we filter anticonvergent lines for convergent groups. We find the Mean Line (the line which passes through the centroid of the drawing and the Mean VP of the group). Anticonvergent lines intersect the Mean Line of the group to which they supposedly belong on the side opposite to the Mean VP. Figure 10 left illustrates two converging lines which intersect the Mean Line near the Mean VP of the group; Figure 7 right illustrates how line 2 converges to the other side.
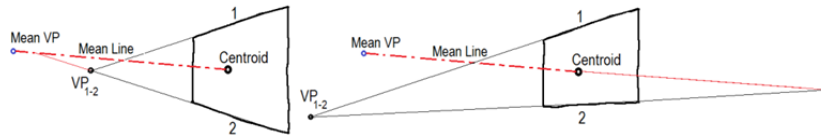
Fig. 10. True convergence (left) and anticonvergence (right) for line 2

Figure 11 illustrates an actual example, in which anticonvergence merges red edges (which clearly converge to the right) with green ones (converging to the left). Our algorithm removes red lines from the group of green lines by repeatedly removing lines (a) which intersect the Mean Line on the opposite side to the Mean VP (marked as a blue dot in figure 8), and (b) whose VP is farthest from the Mean VP (which is recalculated after removing every line). Lines nearly collinear with the Mean Line are considered parallel and are not removed, irrespective of their actual orientation.
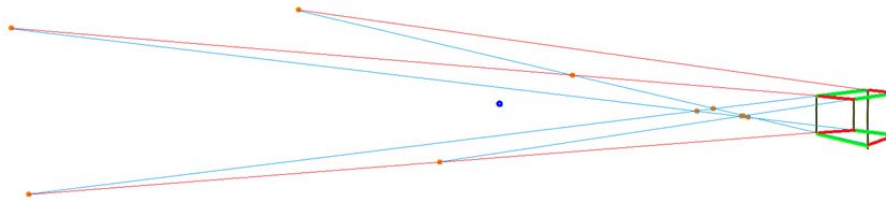

Fig. 11. Red edges do not converge to the left side, where the Mean VP is located

3.7 Filtering opposite convergence

Nearly-parallel lines pose the question: were they intended to meet at a single VP, or at two opposed VPs, or were they intended to be parallel?
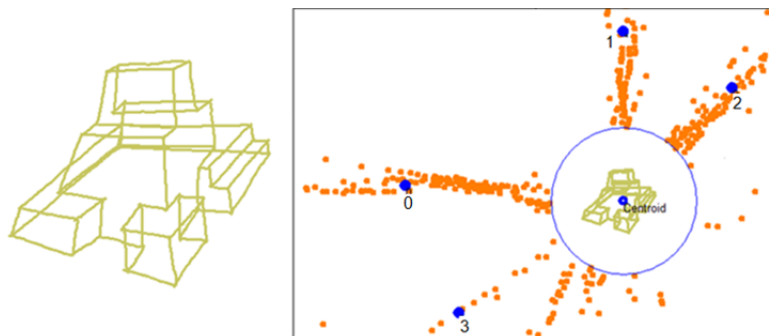

Fig. 12. Example of opposite VPs (#2 and #3)

We have already seen (in figure 4) that, even when convergence is intended, poor sketch quality can create artefacts: line orientations all close to, but not exactly, the intended angle α produce some candidate VPs located around α and others located around $\alpha+180°$ (figure 12). For clearly convergent lines this "opposite convergence" does not affect the results as the superfluous clusters are removed by the filtering stage of repeated edges. In this stage, we only merge genuinely parallel groups.

3.8 Resampling isolated edges

After running previous filters, some lines may remain ungrouped. To allow for this, where an isolated line is nearly parallel to a group of parallel lines, and does not intersect the group, we add the isolated line to the group. We implement this *resample of isolated parallel lines* as a threshold: the orientation of the isolated line differs by less than CD from the orientation of the group of parallel lines. In the *resample of isolated convergent lines*, any remaining isolated line is grouped into a nearby convergent group if (a) the line orientation differs by less than CD from the mean line of the group and (b) the line does not intersect any line of the group.

4 Analysis

Since engineering sketches are intended at exploring and communicating design ideas, our "ground-truth" is neither what can be measured with ruler and compass in the sketch, nor what the sketcher had in mind while sketching. What matters is what they actually communicate to humans. Hence, our goal is algorithmically interpreting sketches as humans do. Thus, we have tested our approach using the 18 examples in [4] (figure 13). 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.
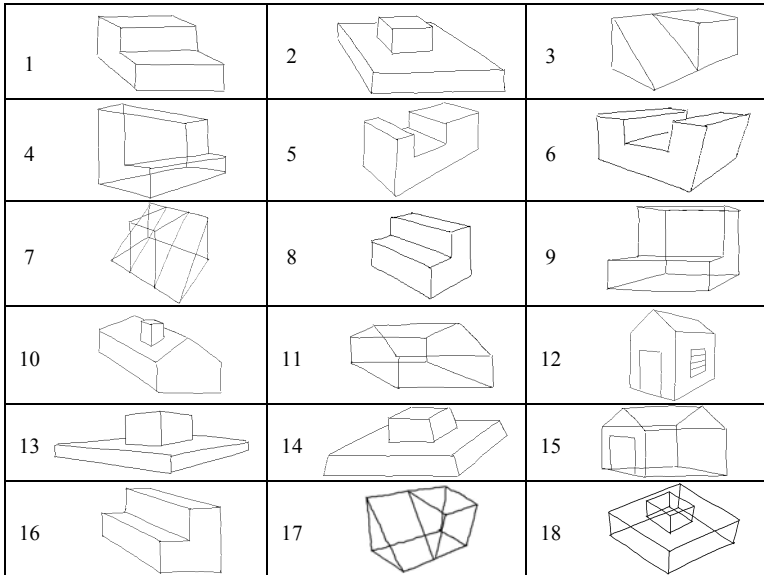
6

Fig. 13. Test examples

We have also tested two variants of an angle bracket, with respectively zero and three perceptual VPs (19 and 20 in Figure 14), and five more complex drawings (21 to 25 in Figure 14).
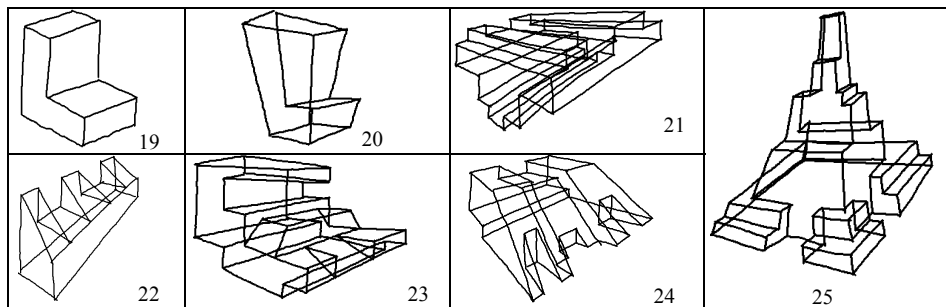

Fig. 14. Additional test drawings

The dataset tests demonstrate that our algorithm is limited neither by the number of VPs, nor by the number of lines in the sketches, and replicates human perception, as it locates VPs and groups lines representing sets of parallel edges in the same way humans did in the experiments described in [4]. For most test drawings, the results are the same as human perception. Quantitative results are available in the annex.

For drawing 6, very poor quality sketching prevents vectorisation from creating a valid line drawing (e.g. the short line highlighted in Figure 15), so our algorithm fails to group one edge (although even if the missing line were vectorised from junction to junction, it would be discarded by the anticonvergence filter). The same happens with drawings 9 and 13. In all three 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.
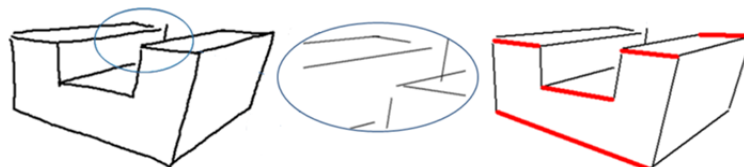

Fig. 15. Poor quality sketching results in incomplete grouping

4.1 Analysis of filtering stages

From our results, we can conclude that intersection and repeated edges filters are necessary to prevent false VPs. They usually work well, as they are built on sound criteria, but they still produce some false negatives. In general, it makes no difference wheth-

er the intersection filter is run before or after the repeated edges filter. In the few cases where it does make a difference (e.g. figure 16), it is always better to run the intersection filter first.
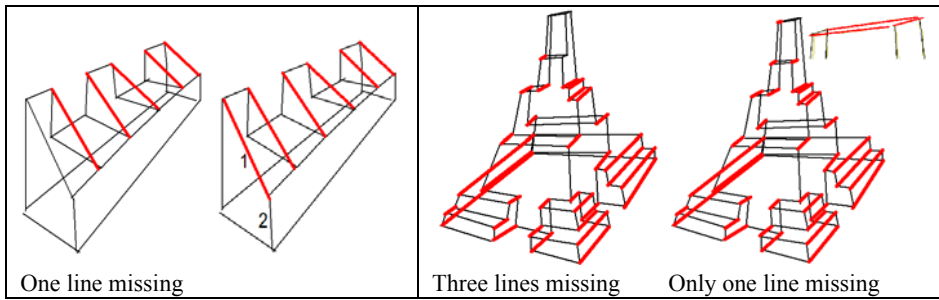

| One line missing | Three lines missing | Only one line missing |

Fig. 16. Two examples where the intersection filter must precede the repeated edges filter

The intersection filter fails in one case (Example 25) where a line is so short that a small absolute gap between endpoints becomes a large relative gap (greater than the 0.15 threshold).

In examples 7, 8, 9 and 16 opposite VPs appear. The filter for merging them into single groups does not detect them because they do not constitute a group. Instead, the complementary filter for isolated lines nearly-parallel to groups of parallel lines adds them to the group of nearly-parallel lines where they belong.

Occasional lines clearly non-parallel belong to parallel groups (example 10). They are there because they intersect at least one of the parallel lines outside the inner circle. A parallelism check removes them. In such cases, filtering repeated edges after filtering anticonvergent lines reduces the number of wrong lines.

4.2 Sensitivity analysis

Our experiments do not support the proposed outer radius of 5.0 as a rigid limit for filtering parallel groups. Some examples switch from parallel to convergent for outer radii in the range 7.0-10.0. Since this value was fixed though an experiment aimed only at finding how people perceive convergent lines, an experiment aimed at determining how people distinguish between convergent and parallel lines is still required. The dispersion threshold should be tuned too. However, the combination of $R_o= 5$ and $MinDisp= 15$ (section 3.5) usually allows the algorithm to determine which groups are parallel and which are convergent.

Example 13 illustrates two problems. Firstly, since the aspect ratio is so high, the mean lines to the two opposed vanishing points are almost parallel. The anticonvergence filter sometimes fails for lines which are both (a) poorly-sketched and (b) nearly collinear with mean lines. This problem is shared with lintel lines in examples 12 and 15. Secondly, the high aspect ratio makes intersection points more sensitive to small sketching errors, increasing the likelihood of internal intersections being accidentally located outside the inner ring and thus being wrongly listed as candidate VPs.

In order to determine how robust our algorithm is to the values of its tuning parameters, we carried out a sensitivity analysis. The algorithm uses the six parameters described previously: inner radius R_i and outer radius R_o (Section 3.1); Cutting-distance CD (Section 3.2); snapping distance SnapD and tolerance angle TolCol (Section 3.3); and minimum dispersion MinDisp (Section 3.5).

We varied all parameters in steps around their recommended values to find the ranges within which the parameters still work reasonably well (table 1), and determined how many more mistakes the algorithm makes. We conclude that for parameter R_i the algorithm is very sensitive, as it makes many more mistakes with different values (see examples 13, 15, 17, 23, 24 and 25 in the annex). It is also sensitive to parameter CD (see examples 13, 15 and 22 in the annex). The algorithm is robust to varying the other four parameters (R_o, MinDisp, SnapD and TolCol).

Table 1. Parameters of the algorithm and their recommended range of values

	Tested		Recommended						Tested	
	Min	Errors	Min	Errors	Default	Errors	Max	Errors	Max	Errors
Ri	0.4	16/25	1.0	13/25	1.5	5/25	2.0	11/25	12	25/25
Ro	1.0	0	4.0	0	5.0	0	7.0	2/25	62	25/25
CD	2°	25/25	8°	8/25	12°	3/25	20°	6/21	95°	25/25
MinDisp	1.0	4/25	7.0	2/25	10.0	1/25	15.0	5/25	77.0	25/25
SnapD	0.005	8/25	0.05	6/25	0.15	1/25	0.3	6/25	1.0	23/25
TolCol	1°	4/25	3°	2/25	5°	1/25	15°	5/25	180°	21/25

Very close VPs (those which correspond to a "fish eye perspective") may require R_i values smaller than 1 (see example 23 in annex), which fail for more usual perspectives.

Increasing SnapD beyond 0.3 raises the risk of detection of false intersections, particularly for dense sketches. TolCol also depends on the sketch quality and density: smaller values should only be used for quite accurate sketches; poor quality sketches would not be correctly processed by increasing TolCol beyond 15°, as this increases the number of false collinearities (e.g. lines 1 and 2 in figure 16).

4.3   Running time

We can compare our approach with a reduced set of approaches which are represented by the work by Tardiff [8]. They are computationally more expensive than our own approach, and were not tuned to mimic human perception.

Our algorithm is polynomial (it is a variant of agglomerative hierarchical clustering), and counting loops shows it to be $O(n^4)$ in theory. Its practical time complexity is illustrated in Figure 17 (where run time is in milliseconds): for typical engineering sketches (orange line), it is around $n^{1.97}$; for sketches with more lines (green line), the $O(n^4)$ clustering algorithm dominates and the time complexity is around $n^{3.64}$. Figure 20 uses the 25 examples illustrated in Figures 13 and 14, plus 13 other examples (not shown) with 134-247 lines.
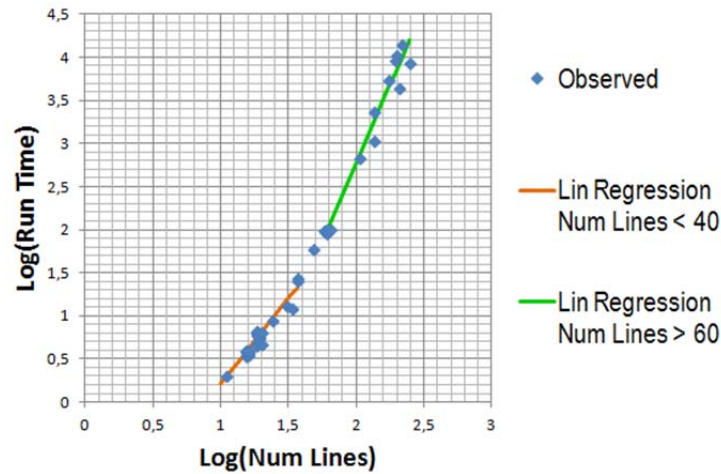

Fig. 17. Practical time complexity

We conclude that the algorithm is fast enough for an interactive approach for drawings of up to 100 lines, but may be impractical beyond that.

4.4   Other considerations

Our 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, except for very poor quality sketches and cases of opposite convergence. The algorithm succeeds irrespective of the number of actual vanishing points, and 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 currently used in SBM for grouping parallel lines. However, such approaches fail when the 2D lines corresponding to parallel edges span angles as big as 28°, as in Figure 18. Our new algorithm processes this and similar drawings correctly. We know of no previous approach which can do this.
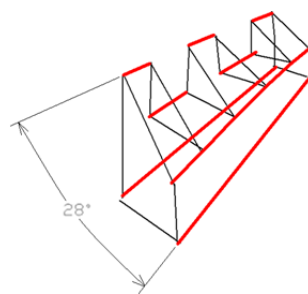

Fig. 18. Grouping parallel edges represented by converging lines

5   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. The new algorithm uses perceptual criteria for tuning thresholds based on the following conclusions from [4]: (1) humans generally perceive vanishing points for sets of lines spanning 12° or more; (2) humans agree about the orientation angle of the VP relative to the sketch, while they often do not agree about the distance of the VP from the sketch; (3) VPs

are easiest to perceive and to locate if they are neither too close nor too far away from the sketch—ideally, at distances not much more than the size of the sketch; (4) enforcement of the acceptance criteria should be tolerant to imperfections inside the main region (1.6x to 3x), and stricter outside (3x to 5x).

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.

During tests, we have frequently noted that failure to vectorise results in failure to detect vanishing points. Future developments should address those typical vectorisation faults which greatly affect the vanishing points algorithm. However, state-of-the-art vectorisation algorithms produce reasonably good input drawings which may be used by our algorithm to detect those vanishing points clearly perceived by humans, together with their associated groups of converging lines that depict parallel edges.

References

1. Plumed, R., Company, P., Piquer, A., 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), 241-250 (2010).
2. McLean, G.F., Kotturi, D.: Vanishing point detection by line clustering. IEEE Transactions on Pattern Analysis and Machine Intelligence, 17(11), 1090-1095 (1995).
3. Rother. C.: A new approach to vanishing point detection in architectural environments. Image and Vision Computing, 20(9-10), 647-655 (2002).
4. 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, 59-68 (2013).
5. Barnard, S.T.: Interpreting perspective images. Artificial Intelligence, 21(4), 435-462 (1983).
6. Magee, M.J., Aggarwal, J.K.: Determining vanishing points from perspective images, Computer Vision, Graphics and Image Processing 26(2), 256-267 (1984).
7. Varley, P.A.C.: Automatic Creation of Boundary-Representation Models from Single Line Drawings. Ph.D. thesis, Cardiff University (2003).
8. Tardif, J.P.: Non-Iterative Approach for Fast and Accurate Vanishing Point Detection. 12th International Conference on Computer Vision, IEEE, 1250-1257 (2009).
9. Almansa, A., Desolneux, A., Vamech, S.: Vanishing Point Detection without Any A Priori Information. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(4), 502-507. (2003).
10. Lin, L., Sherman, P.D.: Cleaning Data the Chauvenet Way. The Proceedings of the SouthEast SAS Users Group, SESUG Proceedings, Paper SA11 (2007).

Annex

A C++ implementation of our algorithm is available at www.regeo.uji.es, where the set of examples can also be found.

Groups of edges detected by the algorithm for the 18 examples in reference [1], plus the 7 examples added to further test it, are highlighted in the table (*parallel* groups are highlighted in thick *green* lines and *convergent* groups in thick *red* lines):

8			
9			
10			
11			
12	Dubious line close to the mean line of two opposite VP		One wrong line
13	Ri= 2.0, CD= 15°	One missing line	
14			
15	Ri= 5.0, CD= 6°		
16			
17			
18			
19			
20			
21	Two missing lines		Nine lines grouped separately
22	CD= 7°		
23	Ri= 0.8	One wrong line (nearly collinear)	Two anticonvergent lines missing / One missing line / One wrong line

	One missing line, (intersects with the wrong line)	One wrong line			
24	Ri= 2.0 TolCol= 9°				Two separate subgroups
25	Ri= 2.0, SnapD= 18% MinDisp= 5		Five lines grouped separately Three lines still missing	Five lines grouped separately One wrong line One missing line	