CMU-CS-83-113

Topological Correspondence in Line Drawings of Multiple Views of Objects

Charles E. Thorpe
Steven Shafer

Carnegie-Mellon University
14 March 1983

Abstract

As an object moves relative to a viewpoint, its appearance changes. In this paper we analyze the topological constraints on the changing appearance of line drawings of objects as the objects or the camera move. We start with a Huffman-Clowes junction dictionary. We show a way of deriving vertex types from junction types by inference rather than by table look-up, and develop a set of transition tables, showing the change in appearance of vertices as the viewpoint changes. We derive three constraints on the change in appearance of an object: conservation of vertices, conservation of vertex type, and conservation of adjacencies. Using these constraints, we develop a matching algorithm that traces vertices from one image to the next. Examples are given showing correct matching for simple objects, including partially visible objects and multiple objects in the same scene.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-81-K-1539.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation and Assumptions

In this paper we present a method for matching vertices in line drawings of multiple views of objects. First, we examine the appearance of trihedral vertices from different viewpoints. This analysis is used to derive the effects of changing viewing position on the appearance of line drawings of trihedral objects. That in turn helps solve the problem of identifying the same vertices in different pictures.

The first part of this paper examines the change in appearance of a vertex as the viewing angle changes. We show the correspondence between three-dimensional vertices and the Huffman-Clowes labels attached to image junctions. We then develop a "transition table", showing how the image of a vertex can change from one type of junction to another as the viewpoint changes.

The objects we consider are trihedral blocks; that is three planar sides meet at each vertex. The drawings are initially presumed to be perfect. The change in viewing position can come either from moving the camera, as in stereo vision, or from moving objects. One reason for working in the trihedral blocks world is to understand the effects of motion in an idealized case, separate from the problems of feature extraction. Another reason is that in analysis of aerial images of urban scenes, many of the main features are trihedral buildings.

In the second part we show a matching process that relies on topology to identify the same vertices in two line drawings of the same scene. The only information used is line labels and connectivity. These give us three constraints: conservation of vertices, conservation of adjacencies, and conservation of vertex types. This process works for scenes with more than one object or with only part of an object visible. We discuss simplifications and extensions and show several simple examples.

2. Trihedral Vertices

2.1 Introduction: Huffman-Clowes Junction Dictionaries

Huffman [5] and Clowes [3] independently developed "junction dictionaries", lists of possible appearances in line drawings of all configurations of trihedral vertices. They first note that there are only two types of edges: concave and convex. The two planes that form the edge divide space into four sections, with the edge appearing topologically different from each section. In the case of a concave edge, the edge will be obscured from three of the sections and will appear simply as a concave edge from the fourth, labeled "-" by Huffman. For a convex edge, the edge will be visible from three sections. From the front, it will appear as a convex edge, designated with a " + ". From either side, it will appear as an occluding (obscuring) edge, with the nearer surface blocking the farther surface from view. Huffman labels occluding edges with an arrow along the edge such that the surface on the right of the arrow occludes the surface on the left.

Trihedral vertices are formed by the intersection of three surfaces, which is also the intersection of three edges. The three surface planes divide space into eight sections, usually called octants without implying that they must be of equal size or of similar shape. Some of the octants can be filled and the rest left empty. The combinations of filled and empty octants that give single, connected objects with three surfaces meeting at the center have one, three, five, or seven filled octants. These vertex types are usually named by Roman numerals according to the number of filled octants. Each edge that meets at a vertex can be either convex or concave. In a type I vertex, all three edges are convex. In a type III vertex, there are two convex and one concave. Type V vertices have one convex and two concave edges, and all three edges of a type VII vertex are concave.

Huffman and Clowes point out that each type of vertex can be viewed from each of the empty octants. The image of a vertex is called a junction. By viewing all vertex types from all empty octants they construct a "junction dictionary" showing all topologically different types of junctions that can occur in the trihedral world. Table 1 shows all junctions, each listed by type of vertex of which it is an image.

2.2 Reasoning From Junction Type to Vertex Type

Notice in table 1 that each junction type can result from only one kind of vertex. For instance, the arrow-shaped junction with one + and two > labels can only be the image of a convex-convex-convex

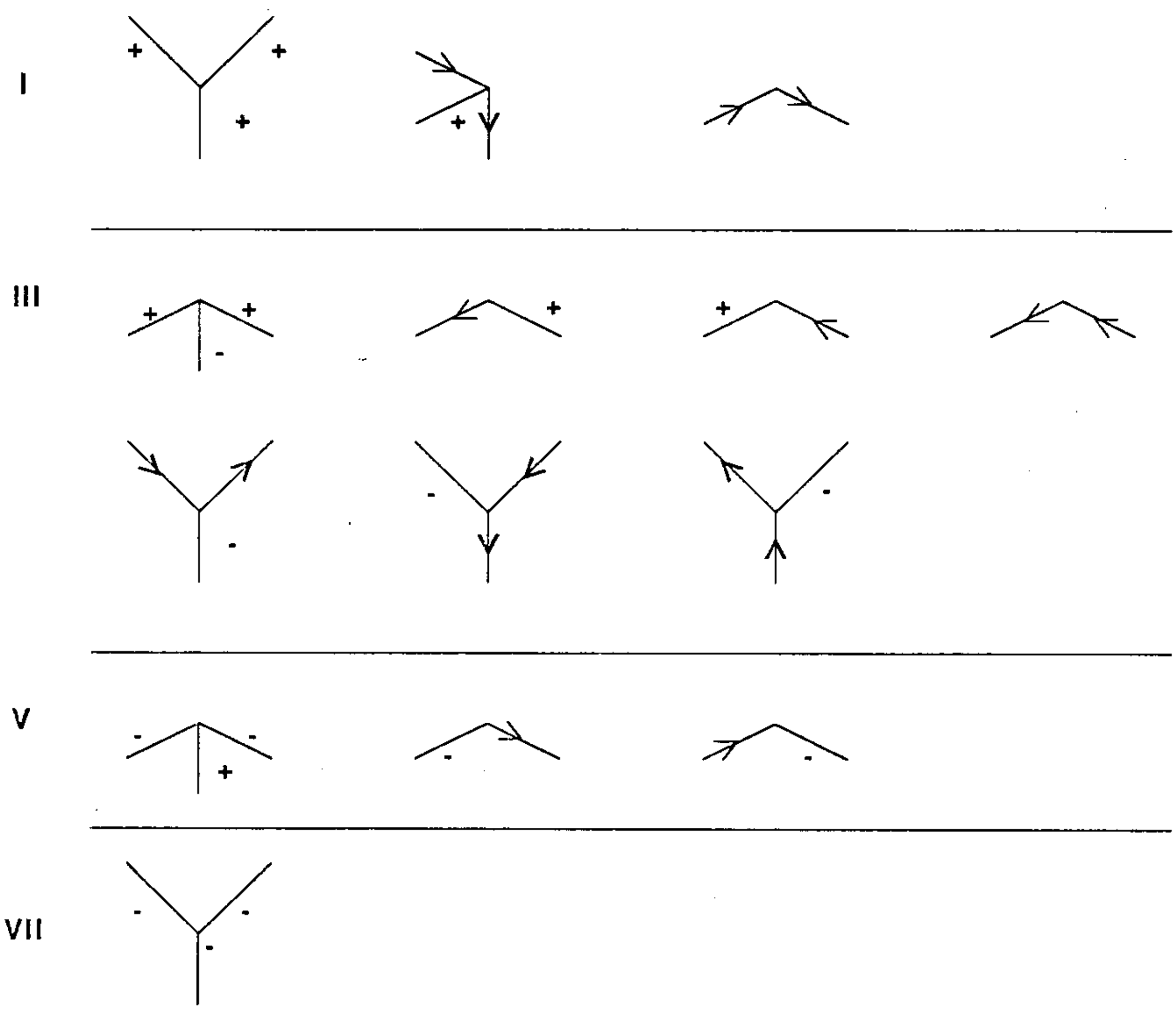


Table 1: Summary Of Junction Types (From [5])

(type I) vertex. So if the labels of all lines on a junction are known, it is possible to tell what three-dimensional configuration must be pictured.

There is a straightforward explanation in terms of edge types. Remember that a convex edge can take any of three labels: +, >, or <. A concave edge, on the other hand, can only be labeled with a minus. So lines with arrow or + labels must be convex. This means that every junction with three lines visible and labeled is easily traceable to the type of vertex that formed it. For instance, the Y junction labeled with two arrows and one minus must come from the type III configuration, since two of its edges are convex and one concave.

This reasoning relies on the visibility of all three edges at a junction. Thus, more analysis is required for L junctions. The key is to find where the hidden third edge must be. This gives the location of the hidden surface or surfaces, and thus gives the edge types.

The first case is when the edge is within the angle ABC (figure 1), and there is only one surface visible. Since the third edge is hidden, the area within arc ABC must be the one visible surface. Then AB and BC both appear as occluding edges, which means that they are both convex. The object must lie below the arc ABC, and the two hidden surfaces must meet in a convex edge. So this is the all-convex vertex, type I.

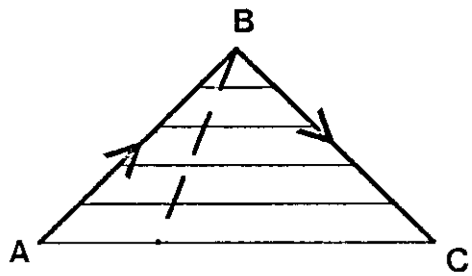


Figure 1: L Junction With One Surface Visible

If the hidden edge is within the arc DBE, the opposite situation exists. The one visible surface is the area outside of arc ABC (figure 2). Edges AB and BC are still occluding, and hence convex, but the direction of their labeling arrows is reversed. The object lies entirely under the visible surface, and the invisible surfaces must meet in a concave edge. This must be a type III vertex.

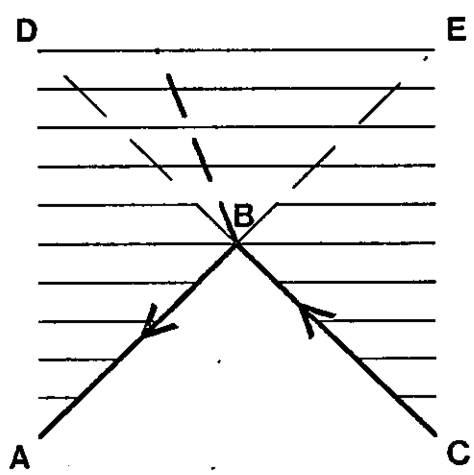


Figure 2: L Junction With The Opposite Area Visible

As the hidden edge swings around into arc ABD (or, in the mirror image, arc CBE), one of the hidden surfaces now becomes partly visible (figure 3). The newly visible surface 1 joins the "top" surface 2 along edge BC; that edge is then labeled with a plus rather than an arrow. The one remaining hidden surface meets surface 2 along edge AB and must extend back underneath it. Its intersection with surface 1 along the hidden edge will then be concave. These must also be formed by vertices with two convex and one concave edge, again type III.

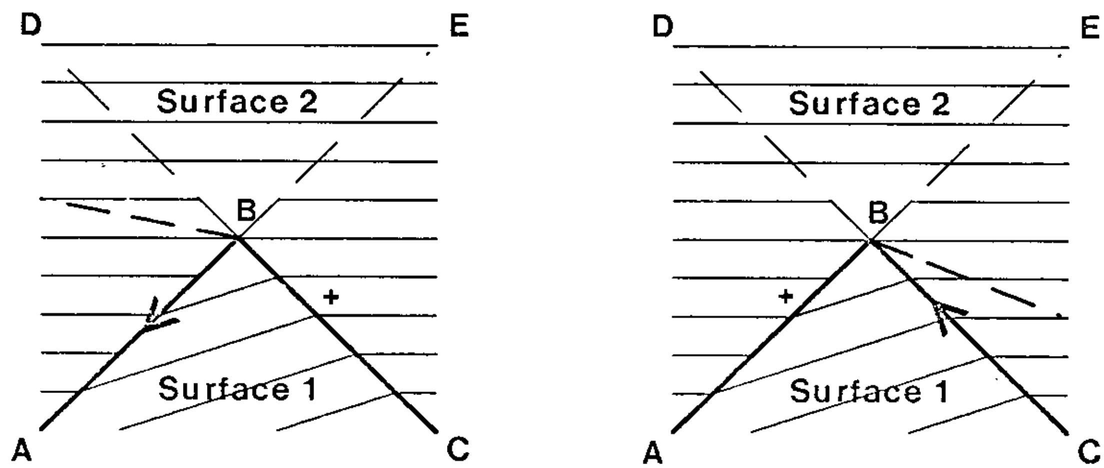


Figure 3: Two Surfaces Visible, Hidden Edge In Arc ABD Or Arc CBE

The final case is two surfaces visible and the third edge hidden by surface 1. The hidden edge is formed by the third surface and surface 2. Since this edge is under surface 1, surface 1 must be in front of surface 2 at that point. This means that the intersection of surfaces 1 and 2 must be concave, since surface 1 sticks out over surface 2. The other edge of surface 1 must then be occluding. Then the hidden surface must run from the occluding edge to the hidden edge, and must form a concave edge with surface 2. This is the one convex two concave (type V) case. The two symmetric cases of this are shown in figure 4.

Note that there are a series of constraints that may be interchanged. It is not necessary to know line labels, surface visibilities, and hidden edge location in order to derive vertex type. For instance, in figure 1, it would be sufficient to know that there is a surface within arc ABC and none outside. In that case, all the rest of the information would be constrained. More information than just surface visibility would be needed to distinguish between cases like figures 3 and 4, but even there line label information could be traded for knowledge of hidden line position. This information could come from other sources, such as intensity or orientation. For instance, in urban scenes there is very often a vertical edge at each vertex. If the camera geometry is known, it is possible to predict the direction in

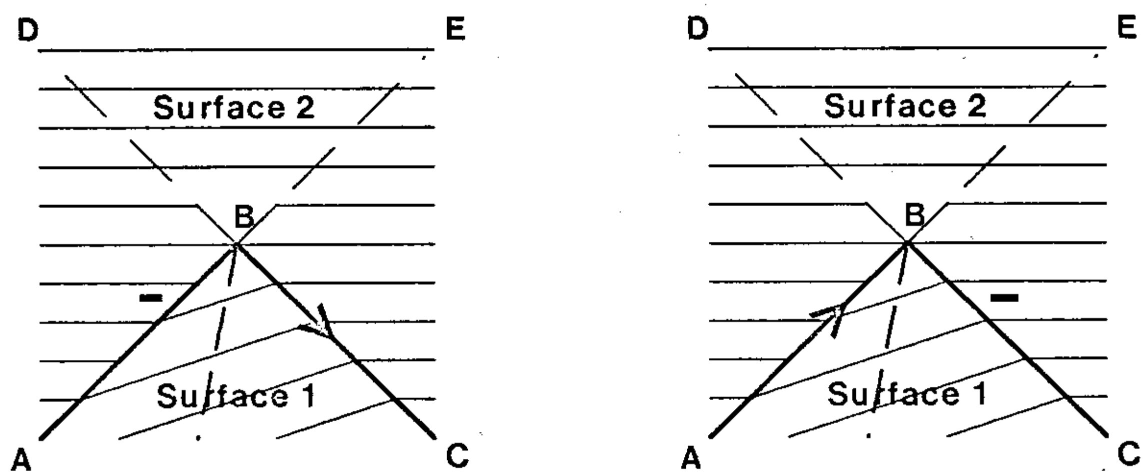


Figure 4: Two Surfaces Visible, Hidden Edge In Arc ABC

the image of the vertical edge. If the vertical edge is the missing one in an L junction, it would be possible to determine the type of the vertex.

2.3 Effect of Motion on Appearance of Trihedral Vertices

As the viewpoint moves relative to the vertex, the appearance of the vertex may change. As long as the viewpoint stays in one octant, the image of that vertex remains the same type of junction. But when the viewpoint crosses into another octant, the vertex will appear as a different junction type. There are several constraints on the change in junction type. The first is conservation of vertex type. Since the underlying vertex type remains the same, the junctions must all be images of that type of vertex. So, for instance, a junction in row I of table 1 can become, due to viewpoint change, any other junction in row I, but cannot become a junction from rows III, V, or VII. The second constraint is that each octant is adjacent to three other octants. As the viewpoint changes, it can only go from an octant to one of the three adjacent octants. So the junction type can only change to the junction type derived from an adjacent octant. These two constraints give a transition graph, showing all possible junction type transitions, as in figures 5 and 6. The conservation of type constraint splits the graph into four disjoint parts, one for each type of vertex. Each part of the graph has eight nodes, one for each octant, with 1, 3, 5, or 7 of the nodes marked "invisible" because they correspond to viewing positions behind the object.

Another constraint comes from the direction of the change. Changing octants means crossing one of the three planes that meet at the vertex. If the motion of the camera is known to be parallel to one of the planes, or is known not to be large enough to cross a plane, the possible junction type

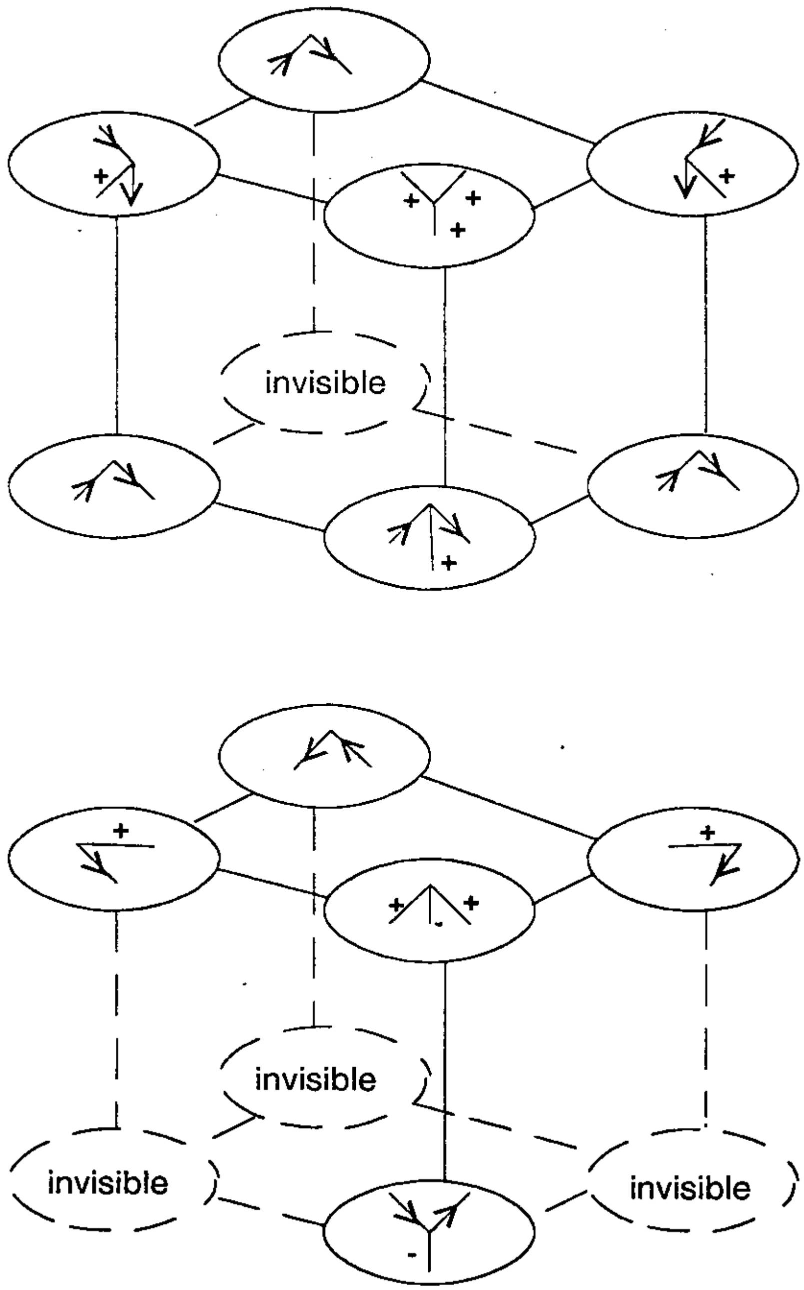
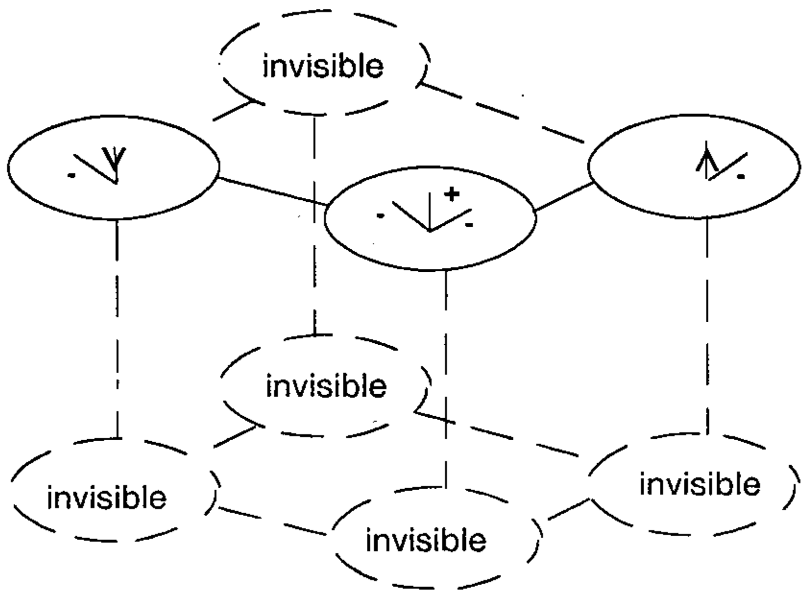


Figure 5: Transition Table, Type I and Type III

transitions are constrained. This is often the case in stereo analysis of aerial photographs of urban scenes. For buildings with horizontal roofs, the camera is always above the plane of the roofs. If the buildings also have vertical walls, it is easy to tell on which side of the plane of the walls the camera lies. Note that this does not guarantee the appearance of a given type of vertex, because the orientation of the vertex is not constrained. For instance, a type III vertex can be oriented as an "L" on its side, as an upright "L", or as an inverted "L". So "above the plane of the roofs" can still be in



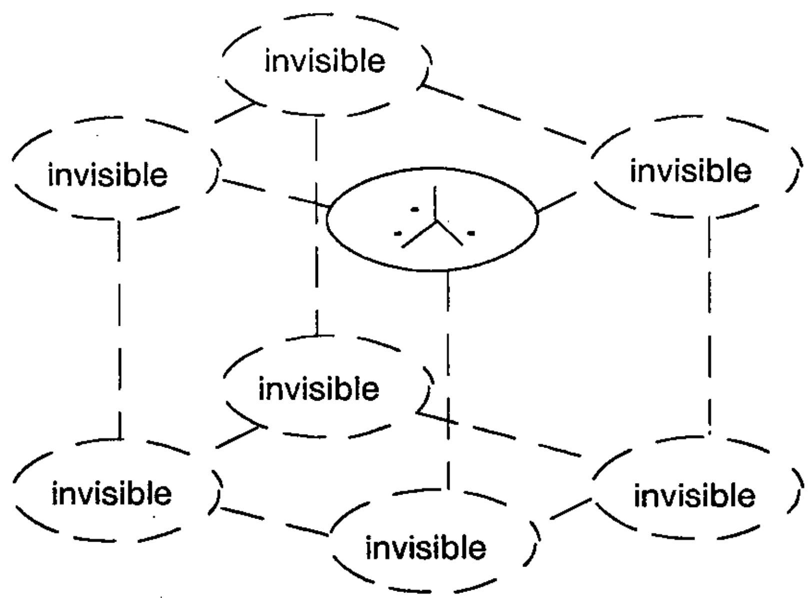


Figure 6: Transition Table, Type V and Type VII

any of the octants relative to the vertex. However, given two images of the same building, it is possible to tell if the viewpoint crossed the plane of either wall. This constrains the change in appearance of the junction.

2.4 Orthogonal Trihedral Vertices

Another type of constraint is available when all vertices are composed of orthogonal edges. Under perspective projection, all lines parallel in the scene will pass through a common "vanishing point" in the image plane. If all lines are aligned in one of three orthogonal directions, it is easy to find the vanishing points and then to classify each line. Then the orientation of each vertex relative to the camera can be established and junction labeling is greatly constrained.

Liebes' treatment of orthogonal trihedral vertices [7] ignores line labels and uses only line directions. The advantage of this approach is that a junction can be catalogued by itself, without looking at adjacent junctions, since only the direction of the lines at that junction matters. The effect is that the appearance of that vertex in another view can be predicted, but it is not possible to tell whether it will be self-obscured. As an example, consider an arrow-shaped junction pointing "down". If this is an image of an orthogonal trihedral vertex, it could be either a top view of a type V vertex or an inverted view of a type I vertex. There are two ways to tell which is correct. First, other information, such as constraints from neighboring junctions, can be used to label the lines. Second, the vertex can be viewed from an octant from which one type of vertex would be visible but the others self-obscured. Since Liebes does not use the first type of information, he cannot make predictions about the second type of constraint.

3. Correspondence

In this part of the paper, we show how to find correspondences consistent with the topological and line-label constraints. The correspondence problem is identifying the same three-dimensional point in two different images of the same scene. Finding matching points is an essential step in tracking object motion and in calculating depth from stereo images. This paper deals with tracking the vertices of trihedral blocks from one line drawing to another. The two views can come from either stereo or from object motion. We make no assumptions about camera geometry (such as known epipolar lines), fixed relations among objects, or magnitude of the change between images. We assume that some other process has done Huffman-Clowes labeling on each image. This carries with it implicit assumptions of completeness of the scene and of a "general viewpoint", that is, no coincidental alignments in the image.

The main reason we are interested in using topological information by itself is to understand the constraints it provides. We can then make better use of those constraints when we combine them with quantitative information.

3.1 Constraints

The object of correspondence is to find a match in the second image for each junction in the first image. There are three constraints that a complete, consistent set of matches has to satisfy: conservation of vertices, conservation of type, and conservation of adjacencies.

Conservation of vertices means that since the same objects are in each image, the same vertices must also be present. They may not be visible in each image, however. Under perspective projection, it would even be possible to have all vertices that are visible in one image invisible in the other. For instance, one view could be looking at a cube from straight in front, so only the front face and the front vertices are visible, and the other view could be from directly behind the cube, with only the back face and vertices visible. Under orthography, at least some vertices must be visible in both images, but there may still be other vertices that are invisible in both of the images. Even though they are not visible, the other vertices are still present in the scene, and have to be accounted for.

Conservation of type means that a vertex must always keep the same shape. Although its appearance may change, that change is constrained (see section 2.3). Since we assume that all lines are labeled, every junction's type is known. A junction in one image can therefore only match a junction in the second image that is the image of the same type of vertex.

Conservation of adjacency is really conservation of edges. If two junctions in one image are directly connected by an edge, the junctions they match in the second image must have a line connecting them or the possibility of an invisible edge between them.

These constraints allow some types of noise in the scene. Missing lines can be handled by the "invisible edge" criterion. Extra lines cannot be handled. Changing angles can be tolerated, since quantitative geometry is not important, but they must not change from concave to convex or vice versa, since vertex types, and thus edge types, must remain fixed. Changing line length is permitted, which may be especially useful in real scenes. But polygonal approximations to curved surfaces will probably not work, since it is difficult to guarantee the same number of vertices in two different polygon fits.

3.2 Matching Algorithms

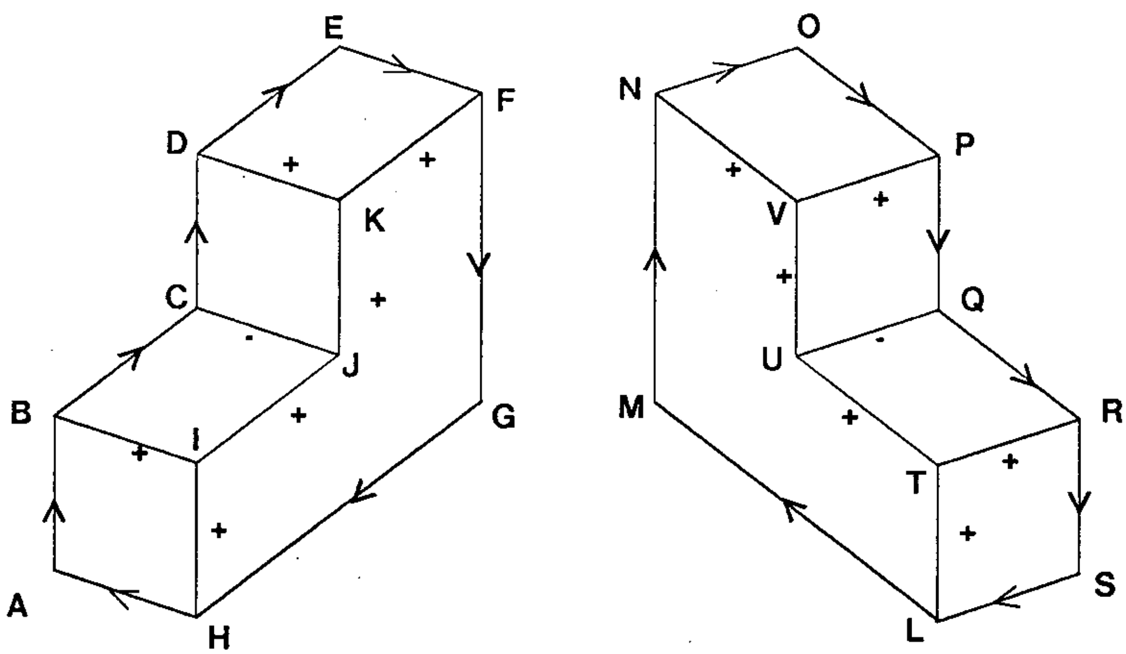


Figure 7: Two Labeled Pictures of an L Block

3.2.1 Correspondence Graph

Our matching algorithms are based on a central data structure, the correspondence graph. Each node in the correspondence graph is composed of a junction from one image and a junction from the second image, where the two junctions may be images of the same actual vertex. Node formation uses the "conservation of type" constraint. In figure 7, junctions B and T are both images of a type I vertex, so BT would be a node. Junction U is an image of a type III vertex. Since B and U come from different types of vertices, they cannot match, so BU would not be a node.

Invisible vertices must also be allowed for in the graph. Unless we know otherwise from outside sources of information, we must assume that there can be any number of invisible vertices and that they can be of any type. We create the special junction (invisible) that can match any or all of the junctions. Figure 8 shows all the nodes formed from the L blocks in figure 7.

AL, AM, AN, AO, AP, AR, AS, AT, AV, A(invisible),
BL, BM, BN, BO, BP, BR, BS, BT, BV, B(invisible),
CQ, CU, C(invisible),
DL, DM, DN, DO, DP, DR, DS, DT, DV, D(invisible),
EL, EM, EN, EO, EP, ER, ES, ET, EV, E(invisible),
FL, FM, FN, FO, FP, FR, FS, FT, FV, F(invisible),
GL, GM, GN, GO, GP, GR, GS, GT, GV, G(invisible),
HL, HM, HN, HO, HP, HR, HS, HT, HV, H(invisible),
IL, IM, IN, IO, IP, IR, IS, IT, IV, I(invisible),
JQ, JU, J(invisible),
KL, KM, KN, KO, KP, KR, KS, KT, KV, K(invisible),
(invisible)L, (invisible)M, (invisible)N, (invisible)O, (invisible)P, (invisible)Q,
(invisible)R, (invisible)S, (invisible)T, (invisible)U, (invisible)V, (invisible)(invisible)

Figure 8: Nodes Formed From Picture of L Block

Links in the graph connect nodes that are "consistent". Consistent means that both nodes could be part of the same overall match. Inconsistent means that if one node is correct, the other is incorrect. Consistency is defined using the "conservation of adjacency" constraint and its logical implications. We define the *distance between two vertices* to be the smallest number of edges that must be traversed to get from one vertex to the other. If all vertices and all edges were visible in an image as junctions and lines, it would be straightforward to determine the distance between two junctions. Then, given two nodes XX' and YY', they would be consistent only if the distance from X to Y in one image was exactly the same as the distance from X' to Y' in the other image. For instance, node AL is consistent with node IR since the distance from A to I (two edges) is the same as the distance from L to R. The difficulty with this is that all vertices might not be visible, so it might not be possible to tell what the shortest distance is between two junctions. The way we handle this is to calculate the upper and lower bounds of the distance. The upper bound is the shortest distance using only visible lines. The lower bound uses invisible lines wherever they could possibly run. Then the criterion for compatibility is that the range of distances must overlap. That is, the lower bound of the distance from X to Y must not exceed the upper bound of the distance from X' to Y' and the upper bound from X to Y must be greater than or equal to the lower bound from X' to Y'.

In figure 7, the upper bound on the distance from A to E is 4. In order to calculate the lower bound, we have to decide if there could be an edge directly connecting A and E. This takes some reasoning about the planes that form the surfaces of an object. If the lines are labeled, it is easy to tell which lines and points lie in the same plane. A line labeled + or - lies in the planes visible on either side of it. An occluding line lies in the hidden plane and the occluding plane but not the occluded plane. Then in the example, lines FK and KJ lie in the same plane. So do lines JI, IH, and HG. So points F, G, H, I, J, and K all lie in a plane, called plane 1. Similarly, if there is an edge from E to A, then A, E, F, G, and H all lie in a plane, plane 2. Since F, G, and H all lie in both plane 1 and plane 2, the planes must be identical. ABIH is also a plane, and since A, I, and H are in plane 1, B must also be. So must C and D. But then the whole figure lies in plane 1, and is not a trihedral figure at all. This argument can be formalized and extended to show that in a line drawing of a trihedral object there can never be just one junction with only two lines accounted for. If there were a line from A to E, G would be the only vertex with two known edges. Since this is impossible, we infer that there must be at least one hidden vertex, and no direct connection from A to E. So the lower bound on the distance from A to E is two.

The lower bound on the distance from (invisible) to a junction is the distance from that junction to the nearest junction with less than three lines, plus one. The upper bound is infinite, since the connections between invisible junctions are unknown. Figure 9 shows all the links for node AL.

BM, BT, BS,
CU, C(invisible),
DO, DV, D(invisible),
EN, EO, EP, ER, EV, E(invisible),
FO, FV, F(invisible),
GN, GR, G(invisible),
HM, HT, HS,
IN, IR, I(invisible),
JQ, J(invisible),
KP, K(invisible),
(invisible)M, (invisible)N, (invisible)O, (invisible)P, (invisible)Q, (invisible)R,
(invisible)S, (invisible)T, (invisible)U, (invisible)V, (invisible)(invisible)

Figure 9: Links of Node AL

3.2.2 Searching the Correspondence Graph

A complete match consists of a subgraph of the correspondence graph such that the nodes contain every junction in each scene and are all linked to each other. Such a completely connected subgraph is called a clique. Finding cliques of a given size is NP-complete, which strongly suggests that the best algorithms will be exponential.

We define two sets of nodes, the set I of instantiated nodes (the clique) and the set P of possible nodes (to extend I). Initially I is empty and P is the correspondence graph. At each step some node n from P is considered for being moved to I. There are two cases: either n is included in I or it is excluded. If it is excluded, it is removed from P. The new I is the old I and the new P is the old P minus n. If it is included, the new I is the old I plus n. To generate the new P, we take the old P minus n intersected with the links of n. This guarantees that all members of P are always linked to all members of I, and all members of I are linked to each other. Each case (n included and excluded) is generated and passed recursively to the search procedure. If at any point the union of I and P does not contain nodes that contain each junction, the conservation of vertices constraint is violated and that branch of the search terminates. Since we don't know in advance how many junctions match (invisible), we don't know the size of the clique we're looking for. It must be greater than the maximum number of junctions in each image less than the sum of the number of junctions in each image plus known invisible vertices. Figure 10 summarizes the algorithm, and figure 11 traces its execution for a few steps, following the "inclusion" branches.

3.2.3 Simplifications

Generating the correspondence graph and searching it do not have to be separate steps. One simplification is to generate nodes but not links. Instead of the conservation of adjacency constraint being applied at the beginning, to set up links, it is applied during the search. And instead of being used to calculate distances between every pair of junctions, it is only applied to adjacent junctions. When a node XX' is moved from P to I during the search, we consider all nodes YY' in P. There are three cases.

1. X not adjacent to Y and X' not adjacent to Y'. There is no information from adjacencies, so we retain node YY'.

2. X adjacent to Y, X' not adjacent to Y' (or, equivalently, X' and Y' adjacent but not X,and Y). The conservation of adjacency constraint is violated, so we delete YY'.

3. X adjacent to Y, and X' adjacent to Y'. The constraint is satisfied, YY' is retained.

```
Procedure include-node (P, I, J)
  n = first node in P
  P = (P - n) ∩ links of n
  I = I + n
  search (P, I, J)

Procedure exclude-node (P, I, J)
  n = first node in P
  P = P - n
  search (P, I, J)

Procedure search (P, I, J)
  for each junction j in J
    if j not found in P ∪ I
      return (fail)
  if P empty then
    print I
    return (success)
  include-node (P, I, J)
  exclude-node (P, I, J)

Procedure start-search
  P = correspondence graph
  I = empty
  J = junctions from image 1 ∪ junctions from image 2
  search (P, I, J)

In each procedure, the declarations are
  set-of-nodes P
  set-of-nodes I
  set-of-junctions J
  node n
  junction j
```

Figure 10: Summary of Search Algorithm

This algorithm has the same effect as the previous algorithm, since a sequence of adjacencies contains the same information as links in the correspondence graph. There is a savings in space and in preprocessing time, since no links have to be created. And during the search, most nodes will not have to be checked since neither of their junctions are adjacent to junctions in the node being moved

Start:	I = { } P = entire correspondence graph	Links from Node AL BM BT BS CU C inv DO DV D inv EN EO EP ER E inv FO FV F inv GN GR G inv HM HT HS IN IR I inv JQ J inv KP K inv (inv) to all except L
	New I = AL New P = BM BT BS CU C inv DO DV D inv EN EO EP ER E inv FO FV F inv GN GR G inv HM HT HS IN IR I inv JQ J inv KP K inv (inv) to all except L	Links from node BT AL AR CU DM DV DS EN EP E inv FO F inv GN GP G inv HM HV HS IL IR JQ KN KP K inv (inv) to MNOPQSV
	New I = AL BT New P = CU DV EN EP E inv FO F inv GN G inv HM HS IR JQ KP K inv (inv) to MNOPQSV	Links from node CU AL AN AP AR BT BV CU DT DV EL EN EP ER FM FO FS G inv HM HO HS IL IN IP IR JQ KL KN KP KR (inv) to MOS
	New I = AL BT CU	

Figure 11: First Steps of Search

into I. At the same time, however, fewer nodes are deleted at each step, so there are more left to be examined at the next step. Which method is more efficient probably depends on the implementation.

3.2.4 Invisible Vertices

In either of these cases invisible vertices must be handled as a special case. Normally, a vertex is allowed to have only one match. For the special symbol (invisible), however, more than one match must be allowed because there may be more than one invisible vertex. On the other hand, a visible vertex only has a limited number of invisible neighbors. There is a special check that, if X matches X' and X has some number i of invisible neighbors, only i of the neighbors of X' may match (invisible). Without this check we would generate matches such as A matches L and everything else matches (invisible). Yet we know that, given A matching L, one of L's neighbors must match B, one must match H, and only one of L's neighbors can match (invisible).

3.3 Comments and Extensions

There may be more than one possible complete, consistent match for a pair of objects. One cube, for example, can match another in 24 legitimate ways. One vertex can match any of eight others. Having fixed that, the cube can be rotated to 3 different positions. These matches are all topologically valid.

The algorithm will also produce 24 incorrect matches for the cube, each the mirror image of a correct match. Since the only information used is connectivity, and not sidedness, any object with a plane of topological symmetry will have spurious mirror image matches. These can be eliminated using simple geometric constraints. If a match is correct, the vertices listed clockwise around a polygon in one image will match the vertices, also listed clockwise, around a polygon in the second image. If the match is a mirror image, the matching vertices will go clockwise in one image but counterclockwise in the second image. The necessary information could be supplied by quantitative data on vertex position, or interactively by the user. It is interesting to note that if the objects are complete wire frames with no occlusion, both possibilities are potentially correct since there is no clear front or back. A clockwise sequence of vertices seen from the front becomes a counter-clockwise sequence if that face is really a back face.

The constraints used make no assumptions about all parts of an object being visible. So without modification the routines can be used to match part of an object with an object template or with another partial view of the object. There are two caveats. First, incomplete images are usually harder to label with Huffman-Clowes line labels. This may cause ambiguities in the matching (see below). Second, incomplete images have more invisible vertices and more invisible edges. This allows lots of matches where only a few vertices are matched and all the rest match (invisible). These are all legitimate possibilities, and examples can be constructed in which they are the only correct matches. But if it is known in advance that most of the vertices in the incomplete image are visible in the other image, either the search can return the possibilities with the fewest invisible matches or an upper bound can be set on the number of invisible matches.

It is not always possible to assign unique Huffman-Clowes labels to an image. If more than one possible labeling exists, the correspondence process can be run separately with each possible labeling. Then incorrect labelings may have no possible match, or only the match all(invisible). So correspondence can be used to refine line labels if multiple views are available.

Trihedrality is at the basis of the assumptions about invisible lines. Extending the algorithm to deal with non-trihedral objects would be possible, but the number of complete, correct matches possible for a pair of images would increase unmanageably unless other constraints were added. Additional information could include, for example, knowing which vertices were non-trihedral, or knowing how many non-trihedral vertices there were.

If quantitative data is available, it can be used in several ways. Conceptually, the matching process could be run just using topological data, and the quantitative information used in a post-processing step to find the most likely match or matches. It would be equivalent, but probably more efficient, to include all the data from the beginning. One scenario is that vertex positions are known to within some distance, or to within so many pixels of an epipolar line. Then the possible matches are constrained, and many fewer nodes would have to be created and searched. If positions are known precisely, they can be used to check the validity of the match. Ullman [8], Aggarwal *et al.* [1, 9], Ganapathy [4], and Lawton [6], have all worked with various numbers of points visible in two or more images, and the available constraints. Asada, Yachida, and Tsuji present a partial transition table and use it with quantitative data in [2].

3.4 Examples

We have implemented and tested both the original correspondence algorithm and the simplified version. Results are identical for the two. We have implemented and tested the correspondence algorithm. Here we show the results for a few simple figures. In each case, the images were labeled (with Huffman-Clowes line labels) by hand. Each of these simple images has a plane of symmetry, so the mirror-image program was run to remove reflections from the output. All the resulting matches are topologically correct.

For the L blocks of figure 7, the program found five possible matches (see table 2). The first match is the obvious one, which can be visualized as rotating the block approximately thirty degrees about a vertical axis. The second match is the same as the first, except that it presumes that there are additional invisible vertices seen in neither view. The third match can be thought of as tipping the block onto its back, so that the horizontal part of the L becomes the vertical part and vice versa, then rotating it some 60 degrees. The last two matches have no vertices visible in both images. The difference between these two is that the last one allows for vertices that are not visible in either image.

1. AL BT CU DV EN FO G(invisible) HS IR JQ KP (invisible)M

2. AL BT CU DV EN FO G(invisible) HS IR JQ KP (invisible)M (invisible)(invisible)

3. AO BP CQ DR ES FL GM HN IV JU KT (invisible)(invisible)

4. A(invisible) B(invisible) C(invisible) D(invisible) E(invisible) F(invisible) G(invisible) H(invisible) I(invisible) J(invisible) K(invisible) (invisible)L (invisible)M (invisible)N (invisible)O (invisible)P (invisible)Q (invisible)R (invisible)S (invisible)T (invisible)U (invisible)V

5. A(invisible) B(invisible) C(invisible) D(invisible) E(invisible) F(invisible) G(invisible) H(invisible) I(invisible) J(invisible) K(invisible) (invisible)L (invisible)M (invisible)N (invisible)O (invisible)P (invisible)Q (invisible)R (invisible)S (invisible)T (invisible)U (invisible)V (invisible)(invisible)

Table 2: Matches for Figure 7

Figure 12 shows the same L-shaped blocks as figure 7, but with only part of the left-hand block visible. The additional constraint was given that there was a total of 6 invisible vertices. This eliminated matches in which one or two vertices from the partial block matched visible junctions in the right-hand image and all other vertices matched (invisible). The first two matches correspond to match 1 in table 2, and the second two to match 3. The difference between matches 1 and 2 is in the match for H. Without quantitative information, it is impossible to tell whether H matches S (as in match 2) or T (as in match 1). The same holds for matches 3 and 4.

Figure 13 shows a block with a corner cut out. It was matched against itself, with the extra constraint given to the program that there were no invisible type VII vertices. This is a strong constraint, since it forces M to match itself. The results are shown in table 4.

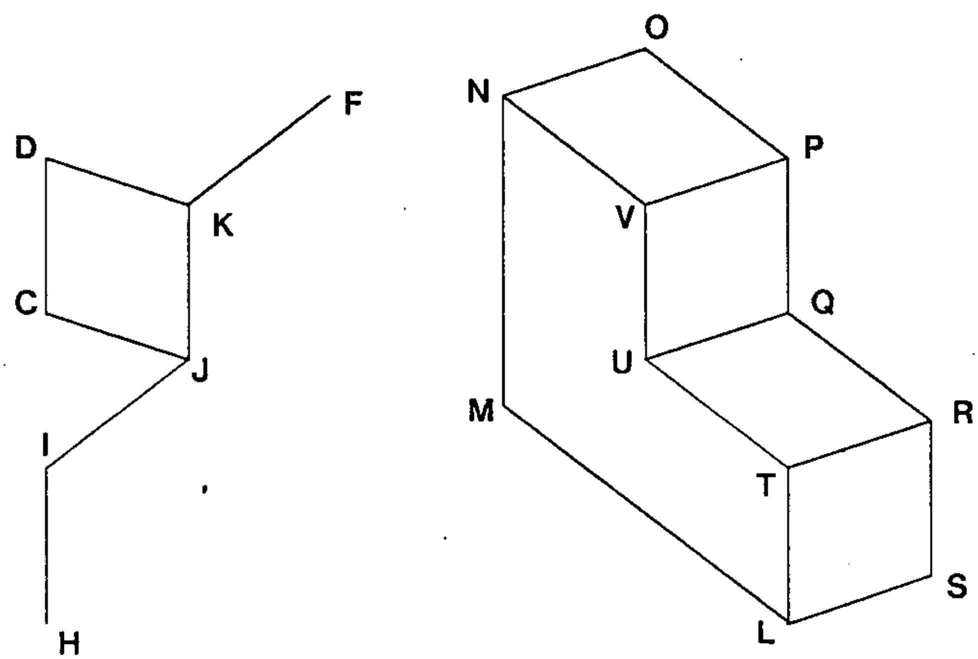


Figure 12: Part of an L Block

1. L(inv) M(inv) N(inv) OF PK QJ RI S(inv) TH UC VD (inv)(inv)

2. L(inv) M(inv) N(inv) OF PK QJ RI SH T(inv) UC VD (inv)(inv)

3. LF M(inv) N(inv) O(inv) PH QC RD S(inv) TK UJ VI (inv)(inv)

4. LF M(inv) NH O(inv) P(inv) QC RD S(inv) TK UJ VI (inv)(inv)

Table 3: Matches for Part of an L Block

Finally, in figure 14 each image contains two objects, the L block and the block with the corner cut out. The program was told that there were no more than seven invisible vertices. It generated 18 matches: the three for the cutout block, times three for the L blocks (the last two L-block matches were eliminated because of the limit on invisible vertices), times a factor of two for different interpretations of the hidden part of the cutout block in the left image. There were three incorrect mirror images for each correct match, since the L block and the cutout block each have their own plane of symmetry.

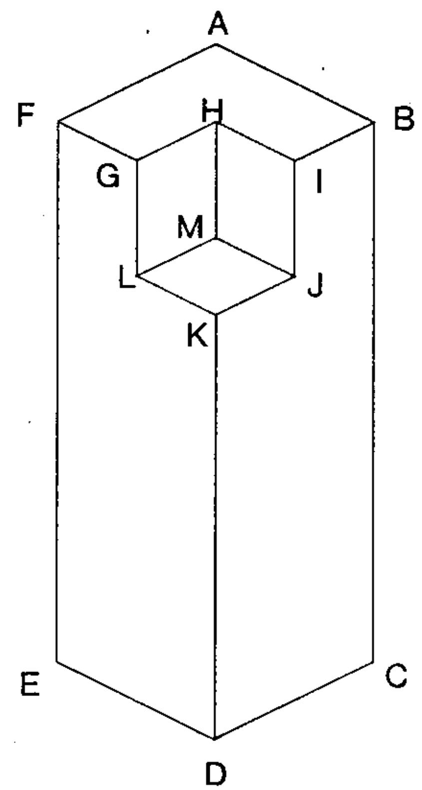


Figure 13: Block With One Corner Cut Out

1. Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm (invisible)(invisible)

2. Ac Bd Ce Df Ea Fb Gi Hj Ik Jl Kg Lh Mm (invisible)(invisible)

3. Ae Bf Ca Cb Ec Fd Gk Hl Ig Jh Ki Lj Mm (invisible)(invisible)

Table 4: Matches for Cut-Out Block

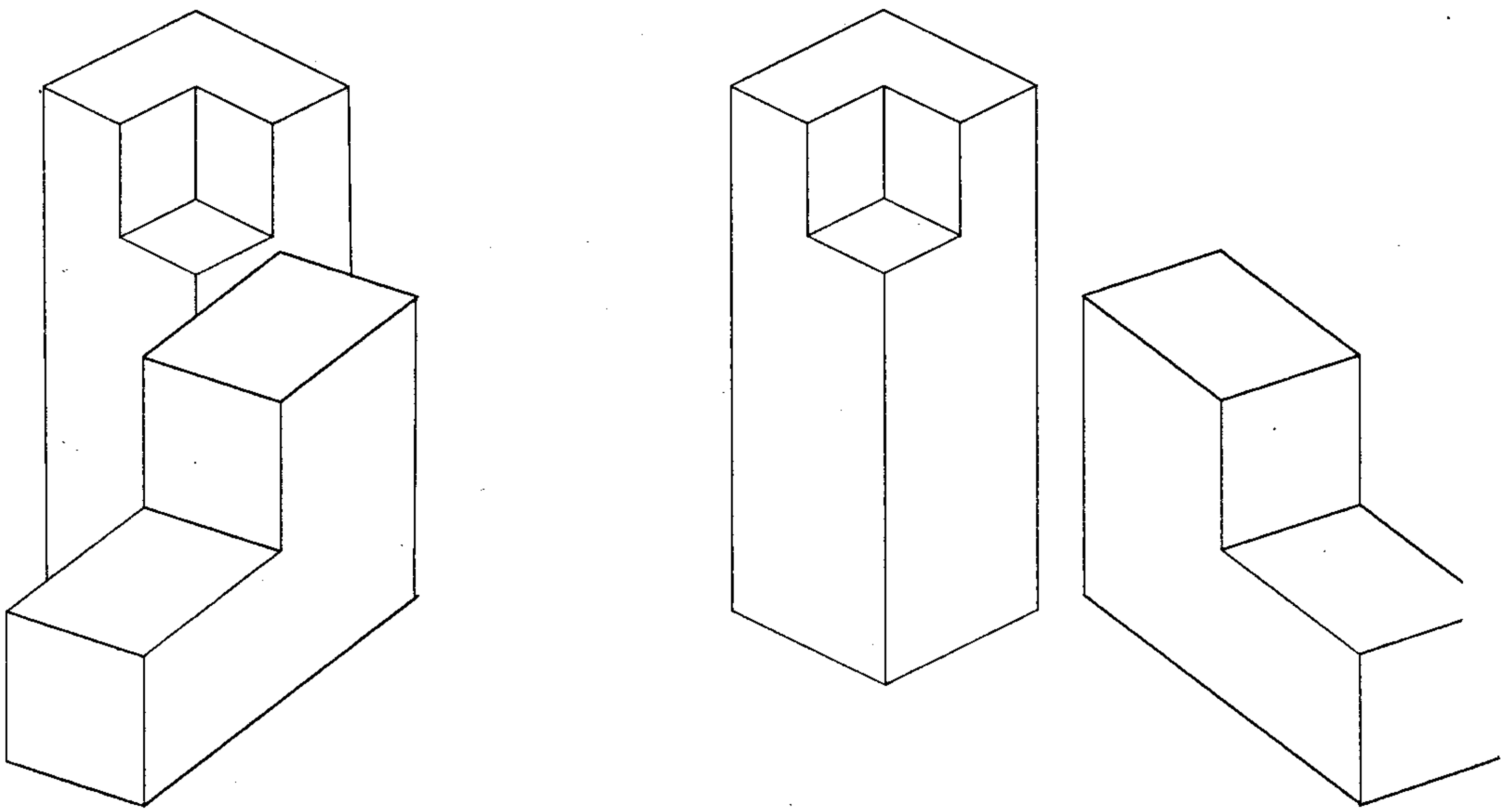


Figure 14: Two Objects

4. Summary

In this paper, we have presented some constraints that come from junction types and edges, and have shown how they can be used to help solve the correspondence problem for either stereo or motion. We have shown a way of inferring vertex type from junction appearance, even for L junctions where one of the legs is invisible. In some cases, an L junction can be correctly and uniquely labeled based on surface visibilities or location of the hidden edge.

We have developed transition tables that show the allowable changes in junction appearance as the viewpoint moves. A matching algorithm has been presented which uses three constraints: conservation of vertices, conservation of vertex type, and conservation of adjacency. The algorithm also uses some simple reasoning about hidden vertices. The matching algorithm does not in general give a unique match. Rather, it gives all the matches that satisfy all the topological constraints. A practical matcher could incorporate the constraints developed here as well as quantitative data and other knowledge.

4.1 Acknowledgements

Many people, especially members of the CMU Image Understanding group, provided helpful comments and discussions. Thanks to Marty Herman, Richard Korf, Leslie Thorpe, and Jon Webb. Special thanks to Takeo Kanade, the head of the Image Understanding group, for encouragement and advice.

References

[1] J. K. Aggarwal and W. N. Martin.
Analyzing Dynamic Scenes Containing Multiple Moving Objects.
In T. S. Huang (editor), *Image Sequence Analysis*, . Springer-Verlag, 1981.

[2] M. Asada, M. Yachida, and S. Tsuji.
Three Dimensional Motion Interpretation for the Sequence of Line Drawings.
In *Fifth International Joint Conference on Pattern Recognition*. IEEE, 1980.

[3] M. B. Clowes.
On Seeing Things.
Artificial Intelligence 2, 1971.

[4] Sundaram Ganapathy.
Reconstruction of Scenes Containing Polyhedra From Stereo Pair of Views.
PhD thesis, Stanford University, January, 1976.

[5] D. A. Huffman.
Impossible Objects as Nonsense Sentences.
In B. Meltzer and D. Michie (editor), *Machine Intelligence 6*, . American Elsevier, 1971.

[6] Daryl T. Lawton.
Constraint-Based Inference From Image Motion.
In *Proceedings of the AAAI*. AAAI, 1980.

[7] Sidney Liebes.
Geometric Constraints for Interpreting Images of Common Structural Elements: Sothogonal Trihedral Vertices.
In *ARPA Image Understanding Workshop*. April, 1981.

[8] Shimon Ullman.
The MIT Press Series in Artificial Intelligence: *The Interpretation of Visual Motion*.
The MIT Press, Cambridge, Massachusetts, and London, England, 1979.

[9] Jon A. Webb and J. K. Aggarwal.
Structure from Motion of Rigid and Jointed Objects.
In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*. 1981.