EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2004)
John F. Hughes and Joaquim A. Jorge (Editors)

Can Machines Interpret Line Drawings?

P. A. C. Varley,[1] R. R. Martin[2] and H. Suzuki[1]

[1]Department of Fine Digital Engineering, The University of Tokyo, Tokyo, Japan
[2]School of Computer Science, Cardiff University, Cardiff, Wales, UK

Abstract

Engineering design would be easier if a computer could interpret initial concept drawings. We outline an approach for automated interpretation of line drawings of polyhedra, and summarise what is already possible, what developments can be expected in the near future, and which areas remain problematic. We illustrate this with particular reference to our own system, RIBALD, summarising the published state of the art, and discussing recent unpublished improvements to RIBALD. In general, successful interpretation depends on two factors: the number of lines, and whether or not the drawing can be classified as a member of special shape class (e.g. an extrusion or normalon). The state-of-the-art achieves correct interpretation of extrusions of any size and most normalons of 20–30 lines, but drawings of only 10–20 lines can be problematic for unclassified objects. Despite successes, there are cases where the desired interpretation is obvious to a human but cannot be determined by currently-available algorithms. We give examples both of our successes and of typical cases where human skill cannot be replicated.

Categories and Subject Descriptors (according to ACM CCS): J.6 [Computer Aided Engineering]: Computer Aided Design

Keywords

Sketching, Line Drawing Interpretation, Engineering Design, Conceptual Design

1. Introduction

Can computers interpret line drawings of engineering objects? In principle, they cannot: any line drawing is the 2D representation of an infinite number of possible 3D objects. Fortunately, a counter-argument suggests that computers *should* be able to interpret line drawings. Human engineers use line drawings to communicate shape in the clear expectation that the recipient will interpret the drawing in the way the originator intended. It is believed [Lip98,Var03a] that human interpretation of line drawings is a skill which can be learned. If such skills could be translated into algorithms, computers could understand line drawings.

There are good reasons why we want computers to interpret line drawings. Studies such as Jenkins [Jen92] have shown that it is common practice for design engineers to sketch ideas on paper before entering them into a CAD package. Clearly, time and effort could be saved if a computer could interpret the engineer's initial concept drawings as solid models. Furthermore, if this conversion could be done within a second or two, it would give helpful feedback, further enhancing the designer's creativity [Gri97].

The key problem is to produce a model of the 3D object the engineer would regard as the most reasonable interpretation of the 2D drawing, and to do so quickly. While there are infinitely many objects which *could* result in drawings corresponding to e.g. Figures 1 and 2, in practice, an engineer would be in little doubt as to which was the correct interpretation. For this reason, the problem is as much heuristic as geometric: it is not merely to find a geometrically-realisable solid which corresponds to the drawing, but to find the one which corresponds to the engineer's expectations.

We suggest the following fully automatic approach, requiring no user intervention; our implementation verifies its utility for many drawings of polyhedral objects. (In a companion paper [VTMS04], we summarise an approach for interpreting certain drawings of *curved* objects with minimal user intervention.)

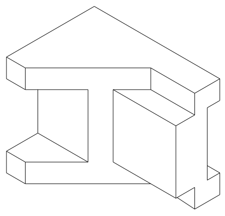
Figure 1: *Trihedral Drawing [Yan85]*

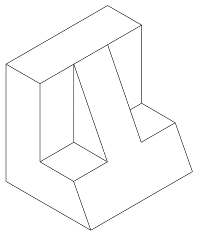
Figure 2: *Non-Trihedral Drawing [Yan85]*

- Convert the engineer's original freehand sketch to a line drawing. This is described in Section 3.
- Determine the *frontal geometry* of the object. The three most crucial aspects of this are:
  - Label the lines in the drawing as convex, concave, or occluding. See Section 4.
  - Determine which pairs of lines in the drawing are intended to be parallel in 3D. See Section 5.
  - Inflate the drawing to $2\frac{1}{2}$D by determining z-coordinates for each vertex. See Section 6.

  Performing these three tasks sequentially, in any order, presents difficulties, as each yields information useful to the others. Approaches which iterate these steps, or perform them in parallel, while still obtaining results in a reasonable time are a subject of current work—see later.
- Determine any symmetry elements (mostly mirror planes) in the object. This is not strictly necessary—the approach outlined in this paper in most cases works just as well without symmetry information. See Section 7.
- Classify the drawing (e.g. extrusion, normalon, general case). See Section 7.
- Complete the object topology by determining the topology of the hidden part of the object. See Section 8.
- Tidy or "beautify" the geometry of the completed object. Beautification also has applications in other fields, such as reverse engineering [VM02]—it is an area of active research in its own right. See Section 9.

2. Glossary

A solid model of a 3D *object* describes the *topology* and *geometry* of its *faces*, *edges* and *vertices*. *Topology* records connectivity between e.g. vertices and edges; *geometry* gives shape and positions e.g. the spatial coordinates of vertices.

A *natural line drawing* [Sug86] is a 2D drawing which represents the object as viewed from some viewpoint, and comprises *lines* (corresponding to visible or partially-visible edges) and *junctions* (where lines meet—most, but not all, junctions correspond to visible vertices of the object). Loops of lines and junctions form *regions*, which correspond to visible or partially-visible faces of the object. Note the careful distinction between 2D ideas (drawings, regions, lines, junctions) and 3D ideas (objects, faces, edges, vertices).

A *frontal geometry* is an intermediate stage between 2D drawing and 3D object (and thus is sometimes called "$2\frac{1}{2}$D"). In a frontal geometry, everything visible in the natural line drawing is given a position in 3D space, but the occluded part of the object, not visible from the chosen viewpoint, is not present.

A polyhedron is *trihedral* if three faces meet at each vertex. It is *extended trihedral* [PLVT98] if three planes meet at each vertex (there may be four or more faces if some are coplanar). It is *tetrahedral* if no more than four faces meet at any vertex. It is a *normalon* if all edges and face normals are aligned with one of three main perpendicular axes.

Junctions of different shapes are identified by letter: junctions where two lines meet are *L-junctions*, junctions of three lines may be *T-junctions*, *W-junctions* or *Y-junctions*, and junctions of four lines may be *K-junctions*, *M-junctions* or *X-junctions*. Vertex shapes follow a similar convention: for example, when all four edges of a *K-vertex* are visible, the drawing has four lines meeting at a K-junction.

When reconstructing an object from a drawing, we take the *correct* object to be the one which a human would decide to be the most plausible interpretation of the drawing.

3. Convert Sketch to Line Drawing

For drawings of polyhedral objects, we believe it to be most convenient for the designer to input straight lines directly, and our own prototype system, RIBALD, includes such an interface. However, it could be argued that freehand sketching is more "intuitive", corresponding to a familiar interface: pen and paper. Several systems exist which are capable of converting freehand sketches into natural line drawings—see e.g. [ZHH96], [Mit99], [SS01].

4. Which Lines are Convex/Concave?

Line labelling is the process of determining whether each line in the drawing represents a convex, a concave, or an occluding edge. For drawings of trihedral objects with no hole loops, the line labelling problem was essentially solved by Huffman [Huf71] and Clowes [Clo70], who elaborated the catalogue of valid trihedral junction labels. This turns line labelling into a discrete constraint satisfaction problem with 1-node constraints that *each junction must have a label in the catalogue* and 2-node constraints that *each line must have the same label throughout its length*. The Clowes-Huffman catalogue for L-, W- and Y-junctions is shown in Figure 3; $+$ indicates a convex edge, $-$ indicates a concave edge, and an arrow indicates an occluding edge with the occluding face on the right-hand side of the arrow. In trihedral objects, T-junctions (see Figure 4) are always occluding.

For trihedral objects, algorithms for Clowes-Huffman line labelling, e.g. those of Waltz [Wal72] and Kanatani [Kan90], although theoretically taking $O(2^n)$ time, are usually $O(n)$ in

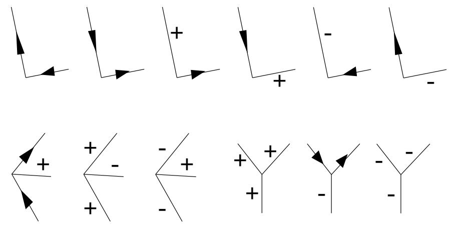

Figure 3: *Clowes-Huffman Catalogue*

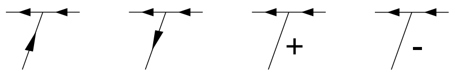

Figure 4: *Occluding T-Junctions*

practice [PT94]. It is believed that the time taken is more a function of the number of legal labellings than of the algorithm, and for trihedral objects there is often only a single legal labelling. For example, Figure 1 has only one valid labelling if the trihedral (Clowes-Huffman) catalogue is used.

Extending line labelling algorithms to non-trihedral normalons is fairly straightforward [PLVT98]. The additional legal junction labels are those shown in Figure 5. Note, however, that a new problem has been introduced: the new T-junctions are not occluding.

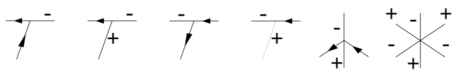

Figure 5: *Extended Trihedral Junctions*

Extension to the 4-hedral general case is less straightforward. The catalogue of 4-hedral junction labels is much larger [VM03]—for example, Figure 6 shows just the possibilities for W-junctions. Because the 4-hedral catalogue is no longer *sparse*, there are often *many* valid labellings for each drawing. Non-trihedral line labelling using the previously mentioned algorithms is now $O(2^n)$ in practice as well as in theory, and thus too slow. Furthermore, choosing the *best* labelling from the valid ones is not straightforward either, although there are heuristics which can help (see [VM03]).

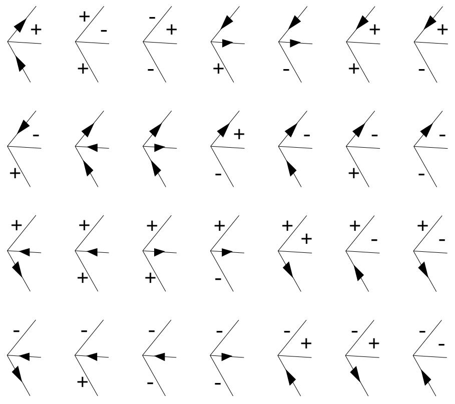

Figure 6: *4-Hedral W-Junctions*

Instead, an alternative labelling method is to use a relaxation algorithm. Label probabilities are maintained for each line and each junction; these probabilities are iteratively updated. If a probability falls to 0, that label is removed; if a probability reaches 1, that label is chosen and all other labels are removed. In practice, this method is much faster—labels which are possible but very unlikely are removed quickly by relaxation, whereas they are not removed at all by combinatorial algorithms. However, relaxation methods are less reliable (the heuristics developed for choosing between valid labellings when using combinatorial methods are reasonably effective). In test we performed on 535 line drawings [Var03b], combinatorial labelling labelled 428 entirely correctly, whereas relaxation labelling only labelled 388 entirely correctly.

The most serious problem with either approach is that in treating line labelling as a discrete constraint satisfaction problem, the *geometry* of the drawing is not taken into account, e.g. the two drawings in Figure 7 are labelled the same. The problems created by ignoring geometry become

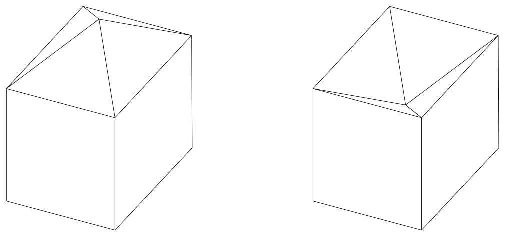

Figure 7: *Same Topology*

much worse in drawings with several non-trihedral junctions (see [VSM04]), and for these, other methods are required.

A new approach to labelling outlined in that paper and subsequently developed further [VMS04] makes use of an idea previously proposed for inflation [LB90]:

- Assign relative i, j, k coordinates to each junction by assuming that distances along the 2D axes in Figure 8 correspond to 3D distances along spatial i, j, k axes.
- Rotate the object from i, j, k to x, y, z space, where the latter correspond to the 2D x, y axes and z is perpendicular to the plane of the drawing.
- Find the 3D equation for each planar region using vertex x, y, z coordinates.
- For each line, determine from the equations of the two faces which meet the line whether it is convex, concave or occluding (if there is only one face, the line is occluding).

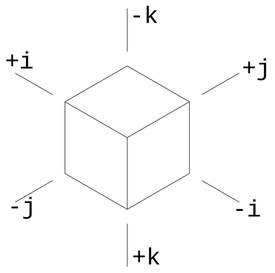

Figure 8: *Three Perpendicular Axes in 2D*

On its own, this method does not work well: e.g. it is difficult to specify a threshold distance d between two faces such that a distance greater than d corresponds to a step, and hence an occluding line, while if the distance is less than d the planes meet and the line is convex or concave. However, using the predictions made by this method as input to a relaxation labelling algorithm provides far better results than using arbitrary initialisation in the same algorithm.

This idea can be combined with many of the ideas in Section 6 when producing a provisional geometry. Various variants of the idea have been considered (see [VMS04]), particularly with reference to how the i, j, k axes are identified in a 2D drawing, without as yet any firm conclusions as to which is best overall. Another strength is that the idea uses the relaxation labeller to reject invalid labellings while collating predictions made by other approaches. This architecture allows additional approaches to labelling, such as Clowes-Huffman labelling for trihedral objects, to make a contribution in those cases where they are useful [VMS04].

Even so, the current state-of-the-art only labels approximately 90% of non-boundary edges correctly in a representative sample of drawings of engineering objects [VMS04].

Note that any approach which uses catalogue-based labelling can only label those drawings whose vertices are in a catalogue—it seems unlikely that 7-hedral and 8-hedral extended K-type vertices of the type found in Figure 9 will be

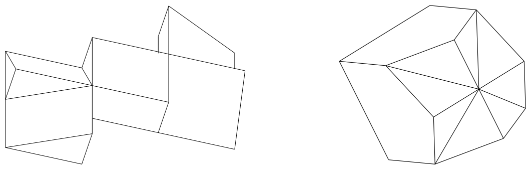

Figure 9: *Uncatalogued Vertices*

catalogued in the near future. In view of this, one must question whether line labelling is needed. Humans are skilled at interpreting line drawings, and introspection tells us that line labelling is not always a part of this process—it may even be that humans interpret the drawing first, and then (if necessary) determine which lines are convex, concave and occluding from the resulting mental model.

Our investigations indicate that line labelling *is* needed, at least at present. We are investigating interpreting line drawings without labelling, based on identifying aspects of drawings which humans are known or believed to see quickly, such as line parallelism [LS96], cubic corners [Per68] and major axis alignment [LB90]. Current results are disappointing. Better frontal geometry can be obtained if junction labels are available. More importantly, the frontal geometry is *topologically* unsatisfactory. Distinguishing occluding from non- occluding T-junctions without labelling information is unreliable, and as a result, determination of hidden topology (Section 8) is unlikely to be successful.

5. Which Lines are Parallel?

Determining which lines in a drawing are *intended* to be parallel in 3D is surprisingly difficult. It is, for example, obvious to a human which lines in the two drawings in Figure 10 are intended to be parallel and which are not, but determining this algorithmically presents problems.

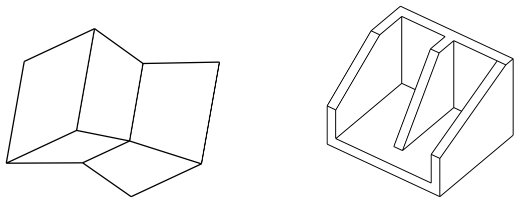

Figure 10: *Which Lines are Parallel?*

Sugihara [Sug86] attempted to define this problem away by using a strict definition of the *general position* rule: the user must choose a viewpoint such that if lines are parallel in 2D, the corresponding edges in 3D *must* be parallel. This makes no allowance for the small drawing errors which inevitably arise in a practical system.

Grimstead's "bucketing" approach [Gri97], grouping lines with similar orientations, works well for many drawings, but fails for both drawings in Figure 10. Our own "bundling" approach [Var03a], although somewhat more reliable, fares no better with these two drawings. The basic idea used in bundling is that edges are parallel *if* they look parallel *unless* it can be deduced from other information that they cannot be parallel. The latter is problematic for two reasons. Firstly, if 'other information' means labelling, identification of parallel lines must occur after labelling, limiting the system organisation for computing frontal geometry. Secondly, to cover increasingly rare exceptional cases, we must add extra, ever more complex, rules for deducing which lines may or may not be parallel. This is tedious and rapidly reaches the point of diminishing returns. For example, a rule which can deduce that the accidental coincidence in Figure 11 should not result in parallel lines would be both complicated to implement and of no use in many other cases.

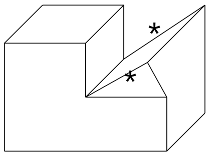

Figure 11: *Accidental Coincidence*

Furthermore, there are also cases where it is far from clear even to humans which edges should be parallel in 3D (edges *A*, *B*, *C* and *D* in Figure 12 are a case in point).

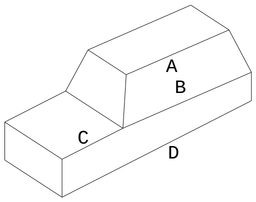

Figure 12: *Which Edges Should Be Parallel?*

In view of these problems, more recent approaches to frontal geometry (e.g. [VMS04]) simply ignore the possibility that some lines which appear parallel in 2D cannot in fact be parallel in 3D. Initially, it is assumed that they are parallel; this information is then re-checked after inflation.

6. Inflation to $2\frac{1}{2}$D

Inflation is the process of converting a flat 2D drawing into $2\frac{1}{2}$D by assigning z-coordinates (depth coordinates) to each vertex, producing a *frontal geometry*. The approach taken here is the simplest: we use *compliance functions* [LS96] to generate equations linear in vertex depth coordinates, and solve the resulting linear least squares problem. Many compliance functions can be translated into linear equations in z-coordinates. Of these, the most useful are:

Cubic Corners [Per68], sometimes called *corner orthogonality*, assumes that a W-junction or Y-junction corresponds to a vertex at which three orthogonal faces meet. See Figure 13: the linear equation relates depth coordinates z_V and z_A to angles F and G. Nakajima [Nak99] reports successful creation of frontal geometry solely by using a compliance function similar to corner orthogonality, albeit with a limited set of test drawings in which orthogonality predominates.

Line Parallelism uses two edges assumed to be parallel in 3D. The linear equation relates the four z-coordinates of the

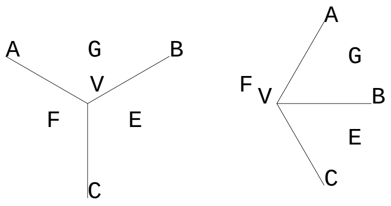

Figure 13: *Cubic Corners*

vertices at either end of the two edges. Line parallelism is not, by itself, inflationary: there is a trivial solution ($z = 0$ for all vertices).

Vertex Coplanarity uses four vertices assumed to be coplanar. The linear equation relating their z-coordinates is easily obtained from 2D geometry. Vertex coplanarity is also not, by itself, inflationary, having the trivial solution $z = 0$ for all vertices. General use of four-vertex coplanarity is not recommended (Lipson [Lip98] notes that if three vertices on a face are collinear, four-vertex coplanarity does not guarantee a planar face). However, it is invaluable for cases like those in Figure 14, to link vertices on inner and outer face loops: without it the linear system of depth equations would be disjoint, with infinitely many solutions.

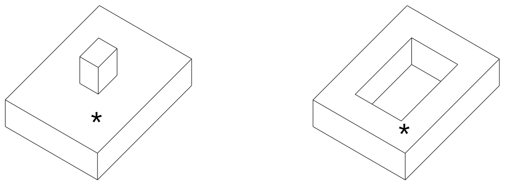

Figure 14: *Coplanar Vertices*

Lipson and Shpitalni [LS96] list the above and several other compliance functions; we have devised the following.

Junction-Label Pairs [Var03a] assumes that pairs of junctions with identified labels have the same depth implications they would have in the simplest possible drawing containing such a pair. An equation is generated relating the vertex depths at each end of the line based on the junction labels of those vertices. For example, see Figure 15: this pair of junction labels can be found in an isometric drawing of a cube, and the implication is that the Y-junction is nearer to the viewer than the W-junction, with the ratio of 2D line length to depth change being $\sqrt{2} : 1$.

Note that two of the most successful compliance functions, line parallelism and junction line pairs, require input information (parallel lines and line labels respectively) which, as we have seen, cannot always reliably be obtained. However, when this input information is both available and correct, inflation using the above compliance functions is the most reliable of the stages of processing described in this

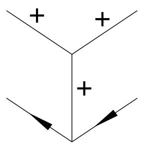

Figure 15: *Junction Label Pair*

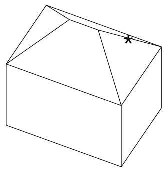

Figure 16: *Incorrect Inflation?*

paper. Although it occasionally fails to determine correctly which end of a line should be nearer the viewer, such failures arise in cases like the one in Figure 16 where a human would also have difficulty. The only systematic case where using a linear system of compliance functions fails is for Platonic and Archimedean solids, but a known special-case method (Marill's MSDA [Mar91]) is successful for these.

In order to make the frontal geometry process more robust when the input information (especially line labelling) is incorrect, we have experimented with two approaches to inflation which do without some or all of this information.

The first is the 'preliminary inflation' described in Section 4: find the i, j, k axes in the drawing, inflate the object in i, j, k space, then determine the transformation between i, j, k and x, y, z spaces. This requires parallel line information (to group lines along the i, j, k axes). Where such information is misleading, the quality of inflation is unreliable, but this is not always a problem, e.g. the left-hand drawing in Figure 10 is labelled correctly despite incorrect parallel line information. Once (i) a drawing has been labelled correctly and (ii) there is reason to suppose that the parallel line information is unreliable, better-established inflation methods can be used to refine the frontal geometry. However, the right-hand drawing is one of those which is *not* labelled correctly, precisely because of the misleading parallel line information.

A second promising approach, needing further work, attempts to emulate what is known or hypothesised about human perception of line drawings. It allocates merit figures to possible facts about the drawing; these, and the geometry which they imply, are iteratively refined using relaxation:

- *Face-vertex coplanarity* corresponds to the supposition that vertices lie in the plane of faces. We have already noted the difficulty of distinguishing occluding from non-occluding T-junctions; to do so, we must at some time decide which vertices *do* lie in the plane of a face.
- *Corner orthogonality*, which was described earlier. At least one inflationary compliance function is required, and this one has been found reliable. Although limited in principle, corner orthogonality is particularly useful in practice as cubic corners are common in engineering objects.
- *Major axis alignment* is the idea described above of using i, j, k axes. This is also an inflationary compliance function. It is newer than corner orthogonality, and for this reason considered less reliable. However, unlike corner orthogonality (which can fail entirely in some circumstances), major axis alignment does always inflate a drawing, if not always entirely correctly.
- *Line parallelism* is useful for producing 'tidy' output (e.g. to make lines terminating in occluding T-junctions parallel in 3D to other lines with similar 2D orientation). However, the main reason for its inclusion here is that it also produces belief values for pairs of lines being parallel as a secondary output, solving the problem in Section 5.
- *Through lines* correspond to the requirement that a continuous line intercepted by a T-junction or K-junction corresponds to a single continuous edge of the object.

A third, simpler, approach assumes that numerically correct geometry is not required at this early stage of processing, and identifying relative depths of neighbouring vertices is sufficient. Schweikardt and Gross's [SG00] work, although limited to objects which can be labelled using the Clowes-Huffman catalogue, and not extending well to non-normalons, suggests another possible way forward.

7. Classification and Symmetry

Ideally, one method should work for all drawings of polyhedral objects; identification of special cases should not be necessary. However, the state-of-the-art is well short of this ideal—in practice it is useful to identify certain frequent properties of drawings and objects. Identification of planes of mirror symmetry is particularly useful. Knowledge of such a symmetry can help both to construct hidden topology (Section 8) and to beautify the resulting geometry (Section 9). Identification of centres of rotational symmetry is less useful [Var03a], but similar methods could be applied.

The technique adopted is straightforward: for each possible bisector of each face, create a candidate plane of mirror symmetry, attempt to propagate the mirror symmetry across the entire visible part of the object, and assess the results using the criteria of (i) to what extent the propagation attempt succeeded, (ii) whether there is anything not visible which should be visible if the plane of mirror symmetry were a genuine property of the object, and (iii) how well the frontal geometry corresponds to the predicted mirror symmetry.

Classification of commonly-occurring types of objects (examples include extrusions, normalons, and trihedral objects) is also useful [Var03a], as will be seen in Section 8.

One useful combination of symmetry and classification is quite common in engineering practice (e.g. see Figures 1 and 2): a semi-normalon (where many, but not all, edges and face normals are aligned with the major axes) *also* having a dominant plane of mirror symmetry aligned with one of the object's major axes [Var03a]. The notable advantage of this classification is that during beautification (Section 9) it provides constraints on the non-axis-aligned edges and faces.

We recommend that symmetry detection and classification should be performed after creation of the frontal geometry. Detecting candidate symmetries without line labels is unreliable, and assessing candidate symmetries clearly benefits from the 3D information provided by inflation. The issue is less clear for classification. Some classifications (e.g. whether the object is a normalon) can be done directly from the drawing, without creating the frontal geometry first. However, others cannot, so for simplicity it is preferable to classify the object after creating its frontal geometry.

8. Determine Hidden Topology

Once the frontal geometry has been determined, the next stage of processing is to add the hidden topology. The method is essentially that presented in [VSMM00]: firstly, add extra edges to complete the wireframe, and then add faces to the wireframe to compete the object, as follows:

While the wireframe is incomplete:

- Project hypothesised edges from each incomplete vertex along the appropriate axes
- Eliminate any edges which would be visible at their points of origin
- Find locations where the remaining edges intersect, assigning merit figures according to how certain it is that edges intersect at this location (e.g. an edge intersecting only one other edge has a higher merit figure than an edge has potential intersections with two or more other edges)
- Reduce the merit for any locations which would be visible (these must be considered, as drawing errors are possible)
- Choose the location at which the merit is greatest
- Add a vertex at this location, and the hypothesised edges meeting at this location, to the known object topology

The process of completing the wireframe topology varies in difficulty according to the type of object drawn. We illustrate two special-case object classes, extrusions and normalons, and the general case. In some cases (e.g. if the object is symmetrical or includes a recognised feature), more than one vertex can be added in one iteration, as described in [Var03a]. Such cases increase both the speed and reliability of the process of completing the wireframe.

Completing the topology of extrusions from a known front end cap is straightforward. Figure 17 shows a drawing and the corresponding completed extrusion wireframe.

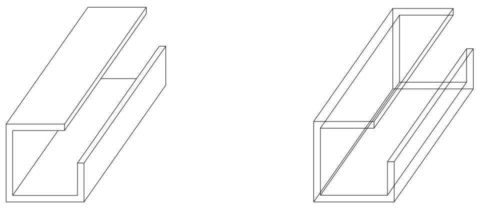
Figure 17: *Extrusion*

Knowing that the object is a normalon simplifies reconstruction of the wireframe, since when hypothesised edges are projected along axes, there is usually only one possibility from any particular incomplete vertex. Figure 18 shows a drawing of a normalon and the corresponding completed wireframe. Similarly, if the object is trihedral, there can be

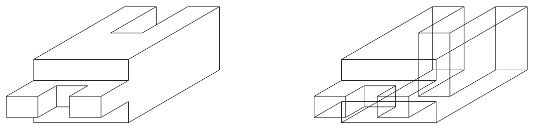
Figure 18: *Normalon [Yan85]*

at most one new edge from each incomplete vertex, simplifying reconstruction of the correct wireframe. Figure 19 shows a drawing of a trihedral object and the corresponding completed wireframe.

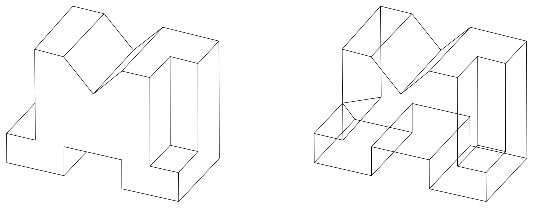
Figure 19: *Trihedral Object [Yan85]*

However, in the general case, where the object is neither a normalon nor trihedral, there is the significant difference that hypothesised edges may be projected in any direction parallel to an existing edge. Even after eliminating edges which would be visible, there may be several possibilities at any given incomplete vertex. The large number of possible options rapidly becomes confusing and it is easy to choose an incorrect crossing-point at an early stage. Although such errors can sometimes be rectified by backtracking, the more common result is a valid but unwanted wireframe. Only very simple drawings can be processed reliably. Figure 20 shows a general-case object and the corresponding completed wireframe; this represents the limit of the current state of the art.

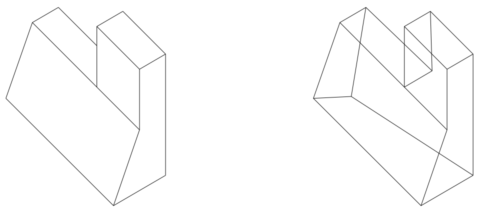
Figure 20: *General Case Object*

One particular problem, a specific consequence of the approach of completing the wireframe before faces, is that there is no assurance that the local environments of either end of a new edge match. It may happen that a sector around a new edge is solid at one end and empty at the other. This is perhaps the single most frequent cause of failure at present, and is especially problematic in that the resulting completed wireframe can *appear* correct. We aim to investigate faster and more reliable ways of determining the correct hidden topology of an object, starting with approaches aimed at correcting this conflicting local environment problem.

Adding additional faces to the completed wireframe topology for which the frontal geometry is already known is straightforward. We use repeated applications of Dijkstra's Algorithm [Dij59] to find the best loop of unallocated half-edges for each added face, where the merit for a loop of half-edges is based both on the number of half-edges required (the fewer, the better) and their geometry (the closer to coplanar, the better). We have not known this approach to fail when the input is a valid wireframe (which, as noted above, is not always the case).

9. Beautification of Solid Models

As can be seen from the Figures in the previous Section, even when topologically correct, the solid models produced may have (possibly large) geometric imperfections. They require 'beautification'. More formally, given a topologically-correct object and certain symmetry and regularity hypotheses, we wish to translate these hypotheses into constraints, and update the object geometry so that it maximises some merit function based on the quantity and quality of constraints enforced.

In order to make this problem more tractable, we decompose it into determination of face normals and determination of face distances from the origin; once faces are known, vertex coordinates may be determined by intersection. The rationale for this partitioning [KY01] is that changing face normals can destroy satisfied distance constraints, but changing face distances cannot destroy satisfied normal constraints.

However, there are theoretical doubts about this subdivision, related to the *resolvable representation* problem [Sug99] of finding a 'resolution sequence' in which information can be fixed while guaranteeing that no previous information is contradicted. For example, determining face equations first, and calculating vertex coordinates from them, is a satisfactory resolution sequence for many polyhedra, including all trihedral polyhedra. Similarly, fixing vertex coordinates and calculating face planes from them is a satisfactory resolution sequence for deltahedra and triangulated mesh models. Sugihara [Sug99] proved that:

- all genus 0 solids have resolution sequences (although if neither trihedral nor deltahedra, *finding* the resolution sequence might not be straightforward);
- (by counterexample) genus non-zero solids do not necessarily have resolution sequences.

Thus, there are *two* resolvable representation issues:

- finding a resolution sequence for those solids which have a non-trivial resolution sequence;
- producing a consistent geometry for those solids which do not have a resolution sequence.

Currently, neither problem has been solved satisfactorily.

Thus, although there are objects which have resolution sequences, but for which determining face normals, and then face distances, and finally vertex coordinates, is not a satisfactory resolution sequence, the frequency of occurrence of such objects has yet to be determined. If low, the pragmatic advantages of such an approach are perhaps more important than its theoretical inadequacy.

Our overall beautification algorithm is [Var03a]:

- Make initial estimates of face normals
- Use any object *classification* to restrict face normals
- Identify constraints on face normals
- Adjust face normals to match constraints
- Make initial estimates of face distances
- Identify constraints on face distances
- Adjust face distances to match constraints
- Obtain vertex locations by intersecting planes in threes
- Detect vertex/face failures and adjust faces to correct them

We use numerical methods for constraint processing, as this seems to be the approach which holds most promise. Alternatives, although unfashionable for various reasons, may become more viable as the state of the art develops: see Lipson et al [LKS03] for a discussion.

In addition to the resolvable representation problem, there is a further theoretical doubt about this approach. When attempting to satisfy additional constraints, it is necessary to know how many degrees of freedom are left in the system once previous, already-accepted, constraints are enforced. This apparently-simple problem appears to have no fully-reliable solution. One solution proposed by Li [LHS01] perturbs the variables slightly and detects which constraints have been violated. However, this is slow, and not necessarily theoretically sound either (e.g. a constraint relating face distances A and B may allow them to move together, but not independently of one another).

Two differing approaches have been tried to the problem of finding whether or not a geometry exists which satisfies a new constraint while continuing to satisfy previously-accepted constraints.

The first encodes constraint satisfaction as an error function (the lower the value, the better-satisfied the constraint), and face normals and/or face distances as variables, using a downhill optimiser to minimise the error function [CCG99,LMM02,Var03a]. Such algorithms use a 'greedy' approach, in which the constraint with the highest figure of merit is always accepted and enforced, and then for each other constraint, in descending order of merit: if the constraint is already satisfied numerically by the object, it is accepted; otherwise we attempt to adjust the variables numerically to accommodate the new constraint as well as all previous accepted constraints. If this succeeds, the new variable values are stored and the constraint is accepted; otherwise, the constraint is rejected and the previous variable settings are restored. The use of a downhill optimiser guarantees that progress is always downhill. However, progress can often be slow, and there is the possibility (unlikely in practice if the initial geometry is reasonable [Var03a]) that the downhill optimiser will be trapped in a local minimum.

To speed up the algorithm, a logical reasoning stage may be used, capable of directly accepting or rejecting constraints which duplicate or contradict previously-accepted constraints. On one hand [Var03a] reports difficulties with this approach, in view of (i) the unsolved resolvable representation and degrees-of-freedom problems, and (ii) the difficulty of reasoning whether constraints of different types duplicate or contradict one another. Such reasoning becomes harder as extra constraint types are added to a system: even simple 2D systems allowing only distance constraints require 30 inference rules [SAK90]; the number of inference rules required for 3D systems allowing both angular and distance constraints is certainly much greater. On the other hand, Langbein et al [LMM04] report considerable success with a logical reasoning approach used in a reverse engineering system. Logical reasoning to accept or reject constraints is clearly faster than an entirely numerical approach, and represents the current state of the art.

An alternative which may warrant further study is to replace the downhill optimisation step above by a faster approach which iteratively updates the variables (face normals and/or distances) *geometrically*, using their current values and geometric inferences based on the constraint set under consideration to predict next-iteration values. While it is harder to guarantee that such updates are converging towards a solution, by making direct use of geometric information they have the potential to be faster.

There has been recent progress in solving systems of geometric constraints using Cayley-Menger determinants (e.g. [PRTT03]), but such work has concentrated on problems where all constraints can be represented as distance constraints. It is not clear that it can be extended to all constraints relevant to sketching, e.g. “macro”-constraints such as enforcement of a plane of mirror symmetry.

Apart from numerical methods, several other possible techniques may be relevant, too many to list here; two with unexplored potential are described. Firstly, although the simple constructivist approach of Suzuki et al [SAK90] cannot be recommended—the chains of reasoning required form so-called ‘loops’—more recent graph-based methods such as [FH97] may make it possible to disentangle these loops. Auxiliary construction methods such as those of Lipson et al [LKS99] and Benko et al [BKV*02] have the merit that seem to correspond to the way a human would approach the same problem.

10. Summary

So, can computers interpret line drawings? Yes, to some extent, but they are nowhere near as skilful as human engineers. On the one hand, there is a whole category of line drawings (extrusions) which computers can interpret flawlessly. On the other hand, there is another category (those with ‘extended K’ vertices) which, as yet, cannot be interpreted at all. In between, the proportion of objects which can be interpreted grows steadily as existing methods are refined and extended.

However, if there is to be a breakthrough, rather than incremental improvement, we must recall our aims: to duplicate the ability of humans to interpret line drawings. By and large, those aspects of this skill which have been identified have already been translated into algorithms. The fact that these algorithms are not enough makes it clear that there are more aspects of this skill still waiting to be identified.

11. Acknowledgements

The support of the Japan Society for the Promotion of Science Fellowship P03717, providing the funding for our own research described here, is gratefully acknowledged.

References

[BKV*02] P. Benko, G. Kos, T. Varady, L. Andor and R. R. Martin, *Constrained Fitting in Reverse Engineering*, Computer Aided Geometric Design, **19**, 173–205, 2002.

[Clo70] M.B. Clowes, *On Seeing Things*, Artificial Intelligence, **2**, 79–116, 1970.

[Dij59] E.W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik **I**, 269–271, 1959.

[FH97] I. Fudos and C.M. Hoffmann, *A Graph-Constructive Approach to Solving Systems of Geometric Constraints*, ACM Transactions on Graphics, **16**(2):179–216, 1997.

[GCG99] J.X.Ge, S.C.Chou and X.S.Gao, *Geometric Constraint Satisfaction using Optimization Methods*, Computer-Aided Design **31**(14), 867–879, 1999.

[Gri97] I.J.Grimstead, *Interactive Sketch Input of Boundary Representation Solid Models*, PhD Thesis, Cardiff University, 1997.

[Huf71] D.A.Huffman, *Impossible Objects as Nonsense Sentences*, Machine Intelligence **6**, 295–323, New York American Elsevier, 1971.

[Jen92] D.L.Jenkins, *The Automatic Interpretation of Two-Dimensional Freehand Sketches*, PhD Thesis, University of Wales College of Cardiff, 1992.

[Kan90] K.Kanatani, *Group-Theoretical Methods in Image Understanding,* Number 20 in Springer Series in Information Sciences, Springer-Verlag, 1990.

[KY01] A.V. Kumar and L. Yu, Sequential Constraint Imposition for Dimension-Driven Solid Models, Computer-Aided Design **33**, 475–486, 2001.

[LB90] D.Lamb and A. Bandopadhay, *Interpreting a 3D Object From a Rough 2D Line Drawing,* In ed. A.E.Kaufman, Proceedings of the First IEEE Conference on Visualization '90, 59-66, IEEE, 1990.

[LMM02] F.G. Langbein, A.D. Marshall and R.R. Martin, *Numerical Methods for Beautification of Reverse Engineered Geometric Models,* Proc. GMP 2002.

[LMM04] F.G. Langbein, A.D. Marshall and R.R. Martin, *Choosing Consistent Constraints for Beautification of Reverse Engineered Geometric Models,* Computer Aided Design, **36**(3), 261–278, 2004.

[LHS01] Y.T. Li, S.M. Hu and J.G. Sun, *On the Numerical Redundancies of Geometric Constraint Systems,* in ed. H. Suzuki, A. Rockwood and L.P. Kobbelt, Ninth Pacific Conference on Computer Graphics and Applications (PG'01), 118–123, IEEE Comp Soc Press, 2001.

[Lip98] H. Lipson, *Computer Aided 3D Sketching for Conceptual Design,* PhD Thesis, Technion-Israel Institute for Technology, Haifa, 1998.

[LKS99] H. Lipson, F. Kimura and M. Shpitalni, *Solving Geometric Constraints by Auxiliary Constructions,* `http://www.mit.edu/~hlipson/papers/auxcon.htm`, 1999.

[LS96] H. Lipson and M. Shpitalni, *Optimization-based Reconstruction of a 3D Object from a Single Freehand Line Drawing,* Computer-Aided Design **28**(8), 651-663, 1996.

[Mar91] T. Marill, *Emulating the Human Interpretation of Line-Drawings as Three-Dimensional Objects,* International Journal of Computer Vision **6**(2) 147–161, 1991.

[Mit99] J. Mitani, *A Study of 3D Sketching used for Aiding Engineering Product Design*, MSc Thesis, The University of Tokyo, 2000. In Japanese.

[Nak99] C. Nakajima, *An Inference Method of Three-dimensional Shape by Depth Perception from One Image*, IPSJ Journal **40**(8) (Special Issue on Meeting on Image Recognition and Understanding), 3179–3187, 1999. In Japanese.

[PLVT98] P. Parodi, R.Lancewicki, A. Vijh and J.K.Tsotsos, *Empirically-Derived Estimates of the Complexity of Labeling Line Drawings of Polyhedral Scenes,* Artificial Intelligence **105**, 47–75, 1998.

[PT94] P. Parodi and V.Torre, *On the Complexity of Labeling Perspective Projections of Polyhedral Scenes,* Artificial Intelligence **70**, 239–276, 1994.

[Per68] D.N. Perkins, *Cubic Corners,* Quarterly Progress Report 89, 207–214, MIT Research Laboratory of Electronics, 1968.

[PRTT03] J.M. Porta, L. Ros, F. Thomas and C. Torras, *A Branch-and-Prune Algorithm for Solving Systems of Distance Constraints,* Proceedings of the 2003 IEEE International Conference on Robotics and Automation, 2003.

[SG00] E. Schweikardt and M. D. Gross, *Digital Clay: Deriving Digital Models from Freehand Sketches,* Automation in Construction **9**, 107–115, 2000.

[SS01] T. M. Sezgin and T. Stahovich, *Sketch Based Interfaces: Early Processing for Sketch Understanding,* 2001.

[Sug86] K. Sugihara, *Machine Interpretation of Line Drawings,* MIT Press, 1986.

[Sug99] K. Sugihara, *Resolvable Representations of Polyhedra*, Discrete and Computational Geometry **21**(2), 243–255, 1999.

[SAK90] H. Suzuki, H. Ando and F. Kimura, *Geometric Constraints and Reasoning for Geometrical CAD Systems*, Computing and Graphics **14**(2), 211-224, 1990.

[VM02] T. Varady and R. R. Martin, *Reverse Engineering,* in: The Handbook of Computer Aided Design, eds. G. Farin, J. Hoschek, M.-S. Kim, 651–681, Elsevier, 2002.

[VSMM00] P.A.C. Varley, H. Suzuki, J. Mitani and R.R. Martin, *Interpretation of Single Sketch Input for Mesh and Solid Models,* International Journal of Shape Modelling **6**(2), 207–241, 2000.

[Var03a] P.A.C. Varley, *Automatic Creation of Boundary-Representation Models from Single Line Drawings,* PhD Thesis, Cardiff University, 2003.

[Var03b] P.A.C. Varley, Sketches of Polyhedral Solids. `http://ralph.cs.cf.ac.uk/Data/Sketch.html`, 2003.

[VM03] P.A.C. Varley and R.R. Martin, *Deterministic and Probabilistic Approaches to Labelling Line Drawings of Engineering Objects,* International Journal of Shape Modelling **9**(1), 79–99, 2003.

[VSM04] P.A.C. Varley, H. Suzuki and R.R. Martin, *Interpreting Line Drawings of Objects with K-Junctions*, Proc. Geometric Modeling and Processing 2004, Eds. S.-M. Hu, H. Pottmann, 249–358, 2004.

[VMS04] P.A.C. Varley, R.R. Martin and H. Suzuki, *Making the Most of Using Depth Reasoning to Label Line Drawings of Engineering Objects*, in ed. G. Elber, N. Patrikalakis and P. Brunet, 9th ACM Symposium on Solid Modeling and Applications SM'04, 191–202, 2004.

[VTMS04] P.A.C. Varley, Y. Takahashi, J. Mitani and H. Suzuki, *A Two-Stage Approach for Interpreting Line Drawings of Curved Objects*, EuroGraphics Workshop on Sketch-Based Input SBM'04, 2004.

[Wal72] D.M.Waltz, *Generating Semantic Descriptions from Drawings of Scenes with Shadows,* Tech Rept AI-TR-271, M.I.T., Cambridge USA, 1972.

[Yan85] H.W. Yankee, *Engineering Graphics,* Prindle, Weber and Schmidt, 1985.

[ZHH96] R.C. Zeleznik, K.P. Herndon and J.F. Hughes, *SKETCH: An Interface for Sketching 3D Scenes,* Proceedings of SIGGRAPH '96. 163–70, ACM Press, 1996.