

Incremental Line Labelling for Sketch Input of Solid Models

I. J. Grimstead and R. R. Martin

University of Wales College of Cardiff, P.O. Box 916, Cardiff CF2 4YN, U.K.
e-mail: {I.J.Grimstead,ralph}@cs.cf.ac.uk

Abstract

Designers need to transfer three-dimensional objects from their minds via a two-dimensional interface into a solid modelling system. We propose a system where objects are sketched interactively in two-dimensions and incrementally line-labelled as the drawing progresses, as the preliminary stage in constructing a solid model. Once the sketch is complete, the line-labels are coupled with various inferred constraints to enable us to generate a solid model. In this paper we describe and evaluate a modification to existing line labelling methods to allow them to work incrementally. In this way, the labelling and region information can be built up as the user sketches, rather than all at once at the end of the sketching process, which reduces the delay before the final solid model is built.

Keywords: Sketch input; line labelling; reconstruction; solid modelling

1. Introduction

In computer aided design there is a need to transfer three-dimensional objects from the minds of designers into solid modelling systems. Various researchers (Goldschmidt[?], Jenkins[?], Ullman[?]) have shown that current systems are slow and unwieldy, frustrating designers and interrupting the cognitive process; what is needed is a quick, natural method to input solid models into a CAD system, using sketch input.

Current research by the CAD community into sketch input either constrains the user to sketch relative to a fixed set of axes (Hale[?], Herot[?]), or still requires input additional to the sketch (Bier[?], Fukui[?]). These methods are not general enough to allow a user to pick up a pen and sketch a solid object on a digitising tablet, with the result being the correct interpretation of the drawing; some constraints are always placed on the user, who must be trained in using the system before being able to draw with it.

To produce a different approach, we have examined work in the vision community that interprets line drawings of images; these systems do not constrain the drawing with menu input or fixed axes,

as they do not rely on any additional information to the image itself. The techniques considered included matching objects against a dictionary of known parts (Jacot-Descombes[?], Munck-Fairwood[?]), using energy functions to fit an object to the view presented (Kriegman[?], Leclerc[?]) and the usage of gradient space to constrain face orientation (Shaffer[?], Ulupinar[?]).

Such techniques from the vision community often use line-labelling as an initial method of deriving information about the relative orientation of faces in a line drawing; starting from this, more information is inferred to constrain interpretations of the drawing. We also use line labelling as the first stage of a system for sketch input of solid models; the overall system is presented in Grimstead[?].

Vision systems are always presented with a snapshot. However, during interactive sketching the input is slowly built-up and is not necessarily complete at any given moment. Thus, we wish to develop an incremental line labelling technique that allows line labelling to proceed simultaneously with sketch input. To optimise use of computer processing time, we wish to build up the labelling as the user proceeds, rather

than processing all lines once the sketch is complete. In this way, it is intended that the delay after sketching is complete, and before the final solid model is reconstructed, will be reduced, improving responsiveness of the system to the user. Note that we also produce some face information as part of our incremental method. We discuss previous work on line labelling in the next section.

2. Overview of Line Labelling

In this section we briefly explain line labelling, and the information derived from it.

Line labelling is a method for interpreting the three-dimensional structure of a two-dimensional line drawing, resulting in labels being placed on lines to indicate their relative position in space. There are four labels used, indicating that a line is convex or concave with respect to the viewer, or occluding something to the left or right of the line. Only certain combinations of labellings are possible at line junctions; the overall set of line labels for a drawing is reduced by rejecting incompatible labelling combinations. Details of the method used can be found in any standard textbook on artificial intelligence (Winston[?], for example).

The initial work in this area covered polyhedra and was carried out by Clowes[?] and Huffman[?], with later work extending the ideas to curved surfaces, *e.g.* Malik[?]. A complete catalogue of possible labellings at each junction type is given in Huffman[?].

We assume in the initial version of our system, and in this paper, that our objects are opaque, trihedral polyhedra in general position. In other words, no hidden lines are drawn, three faces must meet at every vertex, all faces are planar and they are not drawn in a special position (*e.g.* no faces contain the viewing direction). Although we restrict ourselves thus in this paper, note that line labelling can be used with vertices more complex than trihedral (see Malik[?]), curved surfaces (see Malik[?]), transparent solids (see Sankar[?]), and drawings with shadows (see Waltz[?]). There is thus no reason to suppose that generalising the work here will not be possible. Note, however, that all these previous line labelling schemes assume that the drawing is complete before they start processing; our contribution is to present an incremental method which works with incomplete sketches.

The Huffman-Clowes line labelling scheme provides immediate information about a scene; it gives order relations between faces and helps isolate individual objects. Sugihara[?] uses the labels to infer further information about the spatial structure that constrains the faces sufficiently to reconstruct the three-dimensional solid; we use this information in a similar way in our

complete system (Grimstead[?]), although we use somewhat different techniques, and methods are also used to reconstruct the hidden part of the object. Dual space techniques, such as those used by Kanade[?], can also be used to directly produce the three-dimensional shape from the constraints represented by line labels.

3. Extra Line Labels for Incremental Line Labelling

We use the Huffman-Clowes labelling scheme as the basis of our incremental labelling algorithm; however, our system works interactively and therefore receives lines as the user sketches. In this paper we use the standard conventions for line labels and junction types used in Huffman[?]. An incremental labelling scheme differs from a traditional labeller in that it has to work with incomplete drawings; this introduces extra possibilities in the junction catalogues. These extra labellings will be derived from incomplete **L** junctions in Section 3.1 and from junctions appearing temporarily on the silhouette in Section 3.2. These labellings are then used in the algorithm presented in Section 4.

3.1. Additional Labellings Formed from Partially Completed Junctions

With a drawing being labelled incrementally, junctions are not stable; they may have extra lines added to them later. If we ignore any lines that do not form part of a closed loop that encloses an area of the drawing, we can be sure that all lines form part of a boundary of a two-dimensional region, and thus all junctions involving such lines are dihedral or trihedral (*i.e.* two or three visible lines are connected to each junction, respectively). As we limit ourselves to trihedral polyhedra, we have the following pattern of events: initially, a junction is created when a new line is drawn by the user; the junction has a single incident line, in which case it is unlabellable (as it does not yet form part of a completed region) and is marked as such. The next thing that can happen to this junction is the user draws another line that is incident to it, so the junction is marked as an **L** junction, which is labellable. The problem arises now of whether the user intends this to remain as an **L** junction, or will later attach another line to form a **Y**, **T** or **Arrow** junction; as there is no way of knowing if an extra line will be attached until it happens, we must allow for this possibility. There now follows a derivation of the extra labellings for an **L** junction which must be considered temporarily valid at least, given that an **L** junction may later be transformed into a **Y**, **T** or **Arrow** junction.

In Figure ??, the arrows indicate those **L** junction

labellings which can be converted into each valid **Y** junction labelling. For example, examine case (i): an **L** junction may be transformed into a **Y** junction that consists solely of convex edges, in which case there must exist an **L** junction consisting solely of convex edges, to which an additional convex edge may be added to produce the **Y** junction. These extra labellings now enable a (temporarily) **L** junction to be converted to a valid **Y** junction at a later time.

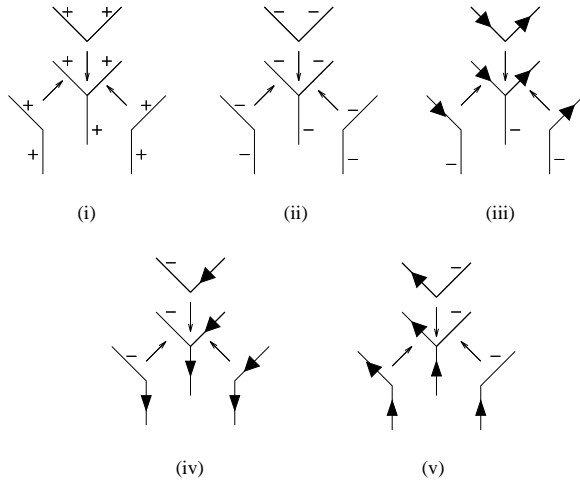


Figure 1: Additional labellings to transform an **L** junction into a **Y** junction

The additional **L** junction labellings required to transform an **L** junction into an **Arrow** or a **T** junction are derived in a similar manner, and their derivations are not presented here. The complete catalogue of additional **L** junction labellings required to allow **L** junctions to be transformed into any trihedral junction type is given in Figure ??.

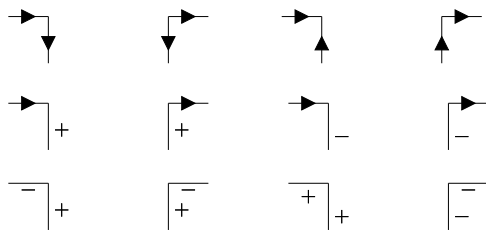


Figure 2: Complete catalogue of additional **L** labellings

3.2. Additional Labellings Formed from Temporarily Silhouette Junctions

This section considers what additional labellings must be temporarily permitted to enable junctions to lie

initially on the silhouette, when these junctions will finally lie in the interior of the drawing, when more of the object has been drawn around them. In the standard line labelling scheme, the silhouette of the drawing is initially labelled with occluding lines as the object occludes the background. These are the only line labels initially assumed, and are used as a starting point for a constraint propagation algorithm. With the incremental scheme, we can still label the final silhouette as occluding the background; however, while a drawing is incomplete we cannot yet do this. For example, see Figure ??(i); the region information around the highlighted junction is currently incomplete (as only two faces meet at the vertex) and hence does not have a valid labelling in the labelling scheme for complete objects. However, in Figure ??(ii), the highlighted junction now forms part of the interior of the drawing and is now surrounded by three faces and hence is labellable. Thus, a problem arises necessitating temporary permission of a normally invalid junction labelling for a junction which will finally be associated with the interior of an object, permitting it to exist on the silhouette of the incomplete drawing. In this section we derive the additional labellings required to let such junctions exist temporarily on the silhouette. Initially, we will derive the labellings to permit a **Y** junction to exist on the silhouette.

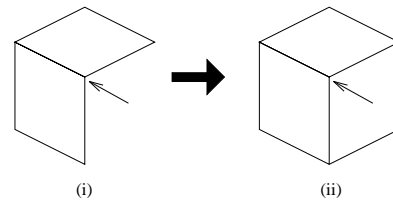


Figure 3: An internal junction of the cube temporarily on the silhouette

These labellings correspond to a **Y** junction on the silhouette that is currently connected to two faces, rather than the three required for the trihedral assumption. Incomplete **Y** junctions can only occur on the (temporary) silhouette, as if they are not on the silhouette, they must lie on the inside of the drawing and must be surrounded by internal regions corresponding to three faces.

When a **Y** junction occurs on the silhouette it must be attached to two regions (any non-region-separating lines are omitted from consideration); the third region at the junction is the background, which is taken to be disconnected from the object. With the **Y** junction on the silhouette, the non-silhouette line must lie on the right hand side of the arrows on the two silhouette lines, as the non-silhouette line must lie on the inside of the silhouette. Figure ?? presents an exhaustive set

of examples showing how the possible silhouette **Y** junctions can be converted into the permitted trihedral **Y** junctions.

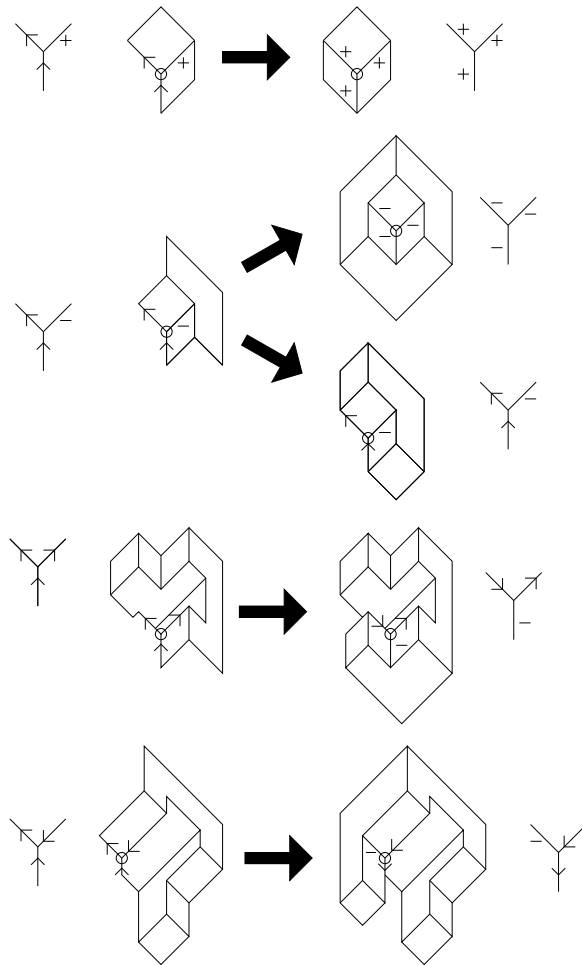


Figure 4: **Y** junction labellings on a silhouette

The left column of the diagram shows an incomplete drawing with the **Y** junction of interest highlighted, preceded by the additional labelling required to permit correct labelling of it. The next two columns show a completed drawing, with the junction again highlighted, followed by the final labelling.

Similar methods can be applied to find additional permissible labellings for **Arrow** and **T** junctions which are temporarily on the silhouette. These are shown in Figure ?? together with the additional **Y** junction labellings.

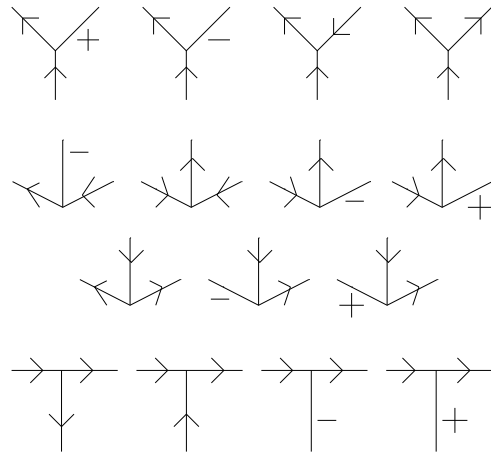


Figure 5: **Y**, **Arrow** and **T** junction labellings on a silhouette

4. Algorithm for Incremental Line Labelling

In this section we present an overview of the incremental line labelling algorithm. Each major module will then be discussed in detail in a separate sub-section.

4.1. Overview

There are four main stages in the incremental labelling method, these being line input and tidying, region creation, drawing update and line labelling, as depicted in Figure ??.

As the user sketches lines, they are placed into a buffer, and a secondary background task then extracts lines from the buffer and processes them through the other stages of the method. With this approach, the user can continue to sketch without interruption; on the other hand, input of lines is an I/O bound process, so much of the CPU power is available for the other stages. When the user has finished sketching, he clicks a button setting a done flag. The overall algorithm is summarised below:

I/O task

```
pool_of_lines := empty
relabel_from_start := false
stop := false
```

repeat

```
  if user adds a line then add it to the pool
```

```
  if user deletes a line then
```

```
    if line is in pool then
```

```
      remove it
```

```
    else
```

```
      if line is the last line drawn then rollback
```

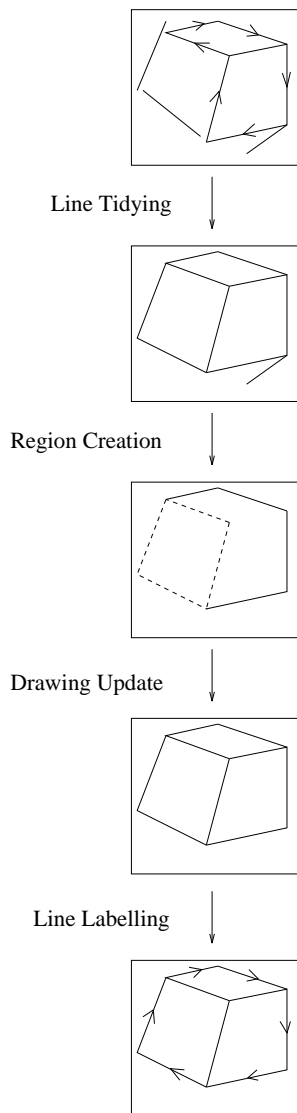


Figure 6: Block diagram of incremental line labelling

```

// this is a common case, so worth special casing
else set relabel_from_start flag
endif
endif

if user is finished then set stop flag
until stop

Processing task

lineList, leftRegion, rightRegion : list of lines
newRegionCreated : boolean
currentDrawing : holds the currently completed
sections of the drawing in terms of lines,

```

```

junctions and regions (excludes all non-region
connecting lines)

```

```

while (not stop)

```

```

if relabel_from_start then
remove required line from currentDrawing
recreate regions affected by line removal
IncrementallyLabelDrawing(all lines in
currentDrawing, currentDrawing)

```

```

else if pool_of_lines is not empty then

```

```

newRegionCreated :=
RegionCreation(currentDrawing,
pool_of_lines, leftRegion, rightRegion)

```

```

if newRegionCreated then
lineList := DrawingUpdate(leftRegion,
rightRegion, currentDrawing)
IncrementallyLabelDrawing(lineList,
currentDrawing)
endif
endif

```

```

endif
endwhile

```

```

remove incremental labellings
produce final labelling

```

Whenever a new region (a closed area of the drawing bounded by lines) is created, the drawing update and incremental labelling procedures are called, which respectively updates region information, removing appropriate lines from the pool, and uses the new information to further constrain the line labelling. This results in the region information and line labelling being produced incrementally.

4.2. Line Input and Tidying

Line input occurs as the user sketches lines on a digitising tablet. Tidied lines are input to the region creation stage: lines drawn by the user are straightened and connected (where applicable) to any previous junctions in the drawing.

4.3. Region Creation

As new lines are read from the input buffer we check if a new region of the drawing has been created; this is detected by the last line being connected at both ends. Region creation takes a series of tidied lines as input and produces completed regions in the drawing. When a new closed region has been produced in the drawing, the two regions on either side of the last new line are passed on to the drawing update module in the form of two new line loops. Note that one of the line loops may represent the silhouette of the drawing, with the background region on one side of it.

We check if the last line drawn by the user has produced a new closed region in the drawing, by finding the drawing regions on either side of it. This is accomplished by following a series of lines connected to the last line drawn, at each stage taking the successive line which corresponds to a left turn from the current line, assigning a marking number (unique to each execution of this procedure) to the left-hand side of the line, until we arrive back where we started, or we pass over a line which has already been assigned the current marking number. This forms a region on the left-hand side of the last line drawn.

If we meet the left-hand side of a line which has already been assigned the current marking number, then we are meeting lines that have already been processed—we are caught in a loop of lines that will not return us to the last line drawn, so the last line has not produced a new region of the drawing. Also, if the marker number is assigned to both sides of the last line, then it does not form a new region.

Otherwise, if the last line does form a new left-hand region, we use a similar method (with a different marking number) to form the region on the right-hand side of the last line drawn, marking the right-hand side and taking right turns instead.

If a right-hand region has been created, then the last line drawn has created a new region in the drawing, so we remove any non-region-separating lines (and any subsequently connected lines) from the two regions, where such a line is defined to be line that does not yet form part of a region boundary, and is detected by having the same marker number on both sides. Examples of non-region separating lines and their corresponding removal are presented in Figure ??, where the numbers adjacent to lines represent the marker number placed on each side of the lines. The dashed line is the last line added, with the arrows indicating the non-region separating lines detected.

```
function RegionCreation(currentDrawing : complete
  sections of current drawing, var pool_of_lines :
  pool of new lines not yet in currentDrawing,
  var leftRegion, rightRegion : list of lines)
  : boolean
```

```
  set marker to a new, unused value
```

```
  leftRegion := empty list
  rightRegion := empty list
```

```
  currentLine := last line drawn in pool_of_lines
```

```
  repeat
    assign 'marker' to left hand side of currentLine
```

```
    at the end of the currentLine, select the
      connected line that forms the tightest
```

```
    left-hand turn (if there are no other
    connected lines at this end, the tightest
    left-hand turn results in going back along
    the same line, in the opposite direction)
```

```
  append currentLine at end of leftRegion
```

```
  currentLine := selected new line
```

```
  until left side of currentLine = marker
```

```
  if currentLine <> last line drawn by user
  or left hand marker = right hand marker
    return false
  endif
```

```
  set marker to a new, unused value
```

```
  construct right hand loop as above, taking
  tightest right hand turns
```

```
  if currentLine <> last line drawn by user
    return false
  endif
```

```
  remove non-region separating lines from
  leftRegion, rightRegion
```

```
  remove lines from pool_of_lines that were
  incorporated in leftRegion, rightRegion
```

```
  return true
  // and values for leftRegion, rightRegion,
  // pool_of_lines
```

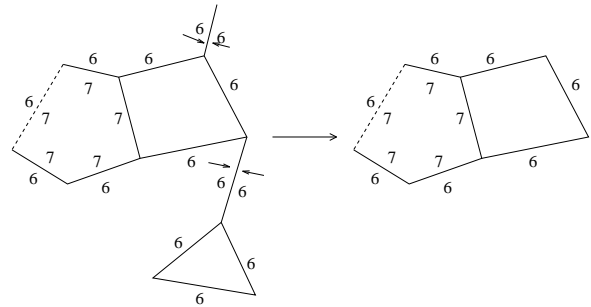


Figure 7: The removal of non-region-separating lines

Once this is complete, we have an updated copy of the region information, with the two line loops producing being passed onto the drawing update module.

4.4. Drawing Update

Drawing update takes the two line loops (forming two picture regions) as input and produces an updated version of the completed sections of the drawing if necessary. All lines lying on the new silhouette are marked

as occluding the background. Any new lines in the two regions are linked to the lines in the partially completed drawing and the junctions are assigned the correct type to reflect the additional lines (for instance, if an **L** junction has an extra line connected to it, it must now be categorised as a **Y**, **T** or **Arrow** junction).

We must decide how the two junction loops fit into the current drawing; five separate cases may arise:

1. A new disjoint region (independent of other regions) is formed, where one junction loop forms the interior of the new region, and the other junction loop forms the exterior; see Figure ??, where the dotted line indicates the last line drawn by the user, and the numbers on either side of a line give the region numbers. Line labels are shown for the silhouette.
2. A new disjoint region is formed containing an old region, the junction loops forming the interior and exterior of the new region; see Figure ??.
3. A new disjoint region contained inside an old region is formed. One junction loop is the interior of the new region, the other is the exterior; see Figure ??.
4. A new connected region external to an object is added, where one junction loop forms the new silhouette of the entire object and the other forms the boundary of a new region; see Figure ??.
5. An existing region internal to an object is split, and each junction loop forms the boundary of a new region, see Figure ??.

In any case, we update the region structure of the drawing as appropriate, creating a new drawing region where required.

```
function DrawingUpdate(leftRegion,rightRegion : list
of lines, var currentDrawing : complete sections
of current drawing) : list of lines

begin case
  case: leftRegion == rightRegion // cases 1,2 or 3

    create new region from leftRegion, and include
    in currentDrawing
    // update currentDrawing as each new region
    // is formed

    give new region number to inside of new region

  begin case
    case: leftRegion is not contained in a region
    // cases 1 or 2

    mark outside of new region as region 0
    (background)

    label boundary of new region as occluding
    the background
```

```
if new region does not contain other regions
then return empty list // case 1

else // case 2
  mark outside of largest contained region as
  the associated number of new region

  return lines of largest contained region
endif

default: // case 3
  mark outside of new region with associated
  number of smallest containing region

  if new region contains other regions then

    mark outside of largest contained region
    with number of new region
  endif

  return lines of new region and regions
  it contains
endcase

case: smaller of leftRegion and
rightRegion is connected to silhouette and is
external w.r.t. line labelling of silhouette
// case 4

  create new region from smaller region

  label lines in larger region as new silhouette

  return lines belonging to leftRegion and
  rightRegion

default: // case 5
  create new region from rightRegion

  find old region number marked in rightRegion

  mark all lines in leftRegion with old
  region number

  mark all lines in rightRegion with
  associated number of new region

  return lines belonging to leftRegion and
  rightRegion
endcase

// all cases return required list of lines to
// be processed
```

4.5. Line Labelling

Line labelling takes the current state of the drawing and a list of all lines affected by the latest line region updates. The labelling is updated by a constraint

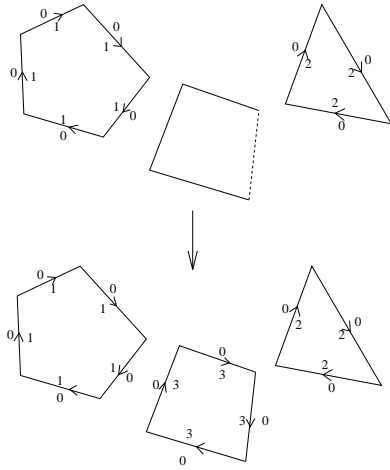


Figure 8: Case 1—A new disjoint independent region

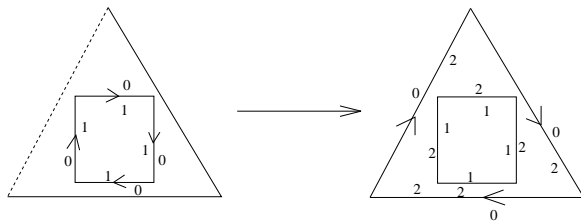


Figure 9: Case 2—A new disjoint region containing another region

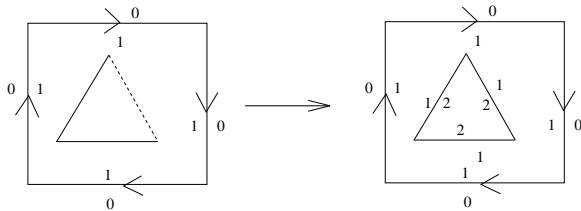


Figure 10: Case 3—A new disjoint region contained inside another region

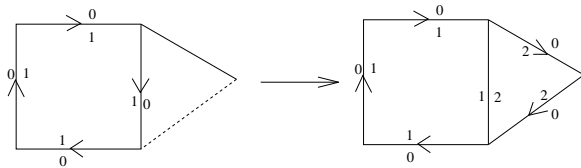


Figure 11: Case 4—A new connected external region

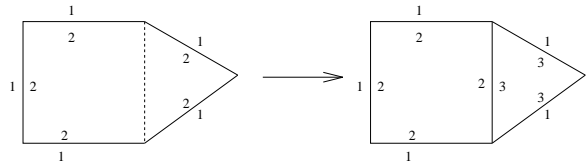


Figure 12: Case 5—A new connected internal region

propagator which reduces the possible labellings at junctions.

This module accepts a list of lines and the current state of the drawing as input, and updates the labelling for the current state of the drawing. This works incrementally as it is called every time a new region is completed in the drawing. To ensure that all possible labellings that could result from the amendments to the drawing are considered, the possible labellings at the end junctions of the new lines passed to this module are all reset (such that all legal labellings are now possible; silhouette labellings are permitted on all lines incident to the background face, and additional L labellings are permitted on internal junctions) each time the module is called. Each junction then has any trivially impossible labellings removed (due to any lines already given labels such as silhouette lines) to reduce the amount of work carried out by the propagation. The list of lines is then passed over to a constraint propagator.

```

procedure IncrementallyLabelDrawing(lineList : list
of lines affected by creation of new region,
var currentDrawing : complete sections of
current drawing)

```

reset line label at each line in lineList

reset junction catalogues at each end of each line
lineList

remove trivially impossible junction labellings

propagate constraints, updating currentDrawing
// updated currentDrawing is returned

4.6. Completion

Once the user has completed sketching an object, all temporary junction labels are now considered invalid; they must be removed from the drawing. The line labeller carries out a final pass through the drawing, and produces one (or more) final labellings in which only the legal trihedral labellings are allowed.

5. Results

In this section we present several drawings; an L-block, bracket and bench-block in Figures ??-??, with a cube and truncated pyramid are shown in Figures ?? and ??. The Cube and Truncated Pyramid are shown together with their stages of incremental labelling.

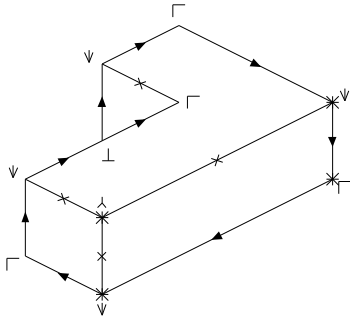


Figure 13: Final labelling produced for an L-block

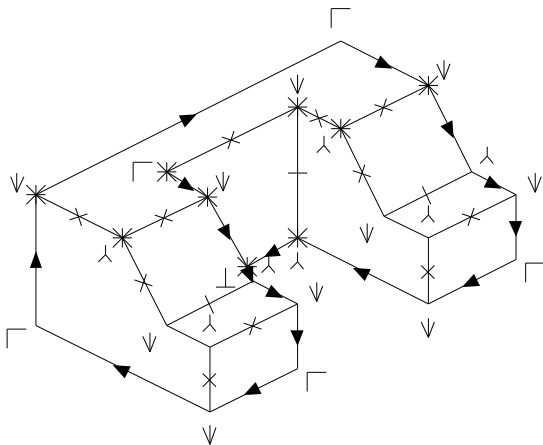


Figure 14: Final labelling produced for a bracket

For example, Figure ?? presents the creation of a cube by the user. The figures are titled with the name of the object and the number in the sequence; a letter after the number indicates that this is one of the possible interpretations of this scene. Each interpretation figure also shows the number of trihedral and non-trihedral junction labellings possible, together with the number of unconnected junctions and the number of the interpretation. Junctions marked with a star are those whose labelling possibilities were reset when the new region was created. Note that the possible

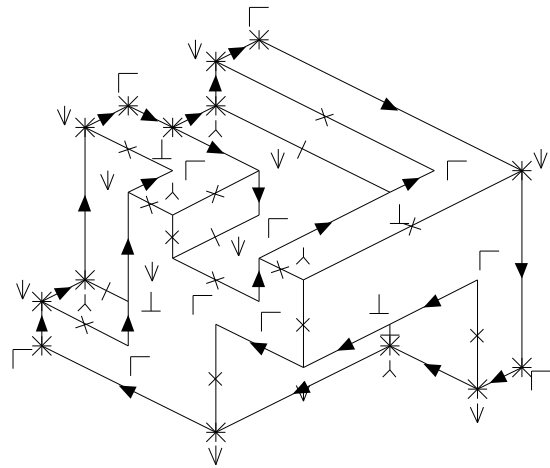


Figure 15: Final labelling produced for a bench block

labellings are only shown the first time they change, which is once a new face is completed.

The timings of our tests are shown in Table ??; the “Total Incremental” column shows the relative total run-time of our incremental algorithm compared to the standard single-pass labelling algorithm. “Temporary Label Removal” shows the time taken to remove all temporarily valid labellings upon completion of the drawing, and “Final Interpretation” indicates the time taken to produce the final labelled drawing (after the temporary labels have been removed); all times are with respect to the total time for the single pass algorithm.

The algorithm takes longer to run if the individual steps are added up. However, as this processing is carried out in the background whilst the user sketches, this is unimportant. It is the final delay perceivable to the user after sketching which matters, and this is smaller than for the single pass algorithm. Note that there are greater savings for the final delay with more complex objects, such as the bench block and bracket.

6. Conclusions

Our incremental algorithm is robust and straightforward. Note that there are no restrictions placed upon the order in which the user may construct a drawing of a trihedral object. In all tested cases there was a decrease in the final delay noticed by the user.

Although our method is somewhat more complex to implement than the standard labelling approach, it is worth using as it provides increased user responsive-

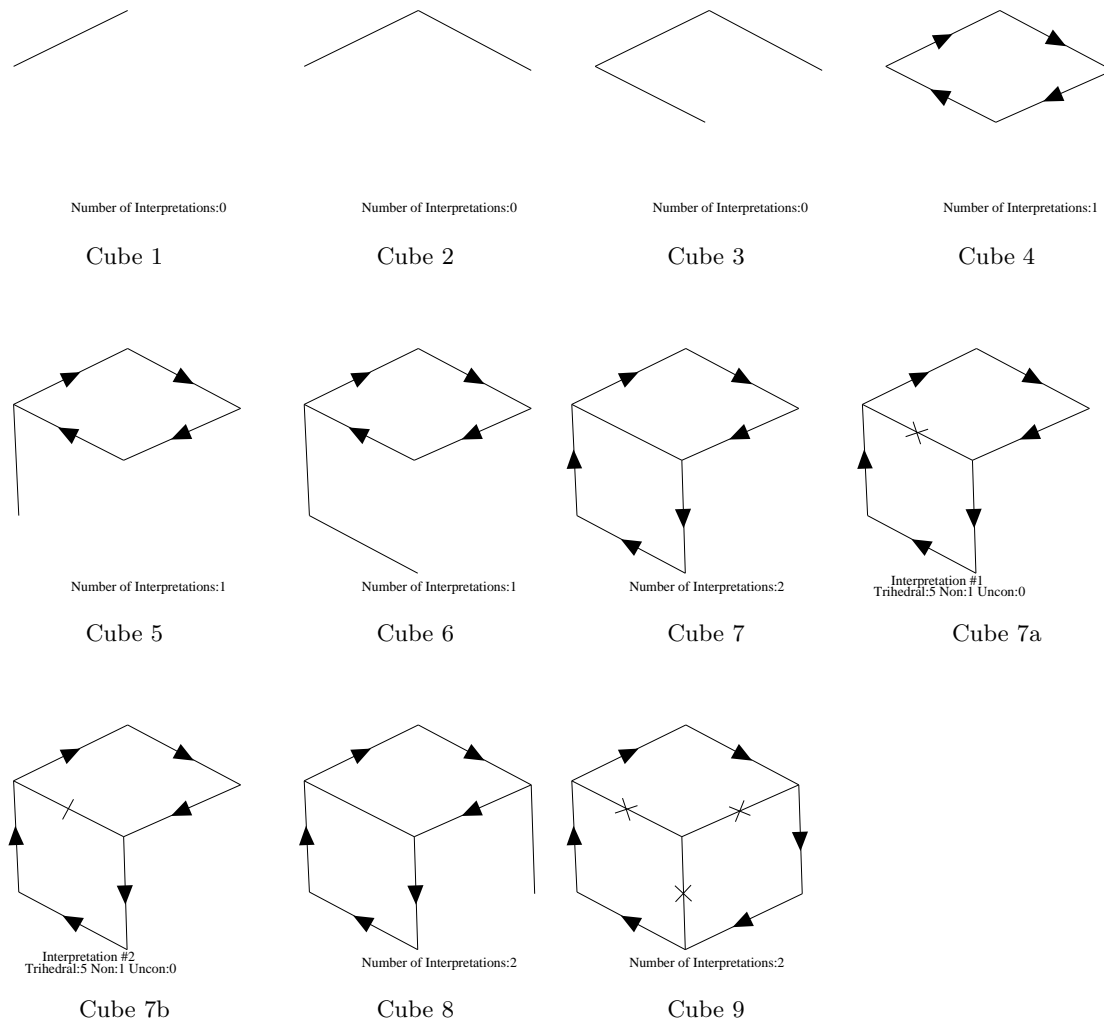


Figure 16: Creation of a cube

Drawing	Total Incremental	Temporary Label Removal	Final Interpretation
Cube	234%	3%	68%
Truncated Pyramid	334%	3%	52%
L-Block	276%	4%	58%
Bracket	580%	1%	30%
Bench Block	566%	1%	30%

Table 1: Timings of our incremental algorithm relative to a single pass algorithm

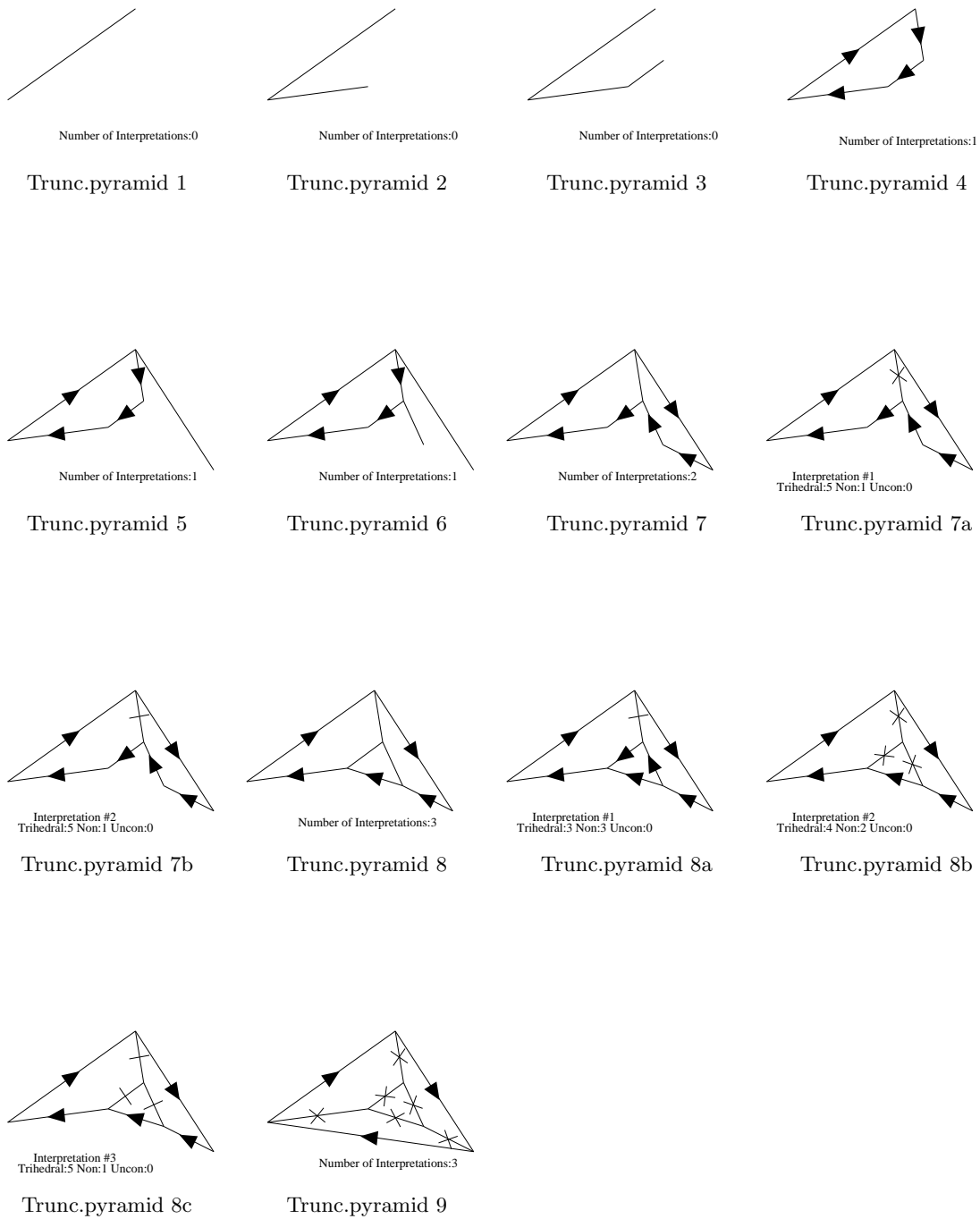


Figure 17: Creation of a truncated pyramid

ness. A side benefit is that the region information is also incrementally produced.

For a more realistic sketch input system, the method needs to be extended to handle non-trihedral and curved cases. This would involve extending the junction dictionary and junction classification. In principle this should be straightforward, and can be based on the work (for instance) of Malik⁷,[?].

References

1. G. Goldschmidt, "Serial sketching: Visual problem solving in designing", *Cybernetics and Systems* **23**, pp. 191–219 (1992).
2. D.L. Jenkins and R.R. Martin, "The importance of free-hand sketching in conceptual design: Automatic sketch input", in *Design Theory and Methodology DTM93*, T.K.Hight and L.A.Stauffer, Eds., **DE-53**, ASME, pp. 115–128 (1993).
3. D.G. Ullman, S. Wood and D. Craig, "The importance of drawing in the mechanical design process", *Computers and Graphics*, **14**(2), pp. 263–274 (1990).
4. B.J. Hale, R.P. Burton, D.R. Olsen and W.D. Stout, "A three-dimensional sketching environment using two-dimensional perspective input", *Journal of Imaging Science and Technology* **36**(2), pp. 188–196 (1992).
5. C. Herot, "Sketch recognition for computer-aided design", in *User Orientated Design of Interactive Graphics Systems*, ACM/SIGGRAPH Workshop, Pittsburg, PA, S. Treu, Ed., pp. 31–35 (October 1976).
6. E.A. Bier, "Snap-dragging in three dimensions", *ACM SIGGRAPH Computer Graphics—Special Issue: 1992 Symposium on Interactive Computer Graphics*, **24**(2), pp. 193–204 (1992).
7. Y. Fukui, "Input method of boundary solid by sketching", *Computer Aided Design*, **20**(8), pp. 434–440 (October 1988).
8. A. Jacot-Descombes and T. Pun, "A probabilistic approach to 3-D inference of geons from a 2-D view", in *SPIE Conference on Applications of Artificial Intelligence X: Machine Vision and Robotics, Proceedings 1992, Orlando, Florida*, **1708**, pp. 579–588 (1992).
9. R.C. Munck-Fairwood and L. Du, "Shape using volumetric primitives", *Image and Vision Computing*, **11**(6), pp. 364–371 (1993).
10. D.J. Kriegman and J. Ponce, "On recognizing and positioning curved 3D objects from image contours", *IEEE Transactions On Pattern Analysis and Machine Intelligence*, **12**(12), pp. 1127–1137 (December 1990).
11. Y.G. Leclerc and M.A. Fischler, "An optimization-based approach to the interpretation of single line drawings as 3D wire frames", *International Journal of Computer Vision*, **9**(2), pp. 113–136 (1992).
12. S.A. Shaffer, T. Kanade and J. Kender, "Gradient space under orthography and perspective", *Computer Vision, Graphics and Image Processing*, **24**, pp. 182–199 (1983).
13. F. Ulupinar and R. Nevatia, "Constraints for interpretation of line drawings under perspective projection", *Computer Vision, Graphics and Image Processing: Image Understanding*, **53**(1), pp. 88–96 (January 1991).
14. I.J. Grimstead and R.R. Martin, "Creating solid models from single 2D sketches", in *Proceedings of Third Symposium on Solid Modelling and Applications*, C. Hoffman and J. Rossignac, Eds., ACM Press, pp. 323–337 (1995).
15. P.H. Winston, *Artificial Intelligence (3rd ed.)*, Addison Wesley (1992).
16. M.B. Clowes, "On seeing things", *Artificial Intelligence*, **2**, pp. 79–116 (1970).
17. D.A. Huffman, "Impossible Objects as Nonsense Sentences", in *Machine Intelligence 6*, American Elsevier, New York, pp. 295–323 (1971).
18. J. Malik, "Interpreting line drawings of curved objects", *International Journal of Computer Vision*, **2**(1), pp. 73–103 (1987).
19. J. Malik and D. Maydan, "Recovering three-dimensional shape from a single image of curved objects", *IEEE Transactions in Pattern Analysis and Machine Intelligence*, **11**(6), pp. 555–566 (June 1989).
20. P.V. Sankar, "A vertex coding scheme for interpreting ambiguous trihedral solids", *Computer Graphics and Image Processing*, **6**, pp. 61–89 (1977).
21. D.L. Waltz, "Understanding line drawings of scenes with shadows", in *The Psychology of Computer Vision*, P.H. Winston, Ed., McGraw-Hill, New York, pp. 19–91 (1975).
22. K. Sugihara, "An algebraic approach to shape-from-image problems", *Artificial Intelligence*, **23**, pp. 19–91 (1984).

23. T. Kanade, "Recovery of the three-dimensional shape of any object from a single view", *Artificial Intelligence*, **17**, pp. 409–460 (1981).