* For the reference see the last page

Inference of A 3-D Object From A Random Partial 2-D Projection

Pamela Hsu and Evangelos Triantaphyllou

Kansas State University

Abstract

The problem we examine here is how to infer the entire topology of an unknown 3-D object from a random partial 2-D projection. In our research we combine the model graph [Wong and Fu, 1985] with the AHR graph to create, what we call, the Adjacency Graph (or AG graph). The AG graph exhibits many interesting properties. These properties relate the way the nodes and branches are connected in complete and incomplete AG graphs. A complete AG graph describes the entire topology of a 3-D object while an incomplete AG graph describes a partial topology of that object. Therefore, the problem we solve here is how to infer a complete AG graph from an incomplete AG graph. The proposed approach is demonstrated by an example taken from the literature. Furthermore, this approach is very efficient on both time and space requirements.

Introduction

In object recognition, graphs are widely used to represent entire or partial views of 3-D objects. The advantage of using a graph representation is that the topology of an object can be described in a simple and efficient way. Nodes and branches of a graph can correspond to different object components. Wong and Fu [1985] recommended nodes to represent object vertices and branches to represent object edges. In other work, however, nodes correspond to object faces and branches correspond to object edges (e.g., [Akinniyi, Wong, and Stacey, 1986] and [Joshi and Chang, 1990]).

In general, very often a graph itself cannot capture all pertinent topological information required for a recognition process. For this reason, Wong and Fu [1985] proposed "allowable junction sets" and "neighbor-constraint-sets" to help describe the required geometry of a 3-D object. These sets are associated with a graph which represents a 3-D object and is called a model graph. The junction types were introduced by Chakravarty [1979]. In [Wong and Fu, 1985] the edges of a 3-D object are allowed to be curved or straight segments. Some other graph representation schemes indicate the type of the angle between two adjacent faces. Attributes are thus assigned to every branch to de-note the corresponding angle type (see, for instance, [Wong and Lu, 1983] and [Joshi and Chang, 1988]).

The typical object recognition problem is described following. Given a random view of an object and a number of complete representations of 3-D objects in a data base, a recognition system tries to identify the new object as one of the objects represented in the data-base. ***In this paper, an inference approach is proposed to infer the entire geometry of a 3-D object from a single random view.*** Given a view of an object the proposed approach can infer the *rest* part of that object, which is ***invisible*** from the given view point. In the present research, we deal only with inferring the geometry of an object. The dimensions of a 3-D object are not considered here.

In this approach, a new graph representation scheme of a 3-D object is presented and is called the Adjacency Graph (or AG graph). AG graphs are based on the junction sets [Wong and Fu, 1985], the Attributed Adjacent Graph [Joshi and Chang, 1988], and the Attribute Hyper-graph Representation [Lu and Wong, 1988]. The difference between the previous work and AG graphs will be discussed in the following sections. The pro-posed approach is also illustrated by inferring a 3-D object from an example given in the literature.

The Concept of the Adjacency Graph

The Adjacency Graph (or AG graph) is introduced to capture the pertinent topological information for the object inference problem. In this paper, AG graphs are classified into two categories; **complete** and **incomplete** AG graphs. A complete AG graph describes the ***entire*** topology of a 3-D object. In a complete AG graph, every object face and edge are represented by nodes and branches, respectively. Contrary to a complete AG graph, an incomplete AG graph describes ***only*** the topology of an object as it is observed from a given view point. This indicates that *only* object faces and their common edges *visible* from a given view are represented in an *incomplete* AG graph.

An object face can be denoted by a sequence of object vertices. Each vertex can be denoted by a junction. Only two junction types are used in this paper; the **T-type** and the **non-T-type**. These junction types are based on those defined in [Wong and Fu, 1985] and [Chakravarty, 1979]. A T-type junction indicates the occurrence of a vision occlusion. More discussion about vision occlusions is presented later. A non-T-type junction denotes general object vertices.

The explicit difference between Wong and Fu's model graph and an AG graph is as follows. In AG graphs a branch does *not only* indicate the adjacency of two object faces *but also* represents a **common edge** between these two faces. Thus, the total number of branches connecting two nodes is equal to the number of common edges between the corresponding faces. For instance, if there are three common object edges between two adjacent faces, then there are three branches connecting the two corresponding nodes in the AG graph. However, the number of common edges between two adjacent object faces is not indicated in a model graph [Wong and Fu, 1985]. In a model graph a branch between two nodes simply expresses that the corresponding object faces are adjacent.

In graph representations of 3-D objects, the connections of nodes are widely used to indicate the adjacency among object faces (see for instance, [De Floriani, 1986] and [Lu and Wong, 1988]). The difference between AG graphs and the previous work is the ***property set*** associated with every AG graph. From a property set, the detailed geometrical information of an object face can be derived.

Basically, components of AG graphs denote only the existence of object faces and edges. An AG graph itself does not offer the information about any vision occlusion. Vision occlusions are indicated in the property set, which is associated with an AG graph. This happens because the junction types in a property set depend on the view point.

A complete AG graph represents *all faces and edges* of a 3-D object as nodes and branches, respectively. The issue of vision occlusions is not considered. Therefore, all the junctions in a property set are of non-T-type. In the case of incomplete AG graphs, however, only object faces and their common edges **visible** from a given view are represented in these graphs. Thus, an incomplete AG graph may alter when the view point changes. Formal definitions of complete AG graphs and their property sets follow.

Definition 1:
*A **complete** AG graph of a 3-D object is a graph $G = \{N,B\}$, where:*
- *$N = \{N_1, N_2, N_3, ..., N_n\}$ is a set of nodes. Node N_i corresponds to the object face i.*
- *$B = \{B_1, B_2, B_3, ..., B_n\}$ is a set of ordered branch lists. Branch list B_i corresponds to node $N_i \in N$. $B_i = \{N_j \mid N_j \in N\}$ is a list of ordered nodes, where $N_j \in B_i$ if and only if object faces i and j have a common edge.*

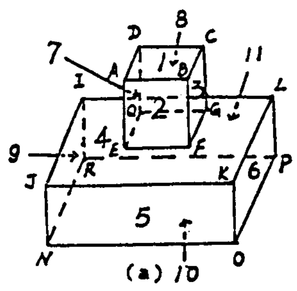
Definition 2:
A property set A of a complete AG graph is the set of ordered junction sets A_i, where $A_i = \{(V_j, \sim T) \mid V_j \in V\}$ and corresponds to node $N_i \in N$. V is the set of object vertices and $\sim T$ denotes a non-T-type junction. $(V_j, \sim T) \in A_i$ if and only if V_j is a vertex of object face i and is a non-T-type junction.

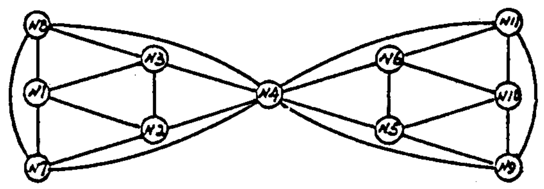
Figure 1 depicts a 3-D object taken from [Akkinniyi, Wong, and Stacey, 1986] and the corresponding complete AG graph with its property set. The adjacencies among faces are denoted as the branches among nodes in the AG graph. For example, face 1 in Figure 1(a) is adjacent to faces 2, 3, 8, and 7. Therefore, in Figure 1(b) node N_1 is connected with nodes N_2, N_3, N_8, and N_7. These adjacencies are described in the corresponding branch list $B_1 = \{N_2, N_3, N_8, N_7\}$. Consider the corresponding property set in Figure 1(b). The vertices which define face 1 and their junction types are represented in terms of ordered pairs in the junction set A_1. Namely, $A_1 = \{(V_a, \sim T), (V_b, \sim T), (V_c, \sim T), (V_d, \sim T)\}$.

Incomplete AG graphs and their corresponding property sets are defined as follows.

Definition 3:
*An **incomplete** AG graph of a 3-D object is a graph $G' = (N', B')$, where:*
- *$N' = \{N_i \mid N_i \in N\}$ is a set of nodes. Node N_i corresponds to a **visible** object face.*
- *$B' = \{B_i \mid B_i \in B\}$ is a set of ordered branch lists. Branch list B_i corresponds to node $N_i \in N'$. $B_i = \{N_j \mid N_j \in N'\}$ is a list of ordered nodes, where $N_j \in B_i$ if and only if visible object faces i and j have a common edge.*

Definition 4:
*A property set A' of an **incomplete** AG graph is the set of ordered junction sets A_i, where $A_i = \{(V_j, J) \mid V_j \in V, J \in \{T, \sim T\}\}$ and corresponds to node N_i $(N_i \in N')$. V is the set of object vertices. J is the set of allowable junction types, where T indicates a T-type junction and $\sim T$ denotes a non-T-type junction. $(V_j, J) \in A_i$ if and only if V_j is a vertex of the **visible** object face i and has junction type J.*





$G = \{N, B\}$

where:

$N = \{N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, N_9, N_{10}, N_{11}\}$

$B = \{B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}, B_{11}\}$
$= \{\{N_2, N_3, N_8, N_7\}, \{N_1, N_3, N_4, N_7\},$
$\{N_1, N_2, N_4, N_8\}, \{N_2, N_3, N_7, N_8, N_5, N_6, N_{11}, N_9\},$
$\{N_6, N_4, N_9, N_{10}\}, \{N_5, N_4, N_{11}, N_{10}\},$
$\{N_1, N_2, N_4, N_8\}, \{N_1, N_3, N_4, N_7\},$
$\{N_5, N_{10}, N_{11}, N_4\}, \{N_5, N_6, N_{11}, N_9\},$
$\{N_6, N_4, N_9, N_{10}\}\}$

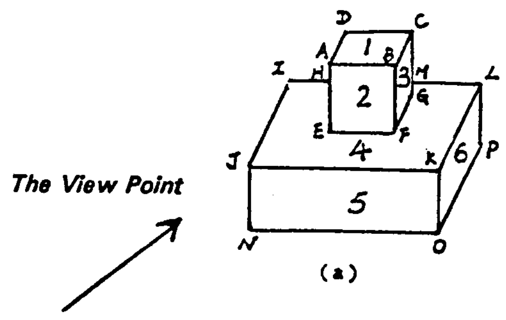
The property set is:

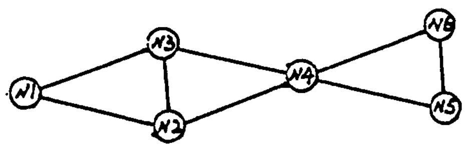
$A = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10}, A_{11}\}$
$= \{\{(V_a, \sim T), (V_b, \sim T), (V_c, \sim T), (V_d, \sim T)\},$
$\{(V_a, \sim T), (V_e, \sim T), (V_f, \sim T), (V_b, \sim T)\},$
$\{(V_b, \sim T), (V_f, \sim T), (V_g, \sim T), (V_c, \sim T)\},$
$\{(V_i, \sim T), (V_j, \sim T), (V_k, \sim T), (V_l, \sim T), (V_q, \sim T),$
$(V_g, \sim T), (V_f, \sim T), (V_e, \sim T)\},$
$\{(V_j, \sim T), (V_n, \sim T), (V_o, \sim T), (V_k, \sim T)\},$
$\{(V_k, \sim T), (V_o, \sim T), (V_p, \sim T), (V_l, \sim T)\},$
$\{(V_a, \sim T), (V_d, \sim T), (V_q, \sim T), (V_e, \sim T)\},$
$\{(V_d, \sim T), (V_c, \sim T), (V_g, \sim T), (V_q, \sim T)\},$
$\{(V_j, \sim T), (V_i, \sim T), (V_r, \sim T), (V_n, \sim T)\},$
$\{(V_n, \sim T), (V_o, \sim T), (V_p, \sim T), (V_r, \sim T)\},$
$\{(V_r, \sim T), (V_p, \sim T), (V_l, \sim T), (V_i, \sim T)\}\}$

(b)

Figure 1. A Complete AG Graph of An Object taken from [Akkinniyi, Wong, and Stacey, 1986].

Figure 2(a) is a random view of the object shown in Figure 1(a). The corresponding incomplete AG graph and the associated property set are given in Figure 2(b). The following section discusses the effect of a vision occlusion on identifying object vertices.





$G' = \{N', B'\}$

where:

$N' = \{N_1, N_2, N_3, N_4, N_5, N_6\}$

$B' = \{B_1, B_2, B_3, B_4, B_5, B_6\}$
$= \{\{N_2, N_3\}, \{N_1, N_3, N_4\}, \{N_1, N_2, N_4\},$
$\{N_2, N_3, N_5, N_6\}, \{N_4, N_6\}, \{N_4, N_5\}\}$

The property set is:

$A' = \{A_1, A_2, A_3, A_4, A_5, A_6\}$
$= \{\{(V_a, \sim T), (V_b, \sim T), (V_c, \sim T), (V_d, \sim T)\},$
$\{(V_a, \sim T), (V_e, \sim T), (V_f, \sim T), (V_b, \sim T)\},$
$\{(V_b, \sim T), (V_f, \sim T), (V_g, \sim T), (V_c, \sim T)\},$
$\{(V_h, T), (V_i, \sim T), (V_j, \sim T), (V_k, \sim T), (V_l, \sim T),$
$(V_m, T), (V_g, \sim T), (V_f, \sim T), (V_e, \sim T)\},$
$\{(V_j, \sim T), (V_n, \sim T), (V_o, \sim T), (V_k, \sim T)\},$
$\{(V_k, \sim T), (V_o, \sim T), (V_p, \sim T), (V_l, \sim T)\}\}$

(b)

Figure 2. An Incomplete AG Graph of The Object Shown in Figure 1(a) Observed From A Given View Point.

Vision Occlusions

Very often vision occlusions result in recognition ambiguities. In general, there are two classes of vision occlusions: ***cross occlusions*** and ***alignment occlusions***. The difference between these two classes of occlusions is the junction types. A cross occlusion always forms a T-type junction such as vertex E shown in Figure 3(a). However, an alignment occlusion happens only when two edges are overlapped without forming any T-type junction. For example, edges DC and JS are overlapped in Figure 3(b). In this paper, we assume that ***only*** cross occlusions occur.

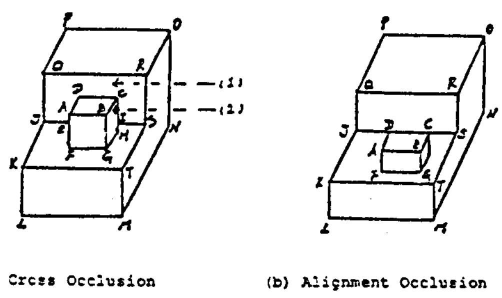

(a) Cross Occlusion (b) Alignment Occlusion

Figure 3. Examples of Vision Occlusions.

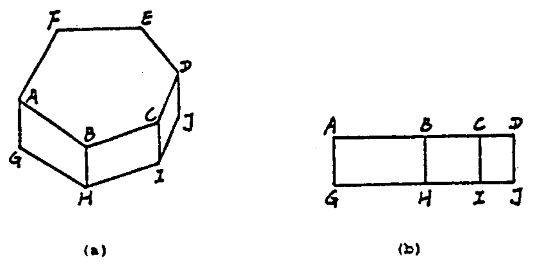


Figure 4. Examples of Virtual T-type Vertices.

A vertex with a T-type junction is a **virtual** vertex. That happens because this junction exists only when a vision occlusion occurs. Therefore, a T-type junction cannot correspond to an actual object vertex. In this paper a vertex is assumed to be formed by exactly three **adjacent** object faces. As the letter T indicates, a T-type junction is formed by three object faces visible from a given view. Since a T-type vertex cannot correspond to an actual object vertex, this vertex thus may not be able to indicate the adjacency among faces, which define this vertex. The adjacency among faces which define a T-type vertex will be discussed later.

In some cases, an actual object vertex may correspond to a T-type junction when it is observed from a certain view point. For instance, in Figure 4(a) the junction types of object vertices B, C, H and I are non-T-type junctions. However, when a side view of
this object is taken, vertices B, C, H, and I become T-type junctions (see, Figure 4(b)).

In order to tell an object vertex from a T-type junction (i.e., a virtual vertex), a different view point thus needs to be considered. In general, if a T-type vertex is an actual object vertex, then its junction will change to a non-T-type junction when the view point moves slightly above or below its current position. In this paper it is assumed that the situation shown in Figure 4(b) never occurs.

Similar to a virtual vertex resulted from a T-type junction, ***virtual edges*** can be formed too. The **top** edge of a T-type vertex can result in a **virtual** edge for any face which contains the T-type vertex. For example, edge CH
edge of vertex I in Figure 3(a). Apparently, edge CH does not only denote an object edge but also forms two virtual edges. Namely, edges CI and IH. These edges are **virtual** edges of the two faces, which contain the T-type junction I. Since virtual edges denote object faces only when an occlusion occurs, then these edges cannot correspond to actual object edges. There-fore, a virtual edge cannot indicate the adjacency between two faces which contain this virtual edge. For example, virtual edge CI does not indicate that faces (1) and (2) are adjacent in Figure 3(a).

Properties of AG graphs

The proposed inference process is a methodology for inferring invisible faces of an object and the adjacency among object faces. As discussed in Section 2, the adjacency of two faces is denoted by a common edge. This implies that the total number of faces adjacent to a visible face can be inferred from the number of object edges, which define this face. For instance, face 3 in Figure 5(a) consists of four object edges. These are edges HA, AJ, JI, and IH. Therefore, the total number of faces adjacent to face 3 is four.

Consider the case in which no vision occlusions occur in an object face. Any edge of this object face is a common edge of two adjacent faces. Therefore, **every** edge of the visible face directly indicates **one** adjacent face. This indicates that the number of faces adjacent to a visible face is ***less than or equal to*** the number of edges which define that ***visible*** face.

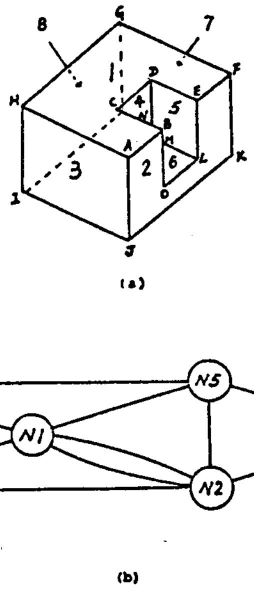


Figure 5. Illustrations of The Adjacency among Faces.

In the previous paragraph, it was mentioned that the number of faces adjacent to a face may be less than the number of edges which define that face. This happens when two adjacent faces have more than one common edges between them. For example, face 1 in Figure 5(a) has two common edges with face 2 (i.e., edges AB and EF). The total number of edges, which define face 1, is eight but the number of faces adjacent to face 1 is seven. However, this adjacency can be shown in an AG graph without distorting the topology of the object. Figure 5(b) presents an incomplete AG graph of the object given in Figure 5(a). The *two common edges* between faces 1 and 2 are denoted as the *two branches* connecting nodes N_1 and N_2.

In case an occlusion occurs, a visible face may be defined by a number of edges from which at least one edge is a virtual edge. Recall that a virtual edge cannot indicate the adjacency between two faces which contain this virtual edge. Therefore, if a **visible** object face contains a virtual edge, then the total number of faces adjacent to this face cannot be determined directly from the number of its edges.

As it was discussed previously, the total number of faces adjacent to a visible face can be determined from the number of its edges if no occlusion occurs. However, in an incomplete AG graph **only** the common edge of two **visible** faces is represented as a branch. In other words, **not every** visible edge is represented in an incomplete AG graph. For instance, edge HI in Figure 5(a) is a common edge of visible face 3 and **invisible** face 8 but is not represented in the graph. Therefore, in an incomplete AG graph the number of branches emerging from a node may be ***less than or equal to*** the number of edges which define the corresponding visible face.

The problem presented here is how to determine the total number of faces adjacent to a visible face. In an object face, we can observe that the number of **edges** of a face is equal to the number of its **vertices**. Both these edges and vertices define the same object face. This implies that the number of faces adjacent to a visible face can be also determined from its object vertices. Therefore, if a visible face *does not consist* of any T-type vertex, then the total number of faces adjacent to that face is less than or equal to the number of vertices, which define that face. The following property summarizes the above observation.

Property 1:
If junction set A_i does not contain a T-type vertex, then the total number of nodes connecting node N_i is less than or equal to the total number of vertices given in junction set A_i.

Applying Property 1, the total number of nodes connecting node N_i is **at most** equal to the number of vertices which define object face i. Consider that in an incomplete AG graph the number of nodes connecting node N_i is equal to the number of nodes in the corresponding branch list B_i. Therefore, the total number of ***invisible*** faces (denoted as M_i) adjacent to face i can be determined by Property 2. Here, the nodes which correspond to invisible faces are called ***new nodes***, since these nodes are not represented in an incomplete AG graph and are created during the inference process.

Property 2:
If junction set A_i does not contain a T-type vertex, then the number of new nodes (currently invisible faces) connecting node N_i is less than or equal to the quantity M_i, where $M_i = |A_i| - |B_i|$.

A common edge of two faces indicates the adjacency between the two faces and consists of **exactly** two vertices. Therefore, the adjacency between two faces can also be inferred from two vertices, which define the common edge of those two faces. We call these two vertices as *common* vertices of the two adjacent faces. In other words, if two faces have two common vertices, then these faces are adjacent. Since we assumed that an object vertex is formed by three adjacent faces, similarly, we can obtain the following observation. If **three** faces have a **common** vertex, then the three faces are **adjacent** to each other.

As it was defined in Section 2, in AG graphs each junction set in the property set corresponds to a node (or equivalently, to an object face). We can observe that vertices in two *consecutive* pairs of a junction set indicate an edge of that face. Therefore, if two junction sets have two consecutive pairs *in common*, then the two corresponding object faces have a common edge. Furthermore, it follows that these faces are adjacent.
In order to avoid redundancy, the first and the last pairs of a junction set are considered to be consecutive pairs. Therefore, vertices in the first and the last pairs indicate an edge of a face. For instance, in Figure 2(a), edge AB of face 2 is denoted by the first and the last pairs in the junction set A_2 given in Figure 2(b). These pairs are $(V_a, \sim T)$ and $(V_b, \sim T)$, respectively. The above observations are summarized in the following property.

Property 3:
Nodes N_i and N_j are connected if and only if the corresponding junction sets A_i and A_j have two consecutive pairs in common.

As discussed at the beginning of Section 3, a T-type junction vertex indicates a virtual object vertex. Furthermore, a vertex with a T-type junction may not be able to indicate the adjacency among the three faces which define this vertex. However, the following property is applicable ***only*** when a vertex is a non-T-type junction. That is, the vertex is an actual object vertex.

Property 4:
Nodes N_i, N_j, and N_k are connected if and only if the corresponding junction sets A_i, A_j, and A_k have one common pair.

For example, the graph which was shown in Figure 2(b) depicts that node N_1 is connected with nodes N_2 and N_3. In the property set, we can observe that junction sets A_1 and A_2 have two common consecutive pairs;i.e., $(V_a, \sim T)$ and $(V_b, \sim T)$. Furthermore, consider that the adjacent pairs $(V_b, \sim T)$ and $(V_c, \sim T)$ are also listed in both junction sets A_1 and A_3. This example illustrates how the adjacency of two faces can be indicated in the two corresponding junction sets. Since all visible vertices have been denoted in an incomplete AG graph, this implies that the adjacency among visible faces holds and should not be altered during the inference process. Therefore, the following property is true.

Property 5:
If node N_i does not connect node N_j in an incomplete AG graph, then node N_i and node N_j cannot be connected in the corresponding complete AG graph.

The Algorithm for Inferring A Complete AG Graph

The five properties presented in the previous section are used in the inference process. The main algorithm of the inference process calls two procedures. The detailed inference process is illustrated with an example in next section.

Algorithm Infer_AG_Graph:

```
INPUT: An incomplete AG graph $G' = \{N',B'\}$ and its
       property set A', where:
       $N' = \{N_1, N_2, N_3, ..., N_n\}$; set of nodes.
       $B' = \{B_1, B_2, B_3, ..., B_n\}$; set of branch lists.
       $A' = \{A_1, A_2, A_3, ..., A_n\}$; property set of G'.

OUTPUT: A complete AG graph $G=\{N,B\}$ and its property
       set A.

BEGIN /* Infer_AG_Graph */

    /* Initialization */
(1)  N <— N'; B <— B'; A <— A'
     Set all nodes in N as "UNMARKED"

(2)  Initialize set C such that set C contain all common
       edges in each junction set $A_i$ ($A_i \in A$)

(3)  REPEAT
(4)    Use Property 2 to find node $N_{next}$ and the quantity:
         $M_{next} = |A_{next}| - |B_{next}|$
       IF ($M_{next} > 0$) THEN
(5)        Create $M_{next}$ new nodes and update node set N

     /* Use PROCEDURE Init_Branch_Lists_&_Junction_Sets */
(6)        Initialize branch lists and junction sets for each of
             the $M_{next}$ new nodes.
       ENDIF

     /* Use PROCEDURE Identify_&_Update. A new vertex
        may need to be created for an invisible common edge*/
(7)    Identify common edges and common vertices in any
         of the previously new junction sets
       Update appropriate branch lists, junction sets and
         common edges in set C

(8)    Set node $N_{next}$ as "MARKED"

     UNTIL (all nodes $N_i$'s in N without a T-type vertex
            have been marked)

(9)  Use Properties 3 and 4 to find dummy new vertices
     Update proper sets B, A and C

END /* End of Infer_AG_Graph */
```

```
PROCEDURE Init_Branch_Lists_&_Junction_Sets
/* Create $M_{next}$ branch lists and junction sets */

INPUT: Next node $N_{next}$;
       $M_{next} = \min\{M\}$, where
       $M = \{M_i \mid M_i = |A_i| - |B_i|\}$; $A_i \in A$ and $B_i \in B$.

OUTPUT: $B = \{B_1, B_2, B_3, ..., B_n, ..., B_{Mnext}\}$;
        $A = \{A_1, A_2, A_3, ..., A_n, ..., A_{Mnext}\}$.

BEGIN
(1)  Create $M_{next}$ branch lists and junction sets. Each
       new branch list contains $N_{next}$. Each new junction
       set is initially empty

(2)  REPEAT
(3)    Find any two consecutive pairs in $A_{next}$(i.e., the
         junction set of node $N_{next}$), which does not indicate
         a common edge

(4)    Assign the previous two pairs of $A_{next}$ to any of
         $M_{next}$ new junction sets which are still empty. Note
         that now this two pairs will denote a common edge
         since they are present in two junction set

     UNTIL (all $M_{next}$ new junction sets are not empty)

(5)  Update branch set B by adding $M_{next}$ new branch lists
       to branch set B and add $M_{next}$ new nodes to branch
       list $B_{next}$.
```

```
(6)  Update property set A by adding $M_{next}$ new junction
       sets to property set A

END /* End of Init_Branch_Lists_&_Junction_Sets */
```

PROCEDURE Identify_&_Update
/* Identify common edges and common vertices in any of the M_{next} new junction sets. Update proper branch lists in B and junction sets in A */

INPUT: B is the branch set which includes M_{next} new branch lists. A is the property set which includes M_{next} new junction sets.

OUTPUT: Updated sets B and A.

```
BEGIN
    REPEAT
(1)    Find three junction sets such that they have a
         common vertex $V_{common}$. Vertex $V_{common}$ is in
         $M_{next}$ new junction sets and is not an intersection
         of two common edges

       IF (An existing vertex is adjacent to vertex $V_{common}$)
          THEN
              Denote this vertex as $V_{new\_c}$

          ELSE
(3)           Create a new vertex. Denote this vertex as
                 $V_{new\_c}$
(4)           Add vertex $V_{new\_c}$ to proper junction sets such
                 that vertex $V_{new\_c}$ is adjacent to vertex
                 $V_{common}$ in those junction sets
       ENDIF

(5)    Update proper junction sets

(6)    Set vertices $V_{new\_c}$ and $V_{common}$ in proper junction
         sets for indicating a common edge

(7)    Update proper branch lists
    UNTIL (there are exactly three common edges
         indicated in every new junction sets)

END /*End of Identify_&_Update*/
```

Procedure **Init_Branch_Lists_&_ Junction_Sets** takes $O(n^2)$ time. To see this observe that the main loop is repeated O(n) times and step (3) takes also O(n) time. Similarly, procedure **Identify_&_Update** takes $O(n^2)$ time. This happens because the main loop is repeated O(n) times and step (6) takes O(n) time too. The main loop in the **Infer_AG_Graph** algorithm is repeated O(n) times. From the times of the previous procedures it follows that each iteration takes $O(n^2)$ time. Therefore, the main algorithm takes $O(n^3)$ time.

An Example of the Inference Process

A demonstration of the proposed algorithm is illustrated by inferring an object presented in [Akinniyi, Wong, and Stacey, 1986]. This object is depicted in Figure 2(a). A corresponding incomplete AG graph is given in Figure 2(b). The inference process is described as follows.

The Inference Process:

Step 1: $N = \{N_i \mid N_i \in N'\}$; $B = \{B_i \mid B_i \in B'\}$;
$A = \{A_i \mid A_i \in A'\}$.
Node N_i's are "UNMARKED", where $N_i \in N$.

Step 2: C = {AB, BC, BF, EF, FG, JK, KL, KO}.

Step 3:
Iteration 1
Step 4: $N_{next} = N_2$; $M_{next} = 1$.

Step 5: New node: N_{21};
$N <- N \cup \{N_{21}\}$.

Step 6: $B_{21} = \{N_2\}$; $A_{21} = \{(V_a, \sim T), (V_e, \sim T)\}$;
$B_2 <- B_2 \cup \{N_{21}\}$; $B <- B \cup \{B_{21}\}$; $A <- A \cup \{A_{21}\}$.

Step 7: New vertex V_q;
$A_4 = \{(V_h, T), (V_i, \sim T), (V_j, \sim T), (V_k, \sim T), (V_l, \sim T), (V_m, T), (V_g, \sim T), (V_f, \sim T), (V_e, \sim T), (V_q, \sim T)\}$;
$B_{21} <- B_{21} \cup \{N_1, N_4\}$;
$A_{21} = \{(V_d, \sim T), (V_a, \sim T), (V_e, \sim T), (V_q, \sim T)\}$;
$B_1 <- B_1 \cup \{N_{21}\}$; $B_4 <- B_4 \cup \{N_{21}\}$;
$C <- C \cup$ {AE, AD, EQ}.

Step 8: Set node N_2 as "MARKED".
Note that the current AG graph is shown in Figure 6(a).

Iteration 2
Step 4: $N_{next} = N_1$; $M_{next} = 1$.

Step 5: New node: N_{11};
$N <- N \cup \{N_{11}\}$.

Step 6: $B_{11} = \{N_1\}$;
$A_{11} = \{(V_c, \sim T), (V_d, \sim T)\}$;
$B_1 <- B_1 \cup \{N_{21}\}$; $B <- B \cup \{B_{11}\}$; $A <- A \cup \{A_{11}\}$.

Step 7: $B_{11} <- B_{11} \cup \{N_{21}, N_3\}$;
$A_{11} = \{(V_g, \sim T), (V_c, \sim T), (V_d, \sim T), (V_q, \sim T)\}$;
$B_{21} <- B_{21} \cup \{N_{11}\}$; $B_3 <- B_3 \cup \{N_{11}\}$;
$C <- C \cup$ {DC, CG, DQ}.

Step 8: Set node N_1 as "MARKED".
Note that the current AG graph is shown in Figure 6(b).

Iteration 3
Step 4: $N_{next} = N_3$; $M_{next} = 0$.

```
Step 7: $B_{11} \leftarrow B_{11} \cup \{N_4\}$; $B_4 \leftarrow B_4 \cup \{N_{11}\}$;
        $C \leftarrow C \cup \{GQ\}$.

Step 8: Set node $N_3$ as "MARKED".
```
Note that the current AG graph is shown in Figure 6(c).

<u>Iteration 4</u>

```
Step 4: $N_{next} = N_5$; $M_{next} = 2$.

Step 5: New node: $N_{51}$ and $N_{52}$;
        $N \leftarrow N \cup \{N_{51}, N_{52}\}$.

Step 6: $B_{51} = \{N_5\}$; $B_{52} = \{N_5\}$;
        $A_{51} = \{(V_j, \sim T), (V_n, \sim T)\}$;
        $A_{52} = \{(V_n, \sim T), (V_o, \sim T)\}$;
        $B_5 \leftarrow B_5 \cup \{N_{51}, N_{52}\}$;
        $B \leftarrow B \cup \{B_{51}, B_{52}\}$; $A \leftarrow A \cup \{A_{51}, A_{52}\}$.

Step 7: New vertex $V_r$.
        $B_{51} \leftarrow B_{51} \cup \{N_4, N_{52}\}$;
        $A_{51} = \{(V_i, \sim T), (V_j, \sim T), (V_n, \sim T), (V_r, \sim T)\}$;
        $B_{52} \leftarrow B_{52} \cup \{N_{51}, N_6\}$;
        $A_{52} = \{(V_r, \sim T), (V_n, \sim T), (V_o, \sim T), (V_p, \sim T)\}$;
        $B_4 \leftarrow B_4 \cup \{N_{51}\}$; $B_6 \leftarrow B_6 \cup \{N_{52}\}$;
        $C \leftarrow C \cup \{JN, NO, IJ, NR, OP\}$.

Step 8: Set node $N_5$ as "MARKED".
```
Note that the current AG graph is shown in Figure 6(d).

<u>Iteration 5</u>

```
Step 4: $N_{next} = N_6$; $M_{next} = 1$.

Step 5: New node: $N_{61}$; $N \leftarrow N \cup \{N_{61}\}$.

Step 6: $B_{61} = \{N_6\}$;
        $A_{61} = \{(V_p, \sim T), (V_l, \sim T)\}$;
        $B_6 \leftarrow B_6 \cup \{N_{61}\}$; $B \leftarrow B \cup \{B_{61}\}$; $A \leftarrow A \cup \{A_{61}\}$.

Step 7: New vertex $V_s$.
      $A_4 = \{(V_h, T), (V_i, \sim T), (V_j, \sim T), V_k, \sim T), (V_l, \sim T),$
          $(V_s, \sim T), (V_m, T), (V_g, \sim T), (V_f, \sim T), (V_e, \sim T),$
          $(V_q, \sim T)\}$;
      $B_{61} \leftarrow B_{61} \cup \{N_{52}, N_4\}$;
      $A_{61} = \{(V_r, \sim T), (V_p, \sim T), (V_l, \sim T), (V_s, \sim T)\}$;
      $B_{52} \leftarrow B_{52} \cup \{N_{61}\}$; $C \leftarrow C \cup \{LP, LS\}$.

Step 8: Set node $N_6$ as "MARKED".
```
Note that the current AG graph is shown in Figure 6(e).

Because all nodes in N' without a T-type vertex are marked, we exit the REPEAT loop.

Step 9:
Vertex V_r is a common vertex in junction sets A_{51}, A_{52}, and A_{61}. By applying Property 4, nodes N_{51} and N_{52}, and N_{61} should be connected. From Property 3, junction sets A_{51} and A_{61} must have a common edge. Therefore, vertex V_s is a dummy vertex and should be replaced by Vertex V_i.
$C \leftarrow C \cup \{IR, LI\}$; $A_{61} = \{(V_r, \sim T), (V_p, \sim T), (V_l, \sim T), (V_i, \sim T)\}$; $A_4 = \{(V_i, \sim T), (V_j, \sim T), (V_k, \sim T), (V_l, \sim T), (V_g, \sim T), (V_f, \sim T), (V_e, \sim T), (V_q, \sim T)\}$.

Therefore, the inferred complete AG graph is as the one given in Figure 6(e) and its property set is as follows.

$$
\begin{aligned}
A = \{ & \{(V_a, \sim T), (V_b, \sim T), (V_c, \sim T), (V_d, \sim T)\}, \\
& \{(V_a, \sim T), (V_e, \sim T), (V_f, \sim T), (V_b, \sim T)\}, \\
& \{(V_b, \sim T), (V_f, \sim T), (V_g, \sim T), (V_c, \sim T)\}, \\
& \{(V_i, \sim T), (V_j, \sim T), (V_k, \sim T), (V_l, \sim T), (V_g, \sim T), \\
& \quad (V_f, \sim T), (V_e, \sim T), (V_q, \sim T)\}, \\
& \{(V_j, \sim T), (V_n, \sim T), (V_o, \sim T), (V_k, \sim T)\}, \\
& \{(V_k, \sim T), (V_o, \sim T), (V_p, \sim T), (V_l, \sim T)\}, \\
& \{(V_d, \sim T), (V_a, \sim T), (V_e, \sim T), (V_q, \sim T)\}, \\
& \{(V_q, \sim T), (V_c, \sim T), (V_d, \sim T), (V_g, \sim T)\}, \\
& \{(V_i, \sim T), (V_j, \sim T), (V_n, \sim T), (V_r, \sim T)\}, \\
& \{(V_r, \sim T), (V_n, \sim T), (V_o, \sim T), (V_p, \sim T)\}, \\
& \{(V_r, \sim T), (V_p, \sim T), (V_l, \sim T), (V_i, \sim T)\}\}
\end{aligned}
$$

Recall that the inferred AG graph and its property set are equivalent to the complete AG graph shown in Figure 1(b). Nodes N_{11}, N_{21}, N_{51}, N_{52}, and N_{61}, in Figure 6(e), correspond to nodes N_8, N_7, N_9, N_{10}, and N_{11} in Figure 1(b), respectively.

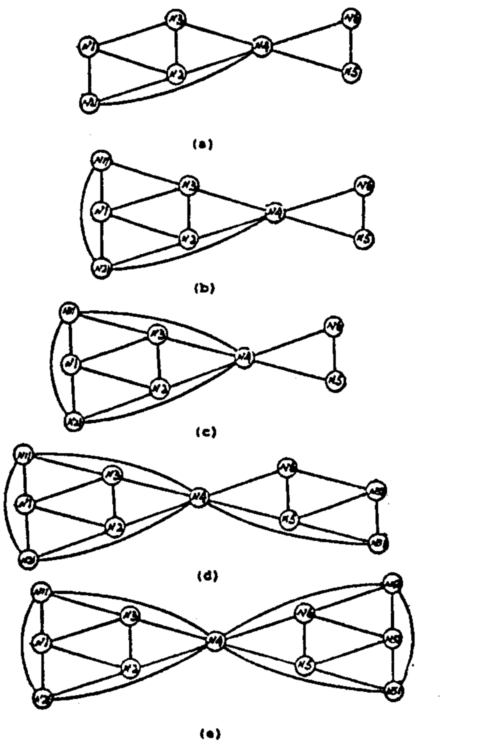


Figure 6. The Current AG graphs during The Inference

Concluding Remarks

In this paper we examined the problem of inferring the entire geometry of a 3-D object, given a partial 2-D projection of the object. For this reason the concept of the

Adjacency Graph (or AG graph) was developed. Two types of AG graphs were considered. *Complete* and *incomplete* AG graphs. Therefore, the problem was to infer a complete AG graph from an incomplete AG graph. An efficient algorithm (it takes $O(n^3)$ time) is given for this purpose and it is illustrated via an example.

The object inference problem is a critical problem in many industrial applications. Such applications include process planning, diagnostic systems, and motion analysis. Future research may focus on inferring the dimensions and angles of the invisible parts of a 3-D object.

REFERENCES

Akinniyi, F.A., A.K.C. Wong, and D.A. Stacey, 1986. *"A new algorithm for graph monomorphism based on the projections of the procedure graph,"* IEEE Transaction on System, Man, and Cybernetics, Vol.SMC-16, No.5, 1986, pp. 740-751.

Chakravarty, I., 1979. *"A generalized line and junction labelling scheme with application to scene analysis,"* IEEE Transaction on Pattern Analysis and Machine Intelligence, PAMI-1, No.2, pp. 202-205.

De Floriani, Lelia, 1986. *"A hierarchical boundary model for variable resolution representation of three-dimensional objects,"* Proceedings of International Conference on Pattern Recognition 8th, 1986, pp.226-229.

De Floriani, Lelia and Bianca Falcidieno, 1988. *"A hierarchical boundary model for solid object representation,"* ACM Transactions on Graphics, Vol 7, No. 1, January 1988, pp. 42-60.

Joshi, S. and T. C. Chang, 1988. *"Graph-based heuristic for recognition of machine features from a 3D solid model,"* Computer Aided Design, 20, 1988, pp. 58-66.

Joshi, S. and T. C. Chang, 1990. *"Feature extraction and feature based design approaches in the development of design interface for process planning,"* Intelligent Manufacturing, 1, 1990, pp.1-15.

Lu, Siwei and Andrew K. C. Wong, 1988. *"Analysis of 3-D scene with partially occluded objects for robot vision,"* Proceedings of International Conference on Pattern Recognition 9th, 1988, pp.303-308.

Wong, E.K. and K. S. Fu, 1985. ed.Julius T. Tou.; *"A graph-theoretic approach to 3-D object recognition and estimation of position and orientation,"* Computer-Based Automation, Plenum, New York, 1985, pp.305-344.

Wong, A. K. C. and S. W. Lu, 1983. *"Representation of 3-D objects by attributed hypergraphs for computer vision,"* Proceedings of International Conference on Systems, Man, and Cybernetics, 1983, pp. 49-53.

Wong, A. K. C. and S. W. Lu, 1985. *"Recognition and knowledge synthesis of 3-D object images,"* Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 1985, pp.162-166.

APPLIED COMPUTING: TECHNOLOGICAL CHALLENGES OF THE 1990'S

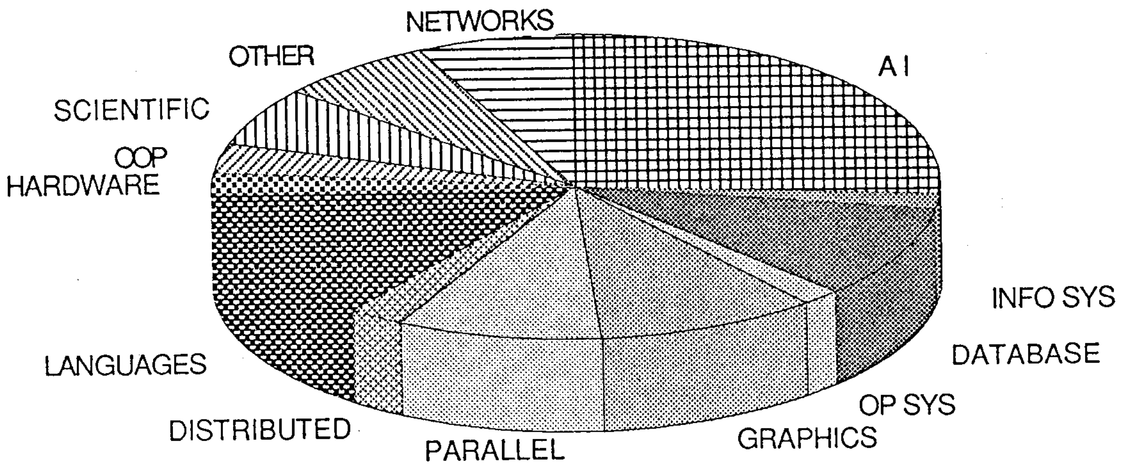
PROCEEDINGS OF THE 1992 ACM/SIGAPP SYMPOSIUM ON APPLIED COMPUTING

VOLUME I

Kansas City Convention Center
March 1-3, 1992

editors:

Hal Berghel
Ed Deaton
George Hedrick
David Roach
Roger Wainwright



acm PRESS