CMU-CS-85-103

Representation and Incremental Construction of a Three-Dimensional Scene Model

Martin Herman

January 1985

[redacted]

Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

This research was sponsored partly 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, and partly by the Air Force Office of Scientific Research under contract F49620-83-C-0100.

Abstract

The representation, construction, and updating of the 3D scene model derived by the **3D Mosaic** scene understanding system is described. The scene model is a surface-based description of an urban scene, and is incrementally acquired from a sequence of images obtained from multiple viewpoints. Each view of the scene undergoes analysis which results in a 3D wire-frame description that represents portions of edges and vertices of buildings. The initial model, constructed from the wire frames obtained from the first view, represents an initial approximation of the scene. As each successive view is processed, the model is incrementally updated and gradually becomes more accurate and complete. Task-specific knowledge is used to construct and update the model from the wire frames. At any point along its development, the model represents the current understanding of the scene and may be used for tasks such as matching, display generation, planning paths through the scene, and making other decisions dealing with the scene environment.

The model is represented as a graph in terms of symbolic primitives such as faces, edges, vertices, and their topology and geometry. This permits the representation of partially complete, planar-faced objects. Because incremental modifications to the model must be easy to perform, the model contains mechanisms to (1) add primitives in a manner such that constraints on geometry imposed by these additions are propagated throughout the model, and (2) modify and delete primitives if discrepancies arise between newly derived and current information. The model also contains mechanisms that permit the generation, addition, and deletion of hypotheses for parts of the scene for which there is little data.

We describe an experiment in which the model is generated and updated from two views.

1. Introduction

The **3D Mosaic** scene understanding system is a vision system that incrementally generates a three-dimensional description (or model) of a complex scene from multiple images. It is an entire system in the sense that it starts with images and ends with symbolic 3D descriptions. It therefore encompasses several levels of the vision process and contains several components, including stereo analysis, monocular analysis, and constructing and updating the scene model. This paper concentrates on the component that constructs and updates the model. For a description of the stereo and monocular analysis components, see [Herman 83, Herman and Kanade 84, Herman, Kanade, and Kuroe 83, Herman, Kanade, and Kuroe 84].

This paper is organized as follows. First, the motivation for the approach of incrementally acquiring the scene model is presented, together with an overview of the system. Then the representation, construction, and updating of the scene model is described, along with the task-specific knowledge used here. Examples are shown of how urban scenes are reconstructed from complex aerial photographs.

2. Description of System

The goal of the **3D Mosaic** system is to obtain an understanding of the 3D configuration of surfaces and objects in a scene. The significance of this goal may be demonstrated by the following tasks.

1. **Model-based image interpretation.** A known 3D scene model can provide significant aid in interpreting arbitrary images of the scene [Barrow, Bolles, et al. 77, McKeown 83, Rubin 80]. The 3D Mosaic system performs the task of acquiring such a model of the scene.

2. **3D change detection.** Change detection is a task that determines how the geometry and structure of a scene changes over time. The conventional approach to this task involves comparing and detecting changes in images. However, because of different viewpoints and lighting conditions, changes in the images do not necessarily correspond to changes in the geometry and structure of the scene. If 3D scene descriptions were obtained from the images first, such descriptions could be compared in 3D to determine changes in the scene.

3. **Simulating the appearance of the scene.** If a 3D description of the scene were to be obtained, displays as seen from arbitrary viewpoints could be generated from it. This is useful for tasks such as familiarizing personnel with a given area, and flight planning by generating the scene appearance along hypothetical flight paths.

4. **Robot navigation.** Three-dimensional descriptions of complex environments may be used to make decisions dealing with path planning or determining which parts of the environment to analyze in more detail.

Note that to perform these tasks, a vision system must do more than classify images, segment them, or identify objects in them; it must be able to generate a 3D description of the scene.

The **3D Mosaic** system deals with complex, real-world scenes (e.g., Fig. 1 and Fig. 2). That is, the scenes contain many objects with a variety of shapes, the object surfaces have a variety of textures and reflectance characterisics, and the scenes are imaged under outdoor lighting conditions. Because of the complexity, there are many difficulties in interpreting the images, including:

1. Any particular image contains only partial information about the scene because many surfaces are occluded.

2. Even portions of the scene that are visible are often difficult to recover. For example, surfaces with dark shadows cast across them, or with highlights, may be difficult to interpret. Highly oblique surfaces may be difficult to analyze if their resolution in the image is poor. Such portions of the scene, therefore, may be recovered with errors and inconsistencies, or may not be recovered at all.

Our approach to the problems of complexity is to use multiple images obtained from multiple viewpoints. This approach aids interpretation in two ways. First, surfaces occluded in one image may become visible in another. Second, features of surfaces that are difficult to analyze and interpret in one image (such as scene edges and texture) may become more apparent in another image because of different viewpoint and/or lighting conditions.

2.1. Incremental Approach

A large number of views will, in general, be required to obtain a fully accurate and complete description of a complex scene. Typically, all these views will not be simultaneously available, while some may never become available. Many of them will only be obtained gradually through interaction with the scene environment. Our system must therefore have the ability to utilize partial descriptions and incrementally update them with new information whenever a new view happens to become available. As a practical example, consider a robot (perhaps a mobile ground robot or an automatically guided airplane) which is attempting to navigate through an unknown environment. The robot would sequentially acquire images of the environment as it moves about. Information derived from each new image would serve to update its internal model, and this partial model would be used to decide where to go next, or where to analyze in more detail.

We have adopted an approach in which the 3D scene model is incrementally acquired over the multiple views. The views of the scene are sequentially acquired and processed. Partial 3D information is derived from each view. The initial model is constructed from 3D information obtained from the first view, and represents an initial approximation of the scene. As each successive view is processed, the model is incrementally updated and gradually becomes more accurate and complete.

In our approach, the scene model plays the role of a central representation with two primary functions. First, it incrementally accumulates information about the scene. Second, at any point along its development,

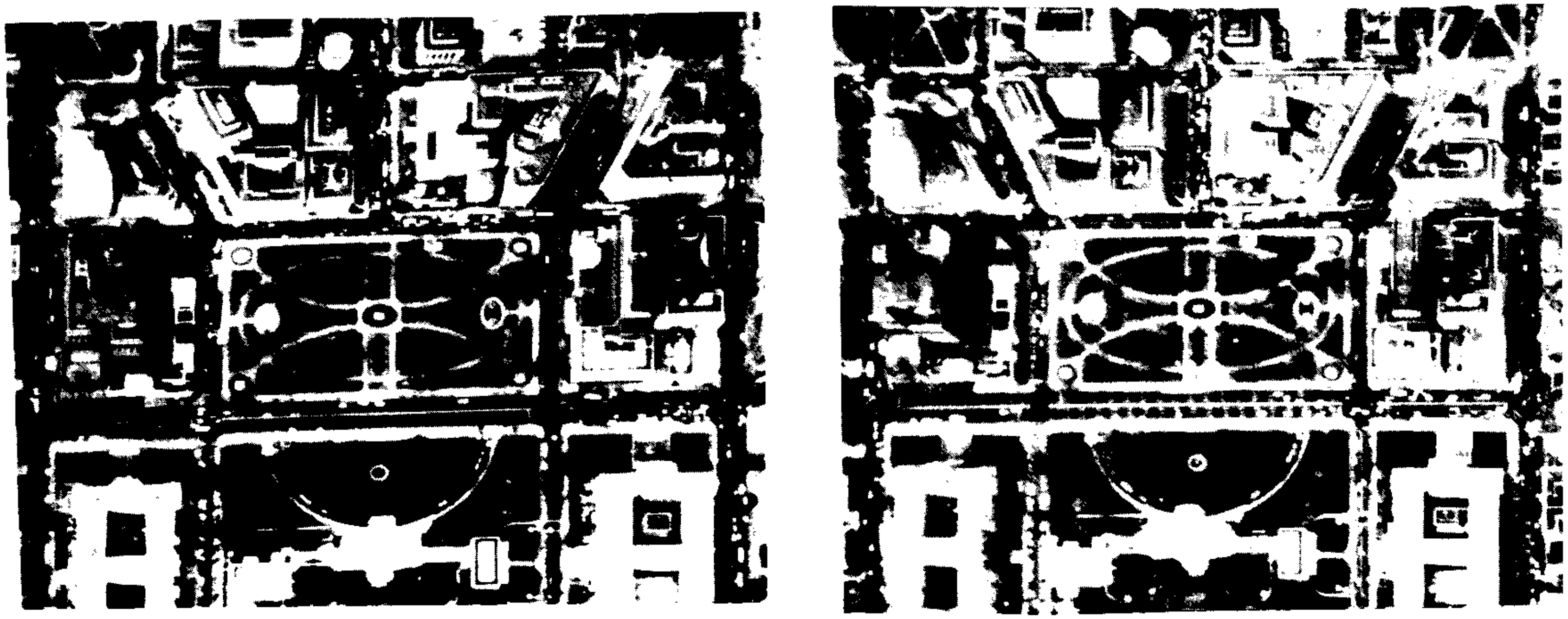


Figure 1: Gray scale stereo images of a region of Washington, D. C.

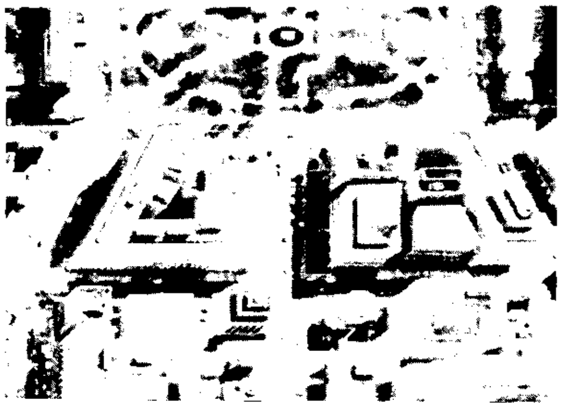


Figure 2: Aerial photograph showing part of Washington, D.C. This is a different view of the same scene as in Fig. 1.

it represents the current understanding of the scene. As such, it may be used for tasks such as matching, display generation, planning paths through the scene, and making other decisions about the scene environment. Two such tasks are important for the incremental acquisition process itself: (1) 3D information derived from a new view must be matched to the model so that updating can occur, (2) higher-level components should be able to use the model to determine which parts of the scene to analyze in more detail, and from which viewpoints to take the next images.

Most previous research efforts at acquiring 3D scene descriptions from multiple views have dealt with relatively simple scenes in controlled environments [Baker 77, Baumgart 74, Bourne, Milligan, and Wright 82, Martin and Aggarwal 83, Potmesil 83, Underwood 75]. This has led, in some cases, to only utilizing occluding contours in the image to form the 3D description [Baker 77, Baumgart 74, Bourne, Milligan, and Wright 82, Martin and Aggarwal 83]. The work of Moravec [Moravec 80] deals with complex indoor and outdoor scenes, but the 3D descriptions generated by his system consist of sparse sets of feature points. Our system, on the other hand, generates full, surface-based descriptions.

2.2. Overview of System

A flowchart for the **3D Mosaic** system, showing the major modules and data structures, is displayed in Fig. 3. The input is a new view of the scene, which may be either a stereo image pair or a single image. The stereo pair undergoes stereo analysis, while the single image undergoes monocular analysis. The purpose of these analyses is to obtain 3D scene features such as portions of surfaces, edges, and corners. The stereo analysis component currently matches junctions extracted from the two images, and generates a sparse 3D wire-frame description of the scene. The monocular analysis component currently extracts linear structures from the image and converts these to 3D wire frames using task-specific assumptions.

The central scene model is a surface-based description which is constructed and modified from these features. It is represented as a graph in terms of primitives such as faces, edges, vertices, and their topology and geometry. It also has mechanisms to add and delete hypotheses for parts of the scene for which there are partial data. Before modifications to the scene model can occur, the 3D features from the new view must be matched to the current model. The scene model may, at any point along its development, be used for tasks such as image interpretation, planning, or display generation. A new view may then be acquired which may further modify the model.

For example, when the stereo analysis component is applied to the images in Fig. 1, the result is the set of wire frames in Fig. 12. The scene model constructed from these wire frames is shown in Fig. 19. When the monocular analysis component is applied to the image in Fig. 2, the result is the set of wire frames in Fig. 20. These, in turn, are converted into the scene model in Fig. 21. Finally, the result of modifying the model in

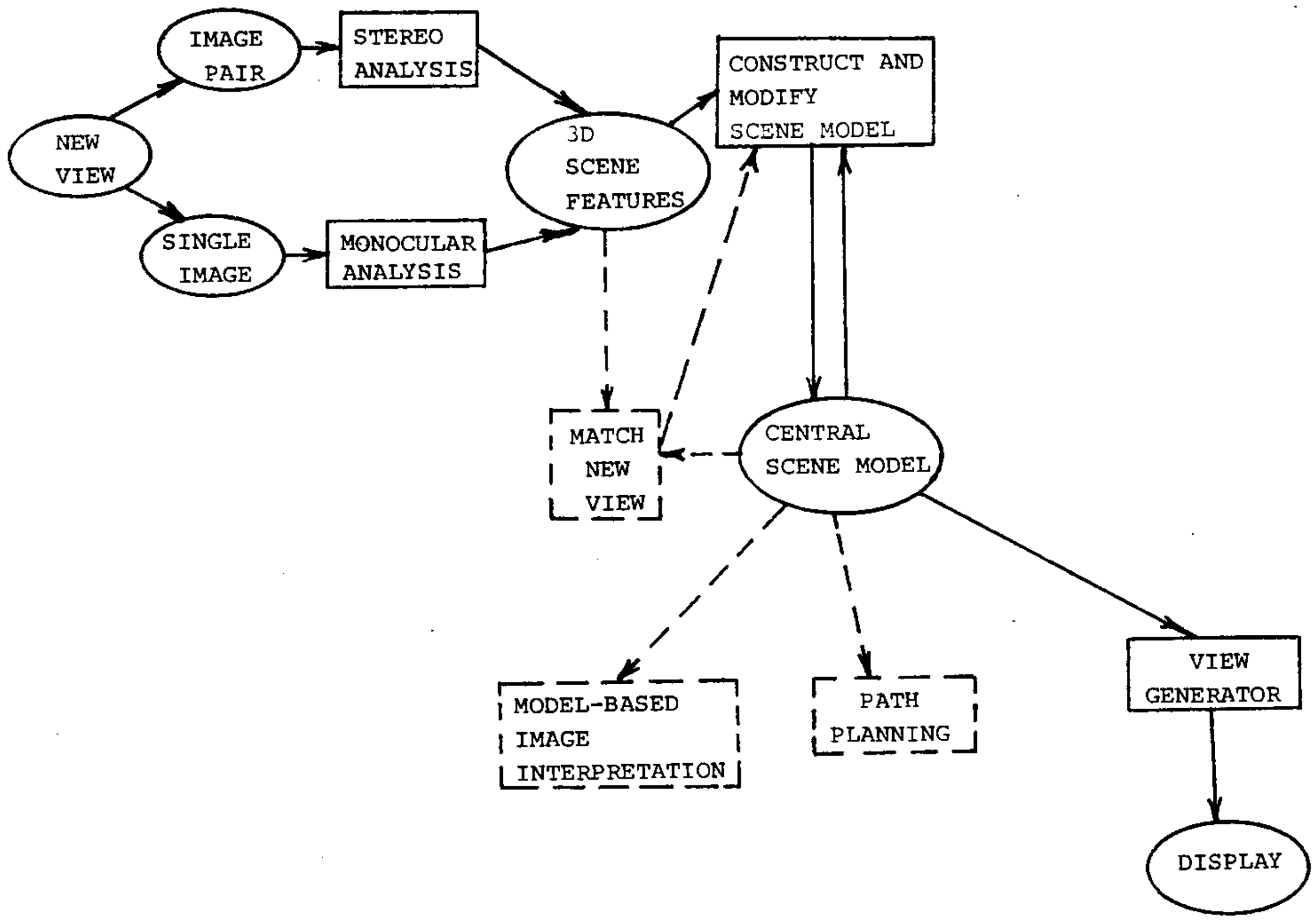


Figure 3: **3D Mosaic** flowchart, showing major modules (boxes) and data structures (ellipses). The dashed lines represent components that have not yet been implemented; the solid lines represent components already implemented.

Fig. 19 with a new view is shown in Fig. 26.

3. Representing and Manipulating the 3D Scene Model

The representation we have developed for the 3D scene model draws on ideas from geometric modelling used in computer-aided design systems [Baer, Eastman, and Henrion 79, Requicha 80]. In these systems, however, the 3D models are usually derived through interaction with a user. Our case is different in that (1) the 3D models are derived automatically from 2D images, and (2) many portions of the scene are unknown or recovered with errors because of occlusions or unreliable analysis.

The following factors have determined how the scene model is represented and manipulated.

1. Partially complete, planar-faced objects must be efficiently described by the model. It is therefore represented as a graph in terms of symbolic primitives such as faces, edges, vertices, and their topology and geometry. Information is added and deleted by means of these primitives.

2. The model must be easy to use in matching.

3. Because scene approximations are often more useful if they contain reasonable hypotheses for parts of the scene for which there are partial data, we introduce mechanisms that permit hypotheses to be generated, added, and deleted.

4. Because incremental modifications to the model must be easy to perform, we introduce mechanisms to (a) add primitives to the model in a manner such that constraints on geometry imposed by these additions are propagated throughout the model, and (b) modify and delete primitives if discrepancies arise between newly derived and current information.

3.1. Representation of Model

The 3D structure in the scene is represented in the form of a graph, called the *structure graph*. The nodes and links represent primitive topological and geometric constraints. The structure graph is incrementally constructed through the addition and deletion of these constraints. As constraints are accumulated, their effects are propagated to other parts of the graph so as to obtain globally consistent interpretations.

The current structure-graph representation models surfaces in the scene as polyhedra. The components of a polyhedral surface are the face, edge, and vertex. We distinguish the topology of the polyhedral components from their geometry [Baer, Eastman, and Henrion 79, Eastman and Preiss 82]. The geometry involves the physical dimensions and location in 3-space of each component, while the topology involves connections between the components.

3.2. Primitive Constraints

In the structure graph, nodes represent either primitive topological or primitive geometric elements. We define five types of primitive topological elements: faces, edges, vertices, objects, and edge-groups. The first three are components of the polyhedral surface. The last two are introduced in order to conveniently represent connected groups of elements. The object is intended to represent a connected set of faces that enclose a volume in 3-space. The edge-group is intended to represent a connected ring of edges that enclose an area in 2-space on a face. Because of the partial nature of the structure graph, however, an object may represent any set of faces, edge-groups, edges, and vertices that are potentially part of a single, closed, connected unit. Similarly, an edge-group may represent any set of edges and vertices that are potentially part of a single edge ring on a face.

Face, edge, and vertex nodes are tagged as either *confirmed* or *unconfirmed.* Confirmed means that the element represented by the node has been derived directly from images. Unconfirmed means that the element has only been hypothesized.

We define three types of primitive geometric elements: planes, lines, and points. These serve to constrain the 3-space locations of faces, edges, and vertices. Plane and line nodes contain plane and line equations, respectively. Point nodes contain coordinate values.

Although an edge is ideally delimited by two vertices, edges derived from images are often incomplete and may be delimited by one vertex and an end point which is not necessarily a vertex. Such a point is tagged as an end point. A point may also be tagged as confirmed or unconfirmed, depending on whether or not it has been derived from images. This is useful when a confirmed edge is hypothesized to extend further in length. The confirmed portion of the edge lies between confirmed points, while the unconfirmed portion may be delimited on one side by an unconfirmed point.

The structure graph contains two types of links: the *part-of* link, representing the part/whole relation between two topological nodes, and the *geometric constraint* link, representing the constraint relation between a geometric and topological node. For example, a vertex may be part of an edge, edge-group, face, or object. A point constrains the position of a vertex, edge, or face if it lies on the vertex, edge, or face, respectively.

Although several points may be constrained to lie on, say, a face, the points need not necessarily be coplanar. The equations of planes for faces and lines for edges are currently obtained by a least squares fit to all the points constraining the face or edge. In general, therefore, edges and vertices that are part of a face need not lie on the plane of the face, and vertices of an edge need not lie on the line of the edge.

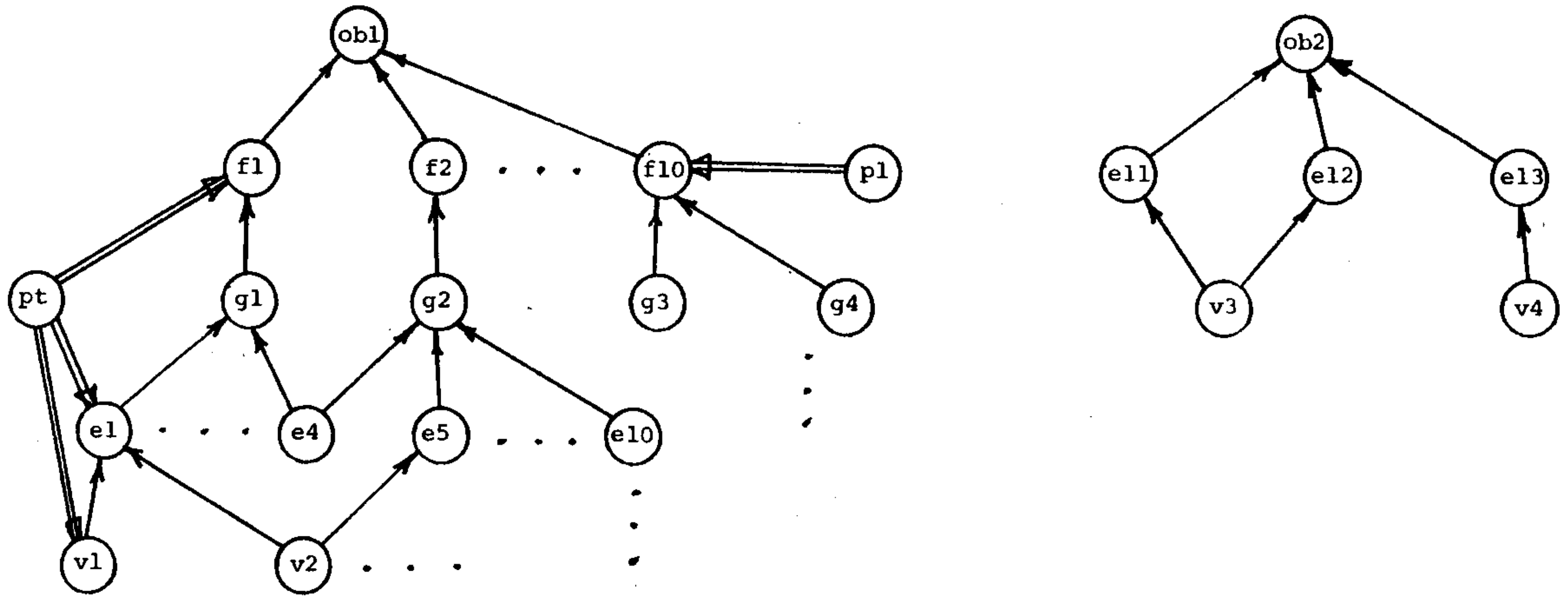


Figure 4: Simple example of a structure graph consisting of two objects, *ob1* and *ob2*. Double line arrows represent geometric constraint links, and single line arrows represent part-of links.

Fig. 4 shows a simple example of a structure graph consisting of two objects, *ob1* and *ob2*. Arrows with single lines represent part-of links, and arrows with double lines represent geometric constraint links. The object *ob1* contains ten faces, *f1* ... *f10*. The faces *f1* and *f2* each contains one edge-group, *g1* and *g2*, respectively. Notice that *f10* contains two edge-groups. These might represent a bounding outer ring of edges and an inner ring of edges that bounds a hole in the face. The edge-groups contain edges, each of which may be part of more than one edge-group. The edges, in turn, contain vertices. The point *pt* constrains *v1*, *e1*, and *f1*. The plane *pl* constrains *f10*. The other object in the structure graph, *ob2*, is highly incomplete. It contains only three edges and two vertices.

4. Modifications to the 3D Scene Model

Modifications to the structure graph are made by adding or deleting nodes and links, or changing the equations of line and plane nodes, or the coordinates of point nodes. All effects of modifications are propagated to other parts of the graph.

4.1. Propagation Due to Geometric Modifications

Consider adding or deleting a geometric constraint link between a geometric and topological node. Any of the three geometric nodes (points, lines, and planes) may constrain any of the three topological nodes (vertices, edges, and faces). Object and edge-group nodes may not be geometrically constrained directly. Fig. 5 shows how a constraint on one node may propagate to others. The arrows in the figure indicate the direction of propagation. The tail of an arrow indicates the source constraint; the head indicates the constraint implied by the source constraint.

We see in Fig. 5 that point constraints propagate upward. That is, if a point constrains a vertex, it must also constrain all edges and faces which contain that vertex. Similarly, a point that constrains an edge must also constrain all faces containing that edge. Note that when a point constrains an edge, we assume that no constraint is implied for arbitrary vertices that are part of that edge, since the point need not lie on any of these vertices. In one sense, the point may be considered to constrain such vertices since they must lie on a line going through the point. This constraint, however, is not useful until another constraint on the line is derived, such as another point that lies on the edge. In this case, our system generates the equation of the line that constrains the edge and propagates the line constraint down to the vertex, as explained in the next paragraph. A more direct and useful constraint is thus imposed on the vertex. Similarly, when a point constrains a face, no useful constraint is implied for arbitrary edges or vertices that are part of the face.

As indicated in Fig. 5, line constraints propagate outward. A line that constrains an edge must also constrain all faces containing the edge and all vertices that are part of the edge. Finally, plane constraints

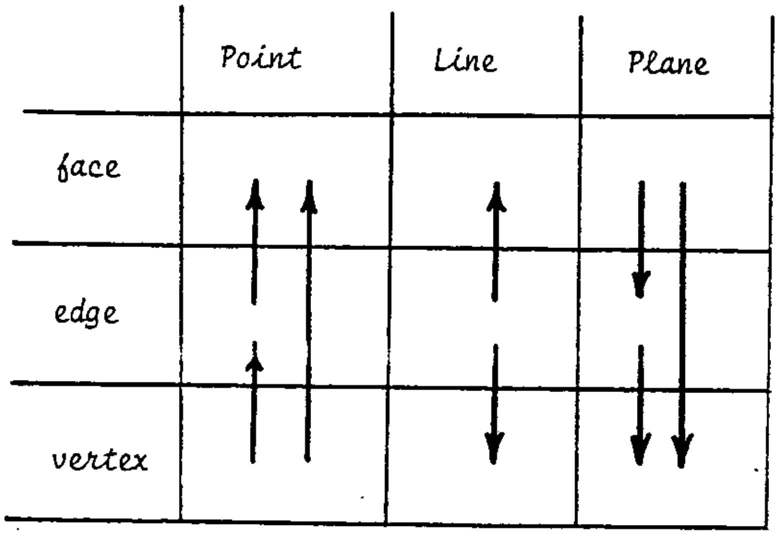


Figure 5: Rectangular boxes indicate geometric constraints on topological nodes. Arrows indicate direction of propagation of constraints.

propagate downward. A plane that constrains a face must also constrain all edges and vertices that are part of the face. Similarly, a plane that constrains an edge must also constrain all vertices that are part of the edge. Whenever a geometric constraint link is added, propagation occurs as indicated in Fig. 5.

When a geometric constraint link is deleted, the rest of the structure graph must be made consistent with this change. Our approach to this problem is based on the TMS system [Doyle 79], using the notion that when an assertion is deleted, all assertions implying it and all assertions implied by it that have no other support should also be deleted. To see this, consider Fig. 6. Let $\{x_1, x_2, \ldots, x_m\}$ be a set of assertions, each of which independently implies the assertion y. The assertion $(y \wedge v_1 \wedge v_2 \wedge \ldots)$, in turn, implies each assertion in the set $\{z_1, z_2, \ldots, z_n\}$. Furthermore, for each i, z_i is independently implied by each assertion in the set $\{w_{ij}\}$. Now suppose the assertion y is deleted, i.e., it is declared false. Then

1. Since each assertion z_i depends on the truth of y, z_i is deleted unless it has other support w_{ij}.

2. All assertions x_i are made false. None of them can be true, for if one were, y must be true. Since x_i may consist of a conjunction of assertions, at least one of them is deleted to make x_i false.

We obtain assertions that imply a given assertion by following backwards along the arrows in Fig. 5, and we obtain assertions implied by a given assertion by following forward along the arrows.

Consider the simple example in Fig. 7a, which depicts three topological nodes (vertex v, edge e, face f) constrained by one geometric node (point p). Suppose now that link 4 is deleted (Fig. 7b), that is, the assertion "p constrains e" is deleted. All assertions which have implied this must now be deleted, for if one were to hold, link 4 would also hold. To find these assertions, we locate the box in Fig. 5 that represents a

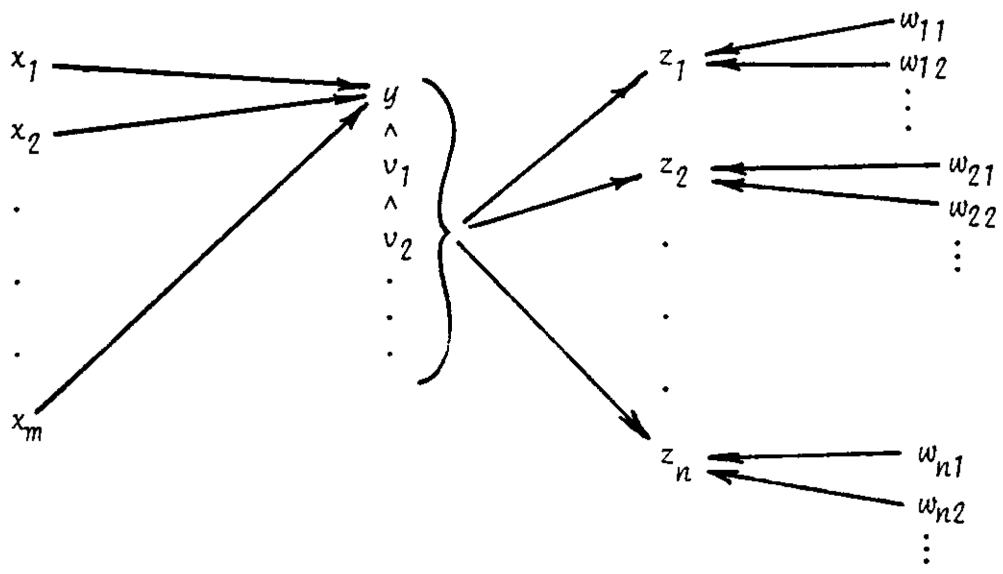


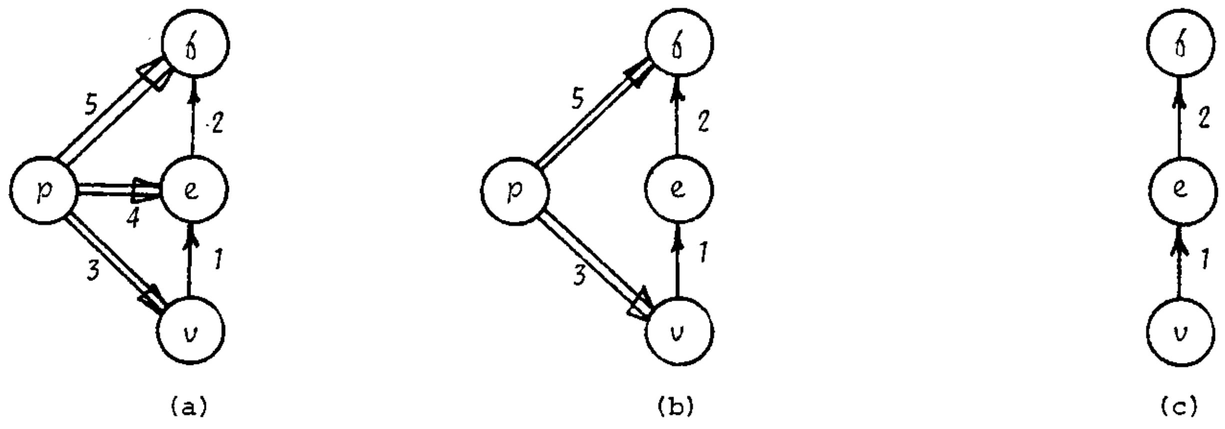
Figure 6: The assertion y is independently implied by each x_i. Each assertion z_i is independently implied by $(y \wedge v_1 \wedge v_2 \wedge \ldots)$ and w_{ij}.

point constraining an edge and follow backwards along the arrow. The result is the box that represents the point constraining any vertex of the edge. In Fig. 7b, this corresponds to the assertion "p constrains v, and v is part of e". This assertion must therefore be made false. To do so, we may delete either link 1, link 3, or both from Fig. 7b. Our intuition tells us that part-of links (link 1) should dominate constraint links (link 3), and thus link 3 is deleted. This seems to work well for our examples.

We now must determine the assertions implied by the one initially deleted. All these assertions must also be deleted unless they have other support. To do so, we follow forward along the arrow from the box in Fig. 5 that represents a point constraining an edge, and the result is the box that represents the point constraining all faces containing the edge. In Fig. 7b, this corresponds to the assertion "p constrains f", which is link 5. This link should therefore be deleted since it has no other support. One possible source of other support is external to the structure graph. Link 5 may have been derived, for example, directly from image data, rather than through structure graph propagation. We rule out the possibility that links 4 and 5 are unrelated, and thus delete link 5. The resulting structure graph is depicted in Fig. 7c.

4.2. Propagation Due to Topological Modifications

When a topological part-of link between two topological nodes is added or deleted, the effects are propagated to other parts of the structure graph. In the following, we will consider both geometric and topological effects.



v is part of *e* (link 1)
e is part of *f* (link 2)
p constrains *v* (link 3)
p constrains *e* (link 4)
p constrains *f* (link 5)

Figure 7: (a) Initial structure graph. (b) Link 4 is deleted. (c) Resulting structure graph after effects of deletion have been propagated.

4.2.1. Geometric Effects

When a topological part-of link is added between two topological nodes, the geometric constraints on each node must be propagated to the other node in accordance with the chart in Fig. 5. There are three main cases to consider: (1) adding a part-of link between a vertex and edge node, (2) between an edge and face node, and (3) between a vertex and face node. These three cases are explicitly covered in Fig. 5. The remaining cases fall into two classes: (a) adding a part-of link between some topological node and an object node, and (b) between some topological node and an edge-group node. Since object nodes cannot be geometrically constrained directly, actions in class (a) have no geometric effects. Since geometric constraints can be propagated through edge-group nodes, actions in class (b) do have geometric effects. These effects, however, can be reduced to the three cases above, as explained in the next paragraph.

Consider the example of adding a part-of link between an edge node E and a face node F. From Fig. 5, we see that all point and line constraints on E must be propagated to F, while all plane constraints on F must be propagated to E. Plane constraints propagated to E are, in turn, propagated to vertices of E. As another example, consider adding a part-of link between an edge-group node G and a face node F. This situation results in the same geometric propagation as the following two cases: (1) add a part-of link from each edge of G to F, and (2) from each vertex of G to F. Similar rules can be established for the other two situations involving edge-group nodes (i.e., adding a link between a vertex and edge-group node, and between an edge and edge-group node).

When a part-of link between two topological nodes is deleted, an attempt is made to nullify any geometric propagation that occurred through the link. This is done by deleting, from the two nodes connected by the link, all geometric constraints that have propagated through the link. The effects of deleting these geometric constraint links are, in turn, propagated to the rest of the graph in the manner described in the previous section.

As an example, consider deleting a part-of link between an edge node E and a face node F. As seen in Fig. 5, all point and line constraints on F that also constrain E were either (1) propagated up from E, (2) propagated up from another edge or vertex of F, or (3) derived from an external source. We rule out the possibility that the same constraints on E and F are unrelated, thus ruling out the external source. Therefore, points and lines that constrain both F and E, but do not also constrain another edge or vertex of F, are deleted from F since we just cut off the only path through which they could have propagated to F. The effects of deleting the point and line constraints from F are, in turn, propagated to the rest of the graph. Similarly, all plane constraints on E that also constrain F are deleted from E unless they also constrain another face that contains E (which would be unusual). The effects of deleting plane constraints from E are then propagated.

An example of a link with more than one source of support is shown in Fig. 8a. Suppose the part-of link between *e1* and *f*, link 4, is deleted (Fig. 8b). According to the chart in Fig. 5, link 8 is a candidate for deletion since the point node *p* constrains both *e1* and *f*. However, since *p* also constrains the edge *e2*, which is part of *f*, link 8 is still valid.

4.2.2. Topological Effects

A topological modification sometimes implies topological changes elsewhere in the structure graph. This is best illustrated through an example. Fig. 9a shows the graph representing the situation in Fig. 9b. The edge *e* has two vertices, *v1* and *v2*, and *v1* is known to be part of the face *f*. Now suppose a part-of link is added between *v2* and *f* (link 4 in Fig. 9c). Since both vertices of *e* are now part of *f*, *e* must also be part of *f*, as shown in Fig. 9d. Therefore link 5 in Fig. 9c is added.

Another kind of topological effect results from the desire to eliminate redundant part-of links. Part-of links serve as paths in the structure graph along which effects of geometric changes are propagated. In order to simplify this process, the number of paths between each pair of topological nodes is minimized using the

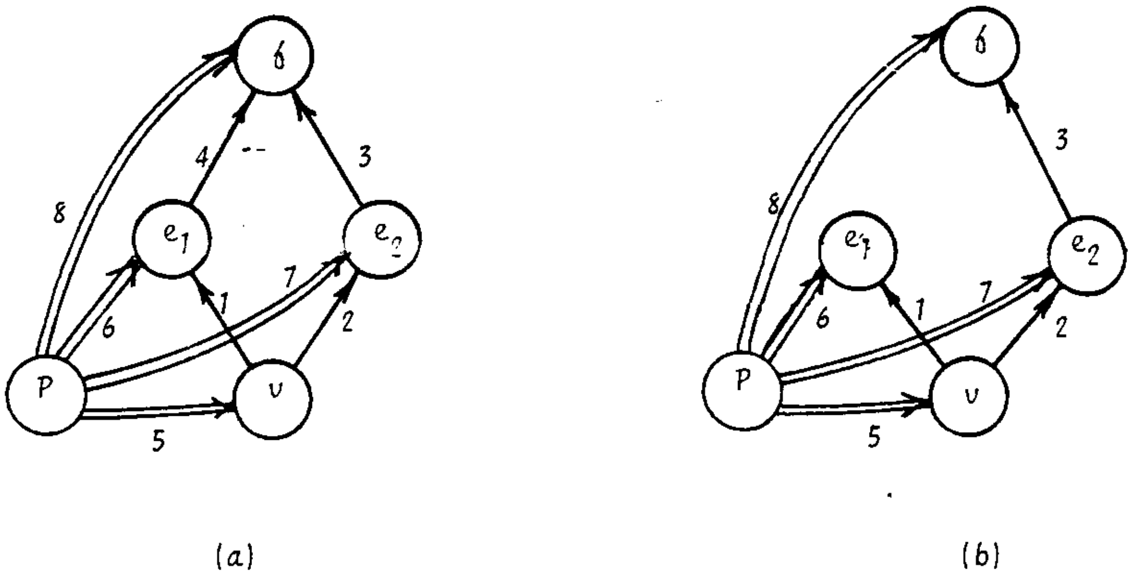


Figure 8: Example of a link with more than one source of support. (a) Initial structure graph. (b) Link 4 is deleted, but link 8 remains because of support from links 3 and 7.

following rule: Two topological nodes may not be directly connected (i.e., by means of a part-of link) if they are also connected through one or more intermediate topological nodes. For example, suppose a part-of link is added between the edge node *e* and the face node *f* in Fig. 10a. To avoid redundancy, all links connecting vertex nodes of *e* and the node *f* (link 1 in Fig. 10a) and vertex nodes of *e* and object nodes containing *f* (link 2) are deleted. In addition, if there were any links between *e* and object nodes containing *f*, they would also be deleted. The final configuration is shown in Fig. 10b. In the example of Fig. 9, the graph in (c) has redundant links. Links 1 and 4 are therefore deleted, resulting in the graph of Fig. 9e.

Although adding a part-of link can result in topological changes elsewhere in the graph, deleting a part-of link does not change the topology anywhere else. No attempt is made to recover previous states of topological connections. Fig. 11b shows the result of deleting link 1 from the graph in Fig. 11a. This technique seems to work well in our experiments.

5. Constructing and Updating the 3D Scene Model

Each view of the scene (which may be either a single image or a stereo pair) undergoes analysis which results in a 3D wire-frame description representing 3D vertices and edges corresponding to portions of boundaries of objects in the scene. The goal of the updating process is to merge the wire-frame description

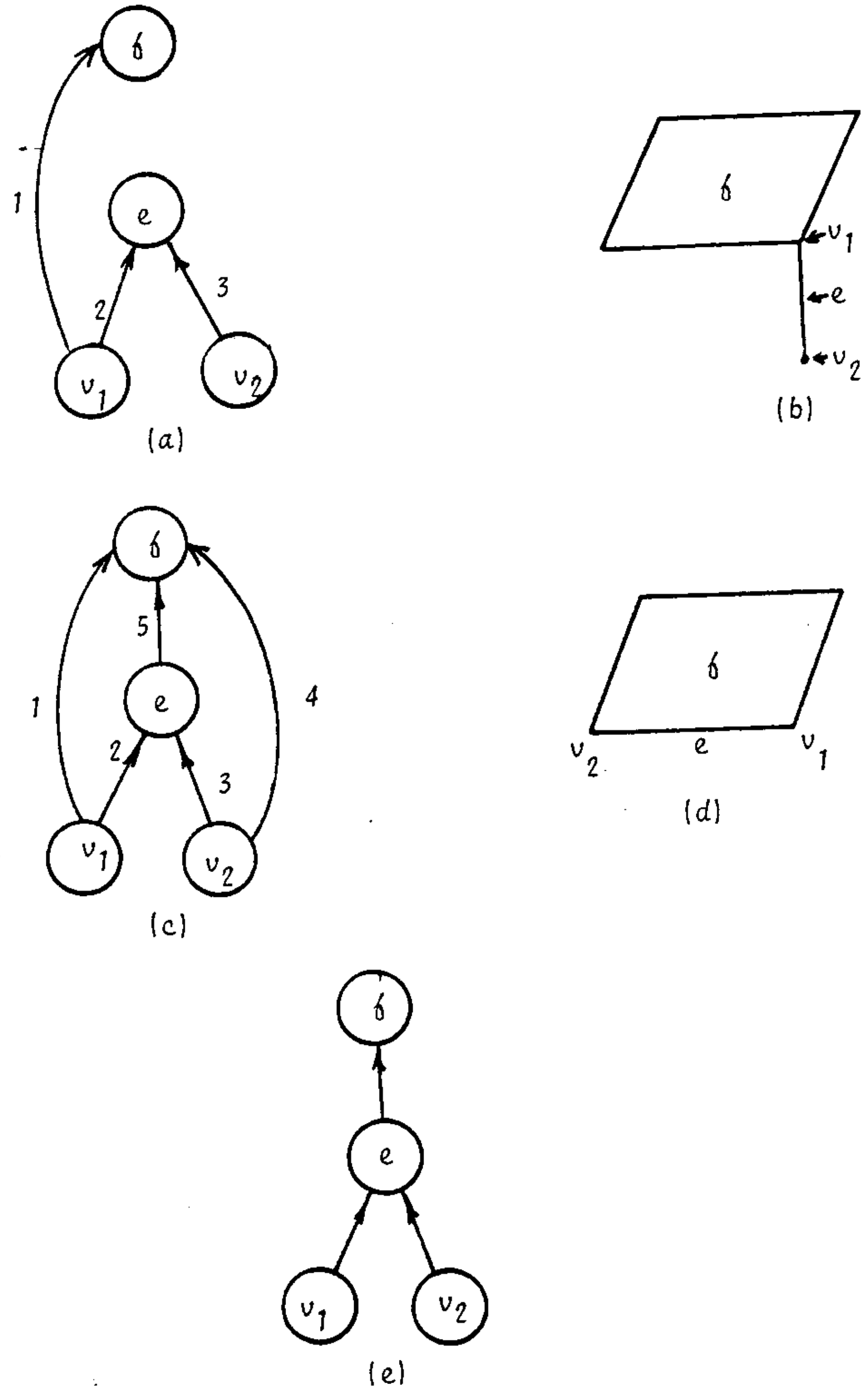


Figure 9: Topological propagation. (a) and (b) Initial situation. (c) and (d) Link 4 is added, resulting in addition of link 5. (e) Redundant links are eliminated.

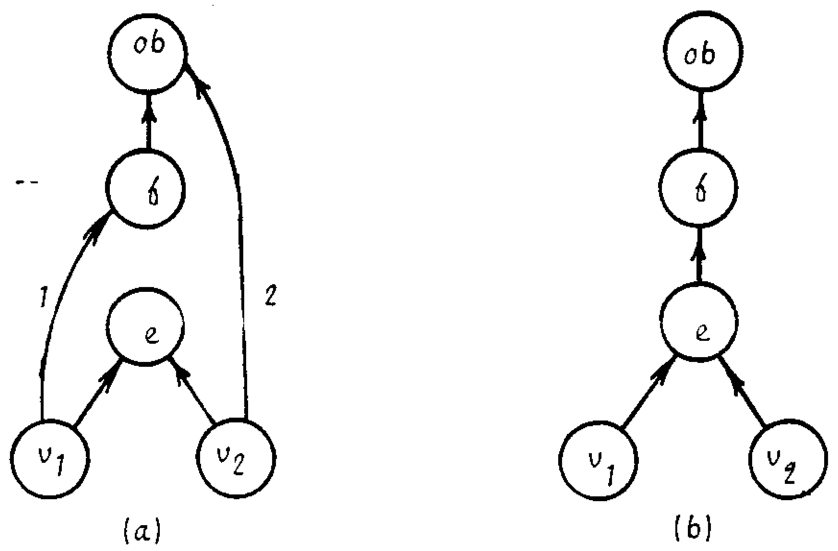


Figure 10: (a) Initial configuration. (b) When a link is added from *e* to *f*, links 1 and 2 are deleted to eliminate redundancy.

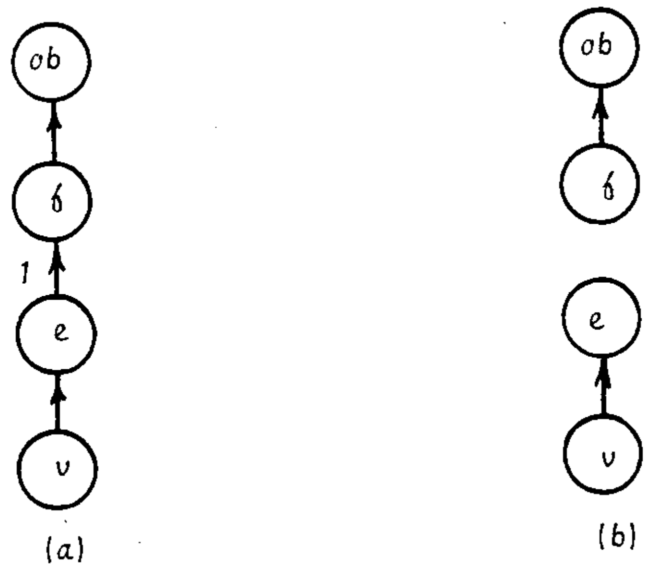


Figure 11: (a) Initial configuration. (b) Final result after link 1 is deleted.

with the current model. In general, this process will result in a partial 3D model which may consist of surfaces at some places but only portions of boundaries at other places. This partial 3D model must then be converted into a full surface-based description by hypothesizing new vertices, edges, and faces. Our current techniques for making such hypotheses exploit task-specific knowledge that falls into two categories: (1) knowledge of planar-faced objects, and (2) knowledge of urban scenes. These categories will be explored in detail in the next two sections.

Both the wire frames and scene models are represented by structure graphs. The wire-frame description extracted from the first view forms the initial state of the scene model, and all of its edges, vertices, and points are tagged as confirmed. This wire-frame model is then converted into a full surface-based model using task-specific knowledge. All elements of the model that were not present in the initial state are hypothesized and tagged as unconfirmed.

When a wire-frame description is extracted from a new view, all of its edges, vertices, and points are tagged as confirmed. This description is then matched to the current model (in order to find corresponding elements in the two and the coordinate transformation from one to the other) and merged with the current model. In the merging process, confirmed elements in the wire-frames and model that match are "averaged" together, resulting in new confirmed elements. Parts of the wire-frames that have no match in the model are then added to the model. Hypothesized elements in the model that are no longer consistent with confirmed parts are deleted. At this point, task-specific knowledge is again used to fill out the model and to form a full surface-based description.

6. Knowledge of Planar-Faced Objects

Since the structure graph has been designed for scenes that can be modelled as collections of planar-faced objects, knowledge of such objects is inherent in the representation and propagation rules, as described previously. In this section, we discuss how knowledge of such objects is used to construct a scene model from wire frames.

When new wire-frame information (derived either from the first or a subsequent view) is added to the model, many object descriptions will be incomplete. A goal of the model construction process, of course, is to complete these object descriptions using task-specific knowledge. The notion of an object description being complete is best expressed in the context of the structure graph. An object node in the structure graph is considered complete if it meets certain requirements, which may be expressed in terms of complete nodes contained by the object node. Each type of node in the graph, therefore, must meet certain requirements to be considered complete. Even though these requirements are only implicitly followed during the model construction process, it is useful to state them explicitly.

1. An object node is complete if it is closed, i.e., each edge node of the object is part of two face nodes, both of which are complete.

2. A face node is complete if it is constrained by a plane node and contains one or more complete edge-group nodes. One of these edge-group nodes must represent a bounding ring of edges on the face. The other, optional edge-group nodes represent inner edge rings, which would be holes in the face. In addition, each edge node of the face must be part of an edge-group of the face.

3. An edge-group node is complete if it contains a single, connected, closed ring of complete edges on a face.

4. An edge node is complete if it is constrained by a line node and contains two complete vertex nodes.

5. A vertex node is complete if it is constrained by a point node.

In the following, we discuss heuristics applicable to planar-faced objects which are used in constructing the model.

6.1. Combining Edges

If there are two confirmed edges in the model that are nearly parallel, very close to each other, and overlap significantly, they are merged into a single edge, which is also labeled as confirmed. The test to determine parallelism and closeness involves checking whether all the points on one edge are within a threshold distance from the line constraining the other edge, and vice versa. The test for overlap involves projecting one edge onto the line of the other and measuring the amount of overlap.

In determining how to merge two such edges, we have thus far considered only one situation, depicted in Fig. 13a. Edges *e1* and *e3* satisfy the merging condition, and each has a single confirmed vertex (*v1* on *e1* and *v2* on *e2*). Furthermore, the confirmed vertices are on opposite ends of each other. This situation is handled by merging the two edges into a single edge whose two end points are the two confirmed vertices, as shown in Fig. 13b. This situation occurs only once in Fig. 12, for the two edges labeled E1 and E2.

6.2. Generating Web Faces

Each vertex in the model is assumed to correspond to a corner of an object. Therefore each adjacent pair of legs ordered around the vertex corresponds to the corner of a planar face. Thus far in our experiments, we have dealt only with trihedral vertices. In this case, every pair of legs of each vertex corresponds to the corner of a separate face. A partial face, called a *web face*, is generated for each pair.

Fig. 14a shows three web faces generated from a trihedral vertex. A web face may lie on either side of a vertex corner. In Fig. 14b, the web face is on the "inside", while in (c) it is on the "outside", of the vertex

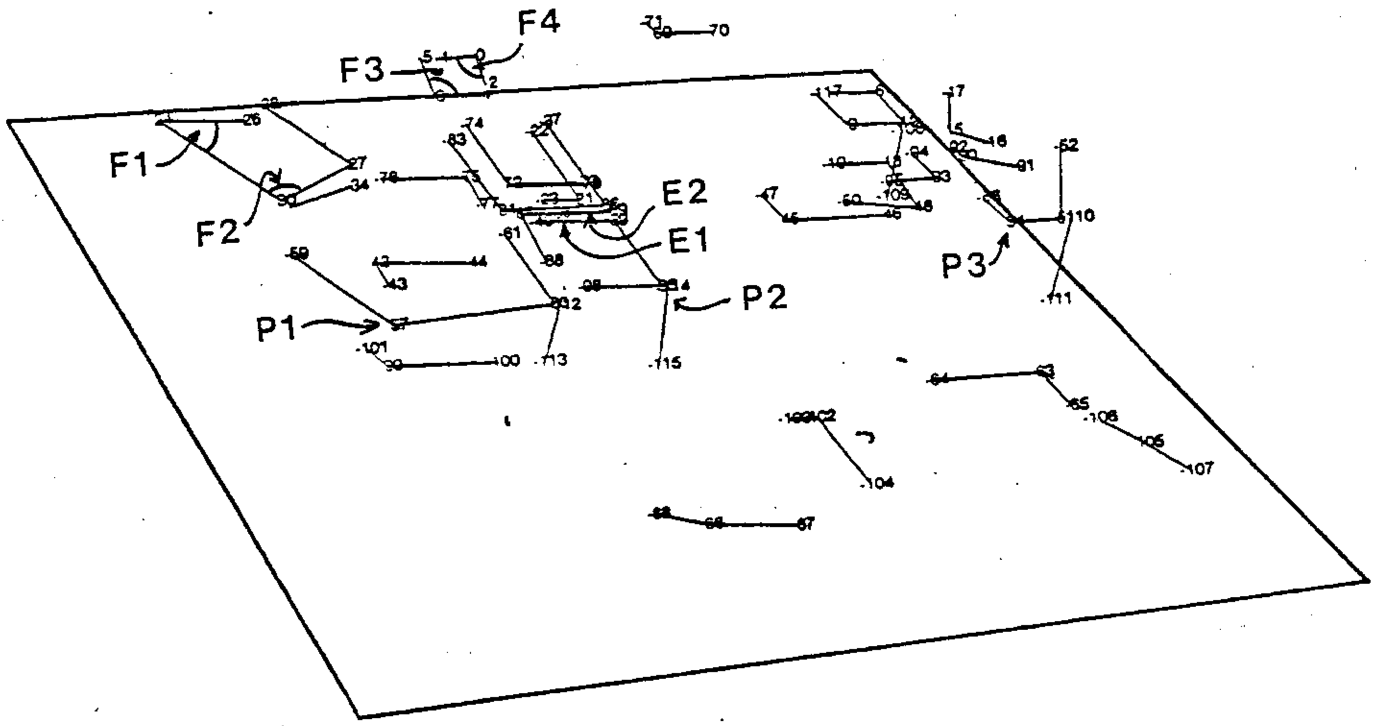


Figure 12: Perspective view of 3D vertices and edges extracted from stereo pair in Fig. 1.

corner. The latter situation results when the vertex is part of a hole in the face. In general, the side on which the web face lies is not known at creation time.

After all web faces have been created, those that represent corners of a single face are merged, as explained next.

6.3. Merging Partial Faces

A face is partial if it is not complete, i.e., all of its edge-groups do not form closed edge rings. One way to complete a partial face is to merge it with nearby partial faces which represent different portions of the same face. The procedure that merges two nearby partial faces distinguishes two situations: (1) two faces that are touching, i.e., they share an edge (e.g., F1 and F2 in Fig. 12), or (2) two faces that are not touching (e.g., F3 and F4 in Fig 12).

Two partial faces that touch each other are merged if they satisfy the following conditions:

1. They must share exactly one edge (by definition of touching). Fig 15a depicts two partial faces, *f1* and *f2*, that share the edge *e2*.

2. The shared edge must serve as a boundary of both faces, but cannot partition them. This condition is satisfied if none of the vertices shared by the two faces lie on two edges of each face. In Fig. 15b,

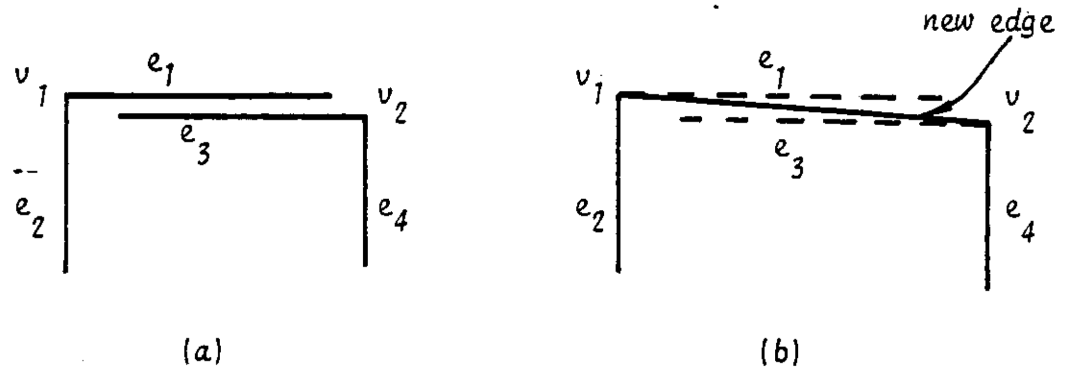


Figure 13: Combining edges. (a) Edges *e1* and *e3* are very close to each other, and each has a confirmed vertex. These vertices are on opposite ends of each other. (b) The new edge is shown as the result of merging *e1* and *e3*.

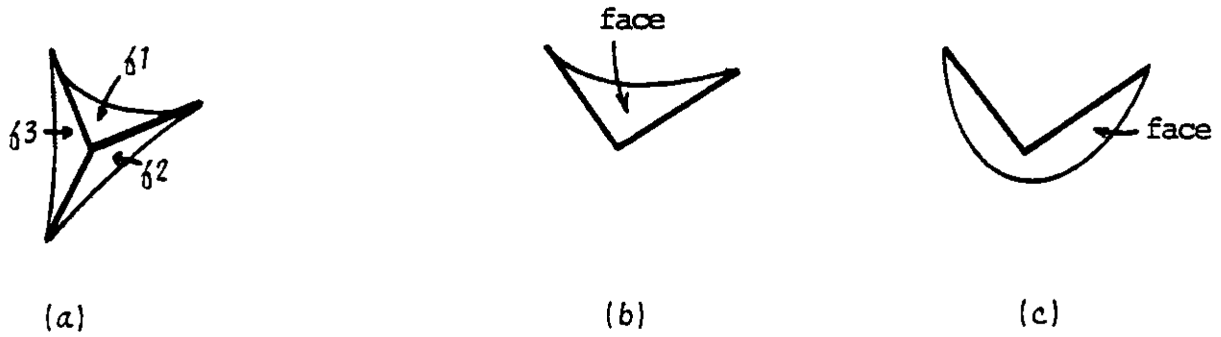


Figure 14: (a) Three web faces generated from a trihedral vertex. A web face may either be on the inside (b) or the outside (c) of a vertex corner.

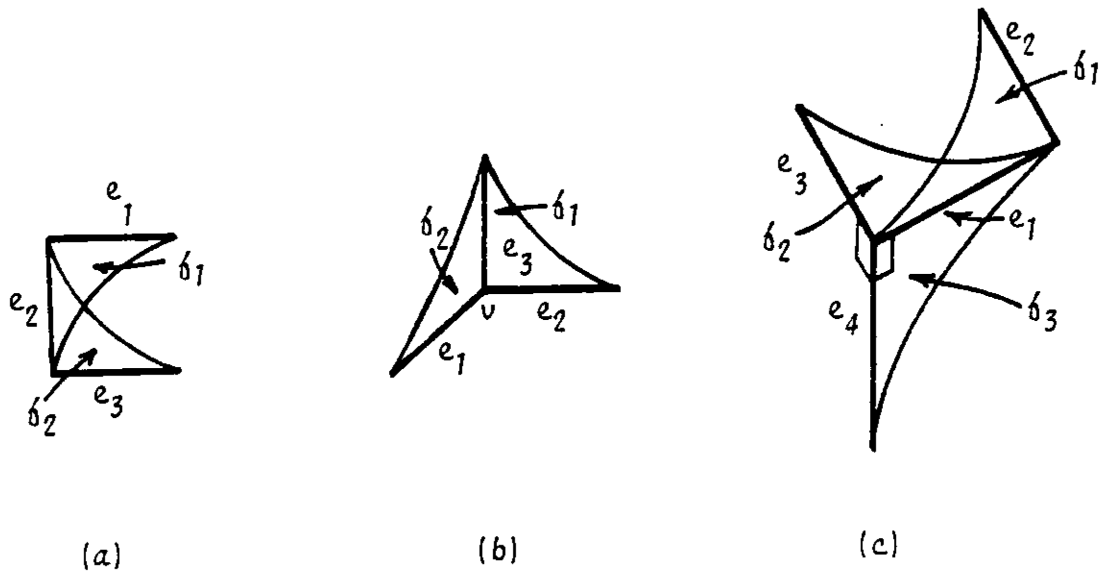


Figure 15: Situations for merging touching partial faces. (a) *f1* and *f2* share one edge. (b) *e3* partitions *f1* and *f2* rather than serving as a boundary for them. (c) *f1* and *f3* share an edge that bounds them, but they are not parallel.

the partial faces *f1* and *f2* share the edge *e3*. These faces should not be merged because they share the vertex *v* which lies on two edges of each face (on *e2* and *e3* of *f1*, and on *e1* and *e3* of *f2*). Notice how *e3* serves to partition the faces, while in Fig. 15a, the edge *e2* serves as a boundary of the faces it joins.

3. The planes of the faces must be nearly parallel and very close to each other. This condition is tested by checking whether all the points lying on one face are within a threshold distance from the plane of the other face. In Fig. 15c, suppose *e4* is perpendicular to both *e1* and *e3*, and *e2* is parallel to *e3*. The partial faces *f1* and *f3* meet both conditions (1) and (2) above, but they do not meet the current condition.

The procedure for merging two touching faces F1 and F2 involves (1) finding the two edge-groups G1 of F1 and G2 of F2 that contain the shared edge, (2) subtracting edges and vertices from G1 (i.e., deleting part-of links in the structure graph) and adding them to G2, (3) subtracting edge-groups, edges, vertices, lines, and points from F1 and adding them to F2, and (4) recalculating the plane equation of F2 as a least squares fit to all the points now constraining F2.

Two partial faces that do not touch each other are merged if they satisfy the following conditions:

1. Each face must have an edge-group containing two non-vertex end points. In Fig. 16a, face *f1* has a single edge-group (consisting of edges *e1* and *e2*) that has the two non-vertex end points *p1* and *p2*. Similarly, face *f2* has a single edge-group with the non-vertex end points *p3* and *p4*.

2. Each of the two end points of the edge-group of one face must be uniquely matched with those of the other face. That is, each end point must be a distance of less than a threshold from exactly one of the two end points of the other face. In Fig. 16a, *p1* and *p3* are uniquely matched because their distance is less than the threshold and the distance from *p1* to *p4* is greater than the threshold. Similarly, *p2* and *p4* are uniquely matched.

3. The planes of the two faces must be nearly parallel and within a small threshold distance of one another.

The procedure for merging two non-touching faces is similar to the one for merging touching faces, in that elements are subtracted from one face and added to the other face, and the plane equation of the resulting face is recalculated. An additional step, however, involves finding the point of intersection of each pair of edges on which the matching pairs of end points lie. The points are then converted into new hypothesized vertices on the edges. The result of merging *f1* and *f2* in Fig. 16a is shown in (b), where two new vertices, *v1* and *v2*, have been hypothesized. Notice that the edge *e1* has been shortened in the process, while the other edges have been extended.

Up till now, we have only discussed the merging of partial faces. However, if the confirmed parts of two faces, each of which may be partial or complete, satisfy the three conditions outlined above for merging non-touching faces, then the faces may be merged. For example, suppose the face *f1* in Fig. 16c contains the

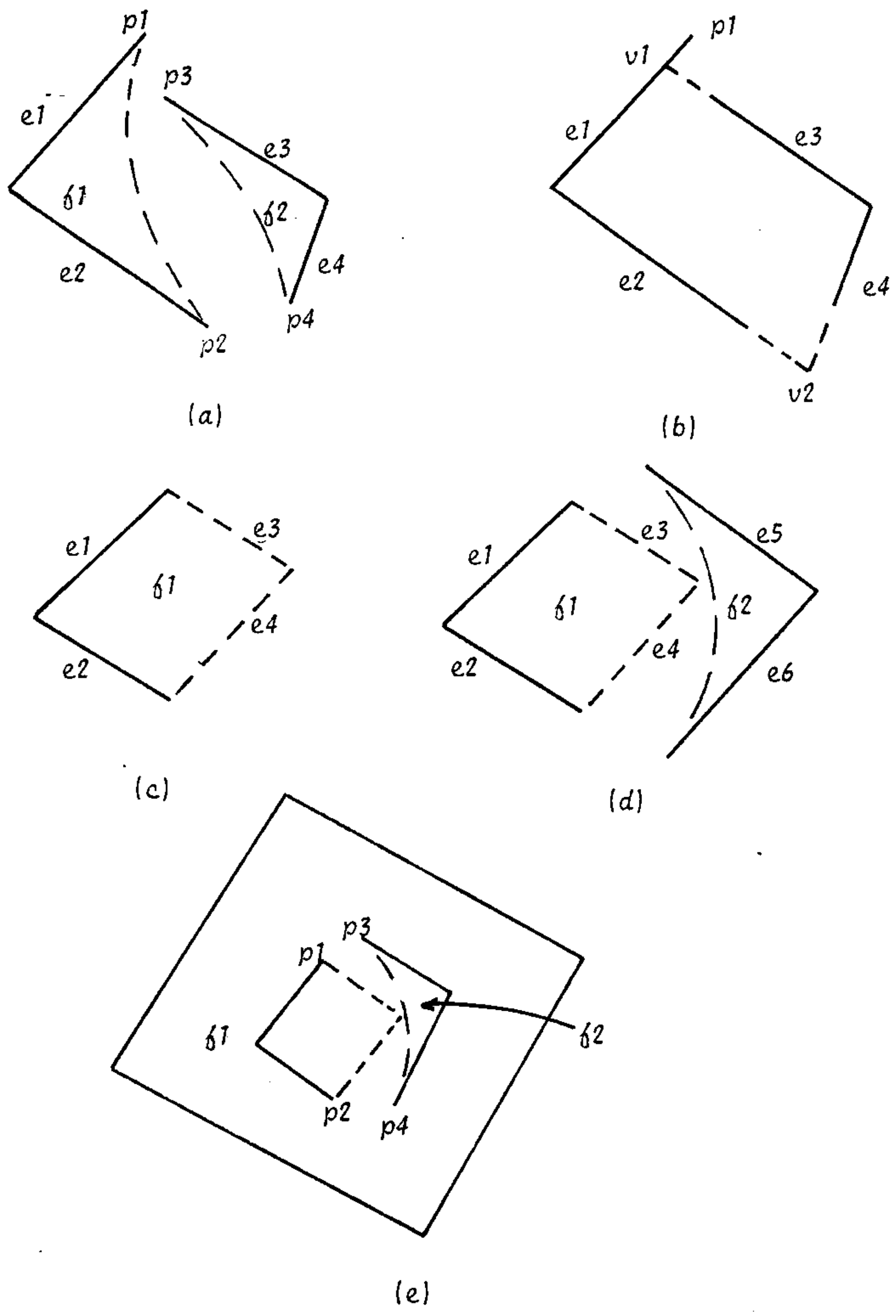


Figure 16: Merging of non-touching faces. (a) *f1* and *f2* satisfy the conditions for merging. (b) Result of merging *f1* and *f2*. (c) and (d) The complete face *f1* is merged with the partial face *f2*. (e) The complete face *f1*, which contains a hole, is merged with the partial face *f2*.

confirmed edges *e1* and *e2* and the hypothesized edges *e3* and *e4*. Now suppose that the web face *f2* in (d) is new information that becomes available, say, from a new view. The confirmed parts of *f1* may then be merged with *f2* if they satisfy the conditions for merging. In the process, hypothesized parts of *f1* must be deleted. The mechanisms for doing this will be discussed later.

Another interesting example is depicted in Fig. 16e, whose situation is similar to that in (d) except that the web face *f2* is merged with confirmed parts of the face *f1*, which has a hole in it. Notice that the condition that the confirmed parts of each face must have two end points which are uniquely matched to those of the other face is satisfied by *p1* and *p3*, and by *p2* and *p4*. As a result of merging, *f2* aids in completing the boundary of the hole in *f1*.

After all mergers have been performed, many faces may still be incomplete. As will be explained later, knowledge of urban scenes is used to hypothesize the shapes of such faces, and they are completed by generating the appropriate edges and vertices.

6.4. Finding and Constructing Holes in Faces

The procedure for finding and constructing holes in faces occurs after all faces have been completed. The face F1 is assumed to represent a hole in the face F2 if the following conditions are satisfied:

1. The planes of the two faces are nearly parallel and within a small threshold distance of one another.

2. The bounding ring of vertices of F1, when projected onto the plane of F2, falls inside the boundary of F2.

If these conditions are satisfied, the edge-group that contains the bounding edges of F1 is subtracted from F1 and added to F2. It now serves as an inner edge-group (an inner ring of edges) of F2. F1 is then deleted from the structure graph.

7. Knowledge of Urban Scenes

Because the wire-frame data extracted from images represent a partial and sparse description of the scene, knowledge of planar-faced objects by itself is generally not adequate for completing many of the objects in the model. As will be described next, knowledge of urban scenes that contain block-shaped objects has been useful for this task.

7.1. Completing Shapes of Faces

Faces in the model may be incomplete because they contain one or more incomplete edge-groups, i.e., edge-groups without closed rings of edges. In these cases, the shape of each incomplete edge-group is hypothesized, and it is completed by generating the appropriate edges and vertices. The following rules are used here:

1. If the partial edge-group represents a single corner, i.e., it contains only two connected edges (the solid lines of face *f* in Fig. 17a), the shape is completed as a parallelogram. Two new edges are hypothesized to complete the shape and are added to the edge-group (dashed lines in the figure).

2. If the partial edge-group consists of three or more edges connected as a single chain (the solid lines of face *f* in Fig. 17b), the shape is completed by connecting the two end points of the chain with a new, hypothesized edge (dashed line in the figure), and adding it to the edge-group.

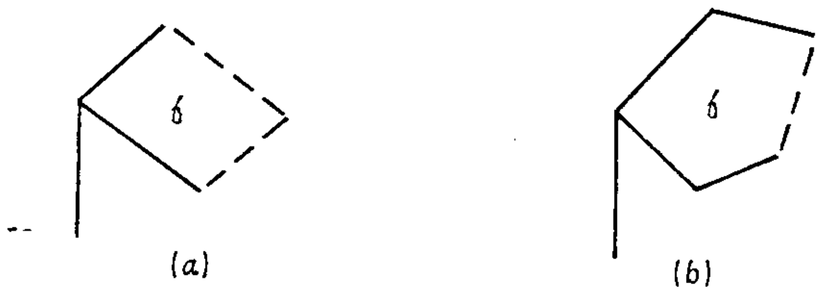


Figure 17: Completing shapes of faces. (a) The face *f* is completed in the shape of a parallelogram. (b) The face *f* is completed by closing the shape.

7.2. Hypothesizing Vertical Faces for Incomplete Objects

Objects in the model may be incomplete because they do not consist of a completely closed, connected set of faces. Since we are dealing with urban scenes, faces that lie high enough above the ground plane are assumed to represent roofs of buildings. A hypothesized vertical wall is dropped toward the ground from each edge of such faces, unless the edge is already part of another face.

The test to determine whether the face is high enough above the ground involves checking whether all the points on the face exceed a threshold distance from the ground. This test rules out faces that intersect the ground (such as building walls) or faces that lie on the ground (such as ground patches). The equation of the ground plane is currently interactively obtained.

A vertical wall is dropped either to the ground plane or to the first face it intersects on the way down. For example, in Fig. 18a, face *f2* is above *f1*, and the distance of each from the ground plane exceeds the threshold. The result after dropping vertical faces is shown in (b), which indicates that faces have been dropped from *f1* to the ground, and from *f2* to *f1*.

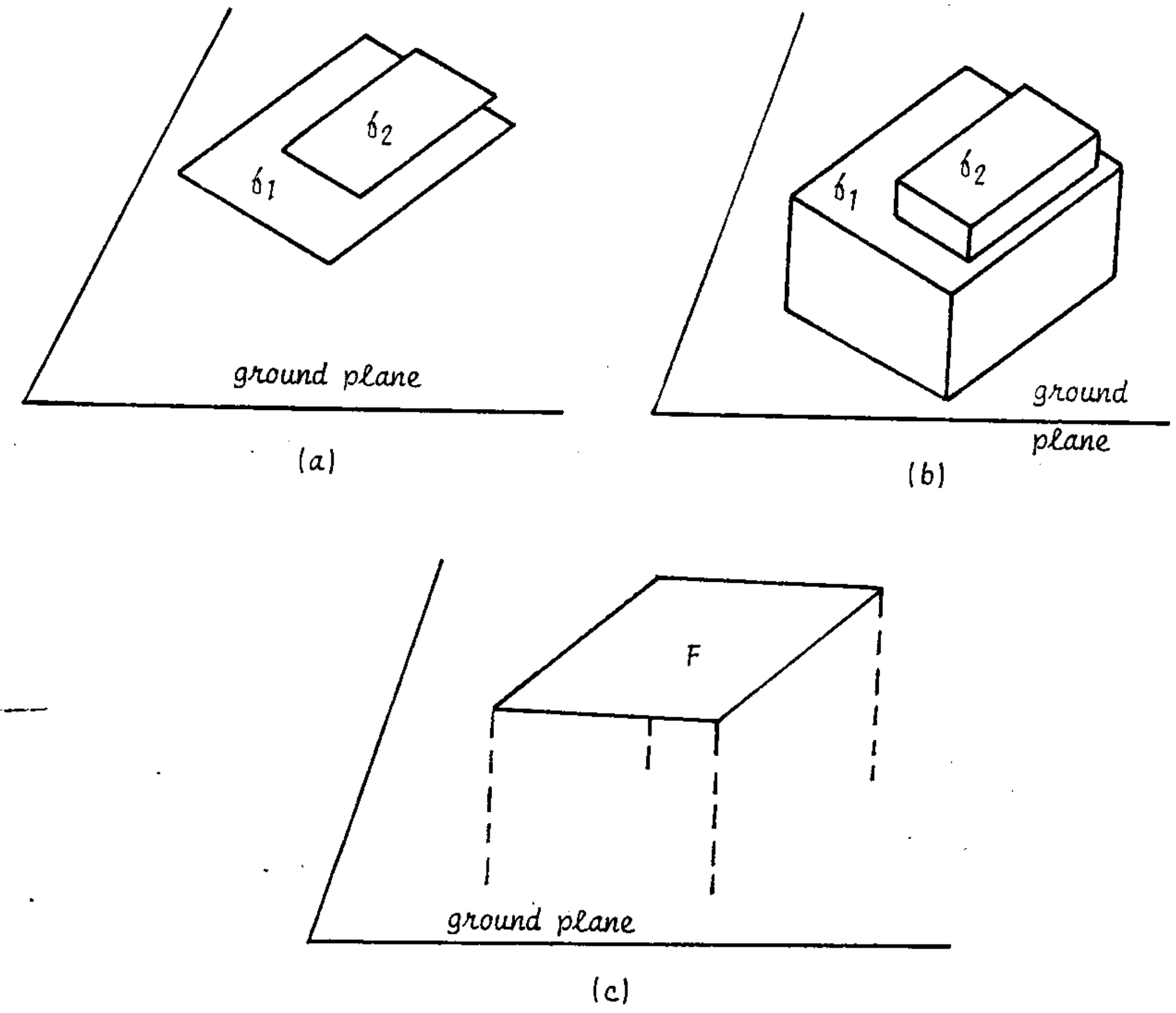


Figure 18: Dropping vertical walls from faces. (a) The face *f2* is above *f1*. (b) Faces are dropped from *f1* to the ground plane, and from *f2* to *f1*. (c) A vertical edge-frame is dropped from the face F.

The procedure for dropping vertical faces from a face F is as follows. For each vertex of F that has fewer than three legs, an edge is dropped either to the ground plane or to the first face it intersects. This results in a *vertical edge-frame* that supports F (the dotted lines in Fig. 18c). The edge-frame is then "filled in" by first creating web faces for each new edge pair at each vertex of F, then merging those that touch each other, and finally completing the resulting partial faces in the ways described earlier.

When the techniques described above are applied to the output of the stereo analysis component depicted in Fig. 12, we obtain the scene model shown in Fig. 19. Notice that one of the buildings has a hole in it, through the roof. The planar patches at the "front" of the scene are part of the ground. Because they were not high enough above the ground plane, they were not treated as building roofs. When these techniques are applied to the output of the monocular analysis component (Fig. 20), we obtain the scene model shown in Fig. 21. Note that all vertices, edges, and faces which have been hypothesized by the procedures described above are marked as such, and will be replaced by more correct versions as more information becomes available from new views.

8. Combining New Views with Current Model

The process of incorporating a 3D wire-frame description extracted from a new view into the current scene model can be divided into three main steps:

1. The wire-frame data must first be matched to the current model. This process provides (a) the scale transformation and coordinate transformation from the wire-frame data to the model, and (b) corresponding elements (i.e., vertices and edges) in the two.

2. The new wire-frame data is then merged with the current model. This process includes (a) merging pairs of corresponding elements, and (b) adding to the model wire-frame elements for which no correspondences were found. The latter procedure is aided by knowledge of the scale and coordinate transformations. During the merging process, hypothesized parts of the model that are inconsistent with the new wire-frame data are deleted.

3. At this point, many objects in the model may be incomplete because (a) new wire-frame data has been added, and/or (b) some hypothesized elements have been deleted. These objects are completed using the techniques described in the previous sections.

To see how these steps are carried out, consider the example of incorporating the information from a second view into the scene model of Fig. 19. This scene model was constructed from the set of wire frames (Fig. 12) automatically extracted from a "front" view of the scene (Fig. 1). The second set of wire frames, shown in Fig. 22, was manually generated to simulate information available from an opposing point of view (viewing the scene from the "back"). The viewpoint for the perspective drawing of Fig. 22 is chosen to be similar to that of Fig. 12 to allow easier comparison by the reader. Notice that the information in Fig. 12 emphasizes edges and vertices facing the front of the scene, while those facing the back of the scene are emphasized in Fig. 22.

8.1. Matching

We assume in this example that the scale and coordinate transformations from the new wire-frame data to the current model is known; the data and model may therefore be described in the same coordinate system. We have not yet implemented a general matcher that provides these transformations between the two.

The next step is to determine corresponding edges and vertices in the data and model. First we label each connected group of edges in the wire-frame data as a distinct wire-frame object. Next, wire-frame objects are matched with model objects. Two objects are said to match if they have confirmed parts that match. Matches

Figure 19: Perspective views of buildings reconstructed from wire-frame data in Fig. 12. These buildings correspond to the cluster of buildings at the upper middle parts in the images of Fig. 1.

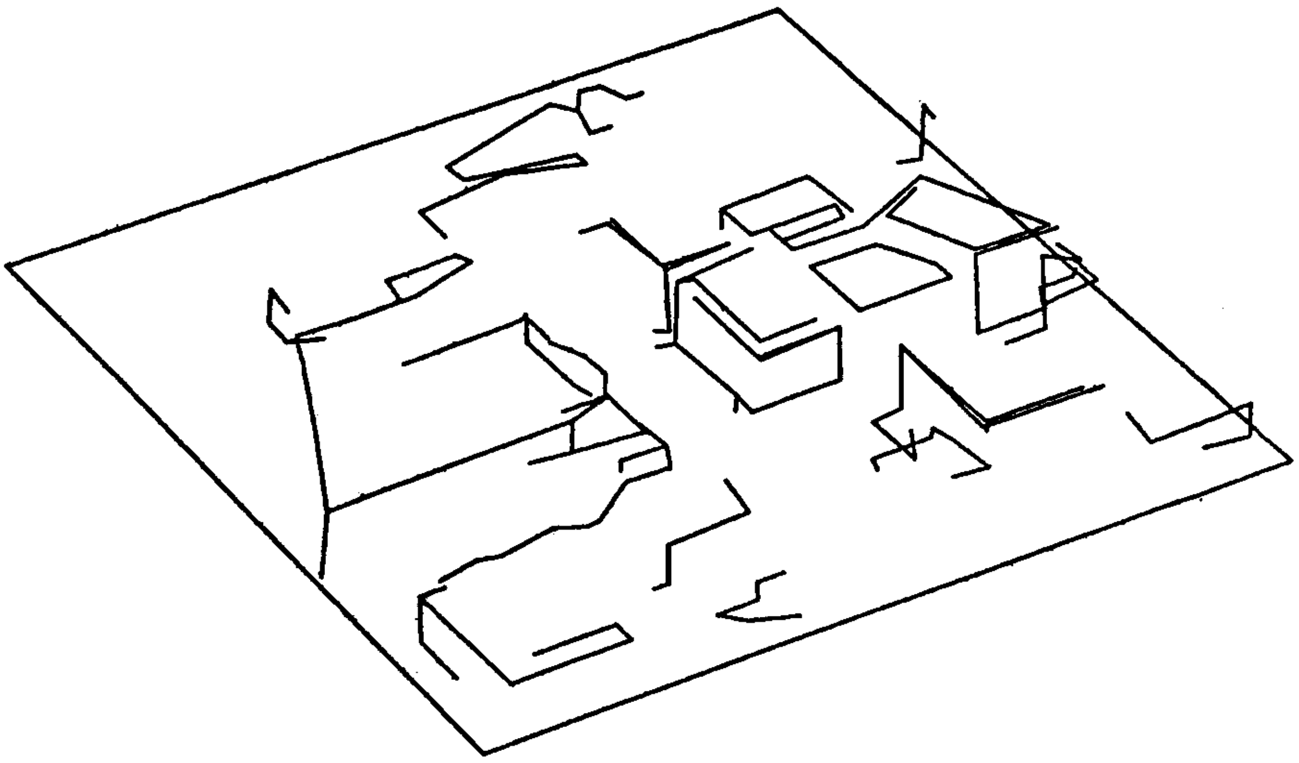


Figure 20: Perspective view of 3D vertices and edges extracted from image in Fig. 2.

are sought only for edges and vertices, since these constitute the only confirmed parts of a wire-frame object. The requirements for two confirmed vertices, one from each object, to match are: (1) they must be very close to each other, or (2) they must be part of matching edges whose other two vertices match. The requirements for two confirmed edges, one from each object, to match are: (1) the two confirmed vertices of one edge must match the two of the other, or (2) one confirmed vertex on one edge matches one on the other, and the two edges are close together and overlap in their lengths. These rules are used in a relaxation algorithm to obtain matching vertices and edges.

As an example, consider Fig. 24. Suppose the object in (a) is part of the model. The edges represented by the solid lines *e1, e2, e3, e4* and *e12*, are confirmed. The edges represented by the dashed lines are hypothesized. Vertices *v1, v2* and *v3* are confirmed, while the others are hypothesized. (Note that there are also edges and vertices in this object hidden from our viewpoint; these will not be considered here.) Suppose the wire-frame object in Fig. 24b has been derived from a new view, and it has been transformed to register with the model object. The following algorithm is used to match the two.

1. Find pairs of confirmed vertices that match by determining which ones lie within a threshold distance of one another. The vertices *v2* and *v100* are found to match, but let us suppose the distance between *v3* and *v101* exceeds the threshold.

2. Find pairs of confirmed matching edges that contain previously found matching vertices. The

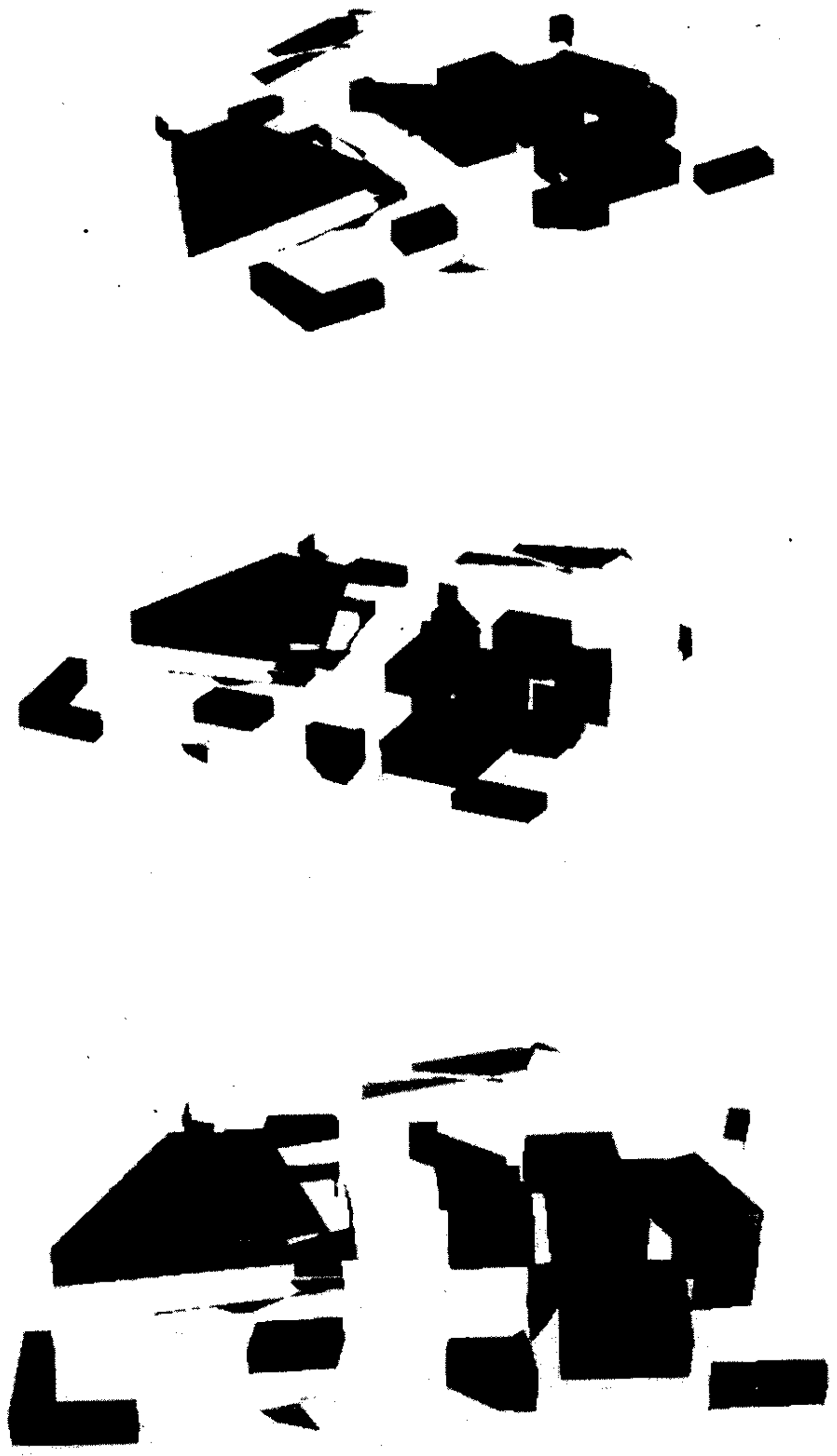


Figure 21: Perspective views of buildings reconstructed from wire-frame data in Fig. 20.

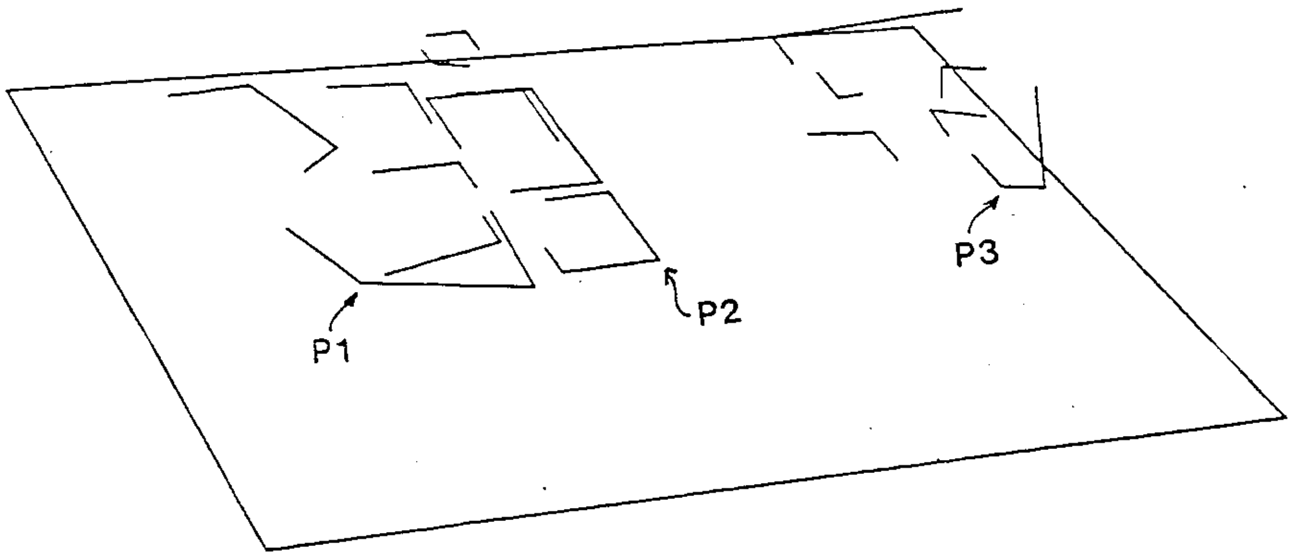


Figure 22: Perspective view of manually generated vertices and edges which simulate information available from images showing an opposite point of view from that shown in Fig. 1. The viewpoint for this drawing is chosen to be similar to Fig. 12. Points P1, P2, and P3, for example, correspond to points P1, P2, and P3 in Fig. 12.

edges *e2, e3* and *e100, e101* contain matching vertices and are therefore compared. In order to match, two edges must be very close together and must overlap in their lengths. The distance threshold for this test, however, is greater than the one for determining matching vertices. This is permitted because the possible matching edges are also constrained by the requirement that they contain matching vertices. Therefore, even though *v3* and *v101* failed to match in step (1) above, the edges *e3* and *e101* are found to match, as are *e2* and *e100.*

3. For each new matching pair of edges found, if they contain a single pair of matching vertices, match their other vertices (if they exist and are confirmed). The vertices *v3* and *v101* match because *e3* and *e101* match. No new matching vertices result from the matching edges *e2* and *e100,* since *e100* has only one vertex.

4. Proceed by repeating step (2) above, i.e., find new pairs of confirmed matching edges that contain previously found matching vertices. The edges *e4* and *e12* are compared with *e102* and *e104.* Using the distance and overlap tests, *e4* and *e102,* as well as *e12* and *e104,* are found to match.

5. Next, step (3) is repeated. New matching vertices are sought that lie on newly found matching pairs of edges. The matching edges found in step (4) contain no new matching vertices, since *v4* and *v6* are unconfirmed. The algorithm therefore halts at this point; it would have continued with step (2) if new matching vertices had been found. The following pairs of matches are returned: *(v2, v100), (v3, v101), (e2, e100), (e3, e101), (e4, e102), (e12, e104).*

8.2. Discrepancies

We must now merge the new wire-frame data into the model. An important issue here is how to handle discrepancies between the two. We consider the following two types of discrepancies:

1. After the coordinate system of the wire-frame data has been transformed to that of the model and scale adjustments have been made, corresponding pairs of confirmed vertices and edges may not register perfectly in 3-space. In order to merge them into single elements, we perform a "weighted averaging" of their positions.

2. Hypothesized elements in the model may be inconsistent with newly obtained elements. We handle this by deleting such hypothesized elements.

To determine whether or not hypotheses are still valid when confirmed elements in the model are modified or deleted, we consider the elements which gave rise to the hypotheses. A hypothesis is dependent on all elements whose existence directly resulted in the creation of the hypothesis. If one of these elements is modified or deleted, the hypothesis must also be modified or deleted since the conditions under which it was created are no longer valid. The dependency relationships for hypothesized elements are explicitly recorded at the time of their creation using dependency pointers [Doyle 81].

We currently record these relationships for the following situations:

1. When two non-touching partial faces are merged (Fig. 23a), each face has two partial edges which are intersected with their counterparts in the other face. The intersection points form two new hypothesized vertices, each of which is dependent on the two edges whose intersection gave rise to it. In Fig. 23a, the arrows indicate the dependencies. Vertex *v1* is dependent on edges *e1* and *e3*, and vertex *v2* is dependent on edges *e2* and *e4*. If one of the edges were to be modified (e.g., if its position were to be displaced), the vertex that depends on that edge would no longer be a valid hypothesis, and would therefore be deleted. A new vertex might then be hypothesized.

2. When an incomplete edge-group is completed in the shape of a parallelogram (Fig. 23b), two new edges and three new vertices are hypothesized. Each of the new edges *e3* and *e4* is dependent on both of the old edges *e1* and *e2*. The edge *e3*, for example, is dependent on *e1* in the sense that its end point is constrained by the end point of *e1*. It is dependent on *e2* in the sense that it is constrained to be parallel to *e2*. The new vertex *v3* is dependent on the two hypothesized edges *e3* and *e4*, while the new vertices *v1* and *v2* are dependent on the confirmed edges on which they lie.

3. When a face is completed by connecting its two end points (Fig. 23c), two new vertices and one new edge are hypothesized. The new edge *e4* is dependent on both *e1* and *e3*, while the new vertices *v1* and *v2* are dependent on the edges on which they lie.

4. When a vertical wall is dropped from a face, the first step is to drop hypothesized edges from vertices of the face. Such edges are dependent on the vertices from which they are dropped. In Fig. 23d, the new edges *e1* and *e2* are dropped from, and are dependent on, the vertices *v1* and *v2*, respectively. A dropped edge is constrained to be perpendicular to the ground plane, and would therefore no longer be a valid hypothesis if the vertex it depends on, which is one of its end points, were to be displaced. After edges are dropped from all vertices of the face, the resulting edge-

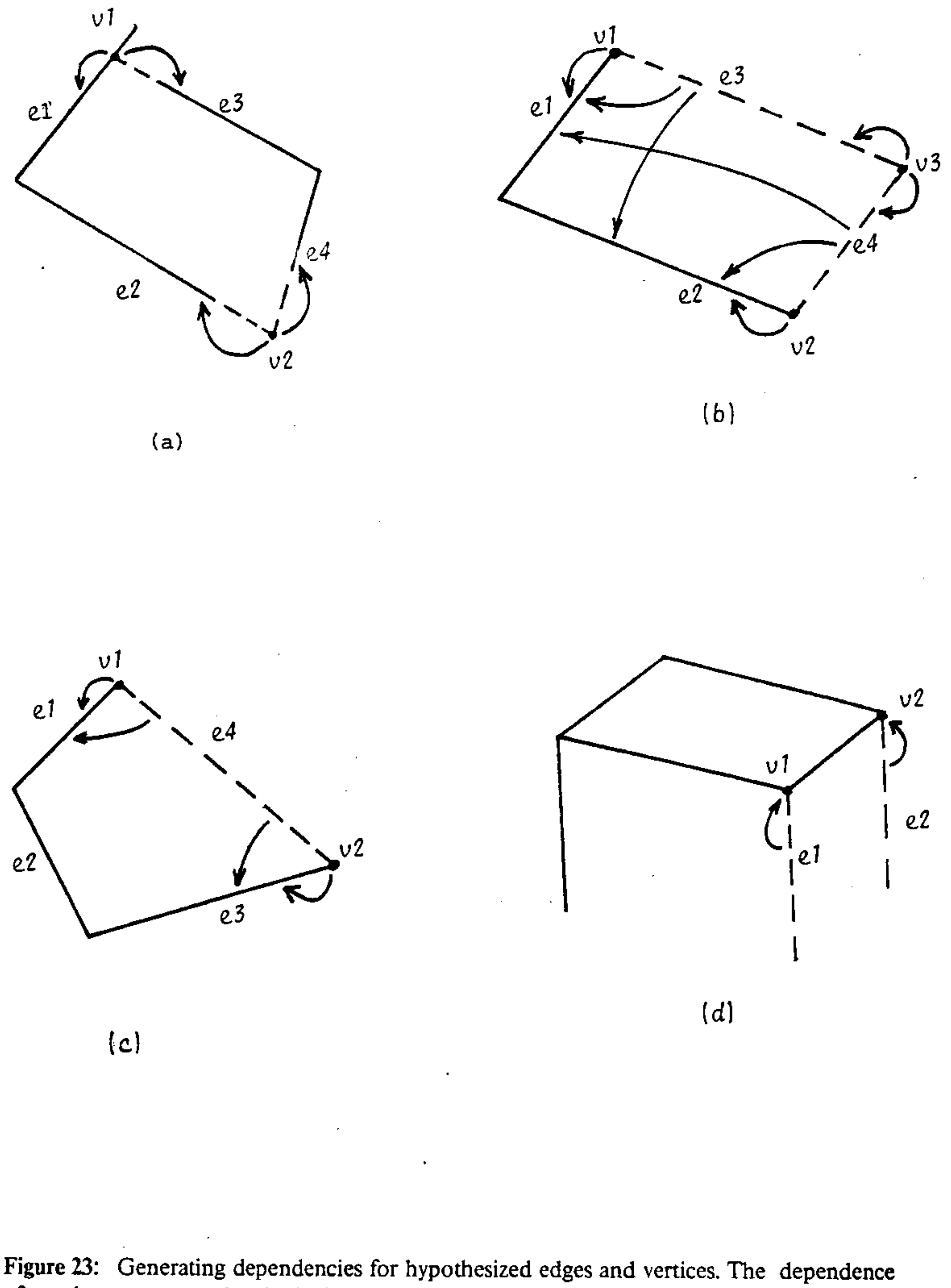


Figure 23: Generating dependencies for hypothesized edges and vertices. The dependence of an element on another is depicted as an arrow from the former to the latter. (a) Two non-touching partial faces are merged. (b) A face is completed in the shape of a parallelogram. (c) A face is completed by connecting its two end points. (d) Vertical edges are dropped from a floating face.

frame is filled in with faces, as described previously. This results in more hypothesized edges and vertices. The situations under which these are created fall under categories (2) and (3) above.

When a confirmed edge or vertex in the model is modified or deleted, the set of all hypothesized elements that depend on it are deleted. Recursively, elements depending on deleted ones are also deleted. When hypothesized vertices and edges are deleted in this manner, it is possible for hypothesized faces to lose minimal support, i.e., they may no longer be constrained by at least three non-colinear points. Such faces are also deleted.

8.3. Merging

The procedure that merges corresponding wire-frame and model objects takes into account the fact that the 3-space positions of end points of edges that are confirmed vertices are generally much more accurate than the positions of non-vertex end points. Therefore, confirmed vertices are given more weight during merging. As an example, consider again Fig. 24, where the wire-frame object in (b) is to be merged with the model object in (a).

The merging procedure starts by merging corresponding vertices in the two objects. This involves the following:

1. The model vertex of each corresponding pair of vertices (*(v2, v100)* and *(v3, v101)* in Fig. 24) is assigned new coordinates -- those of the midpoint of the line connecting the two initial vertices.

2. If the distance between the initial and resulting points of the model vertex exceeds a threshold, all hypothesized edges and vertices in the model that recursively depend on this vertex are deleted. Hypothesized faces that have lost minimal support are also deleted.

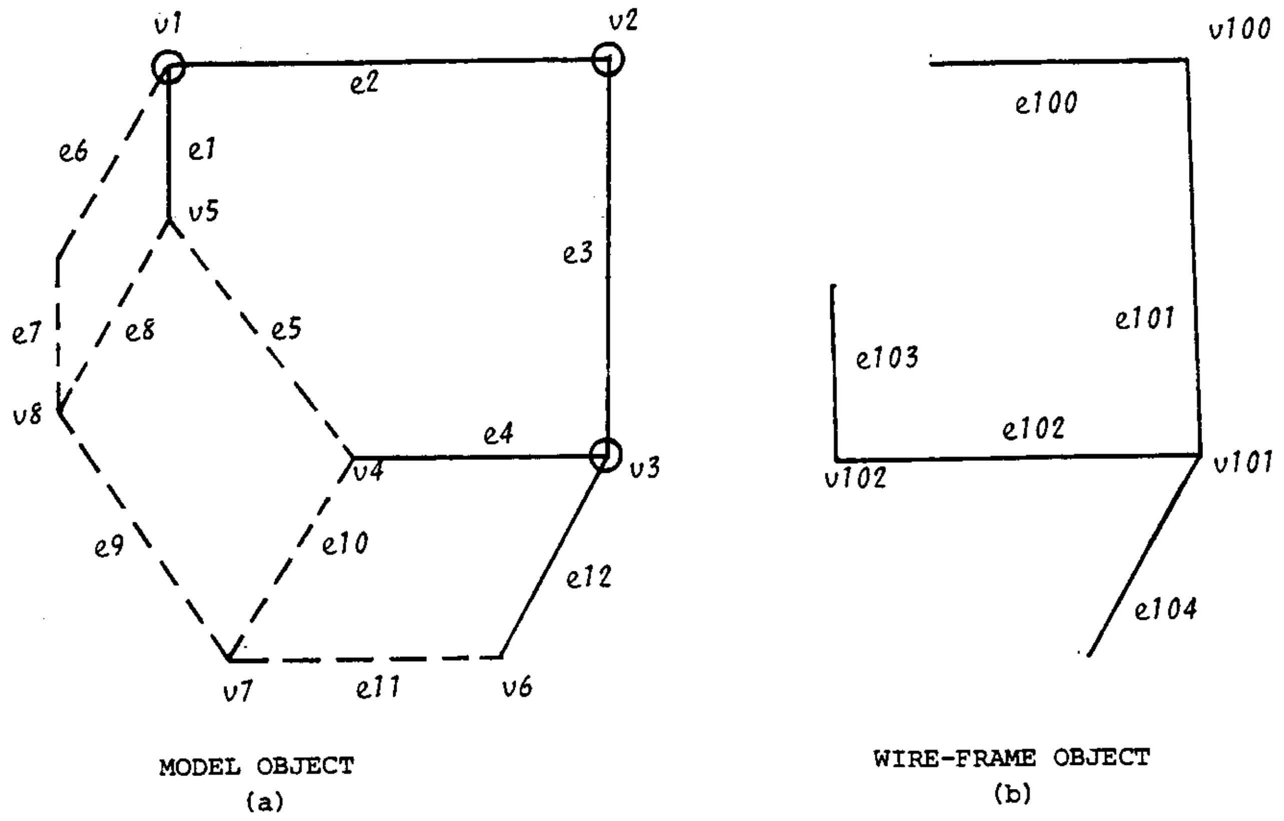
3. The vertex in the wire-frame object is deleted and replaced by the model vertex.

At this point, all corresponding pairs of edges will share at least one vertex. The corresponding edges are merged next as follows:

1. If the two edges share both their vertices (Fig. 25a), merging involves recalculating the line equation of the model edge and deleting the wire-frame edge. In Fig. 24 this situation occurs between edges *e3* and *e101*.

2. If the two edges share one vertex but only one of them contains another confirmed vertex (Fig. 25b), the edge with one confirmed vertex is deleted, leaving the edge with two vertices as the result. In Fig. 25b, the result of merging *e1* and *e2* is *e1*. Notice that the non-vertex end point in this case is given zero weight. If the resulting edge is from the wire-frame object, it is subtracted from this object and added to the model object. In Fig. 24, this situation occurs between edges *e2* and *e100*, and edges *e4* and *e102*.

3. If the two edges share one vertex and the other end points are not confirmed vertices (Fig. 25c),



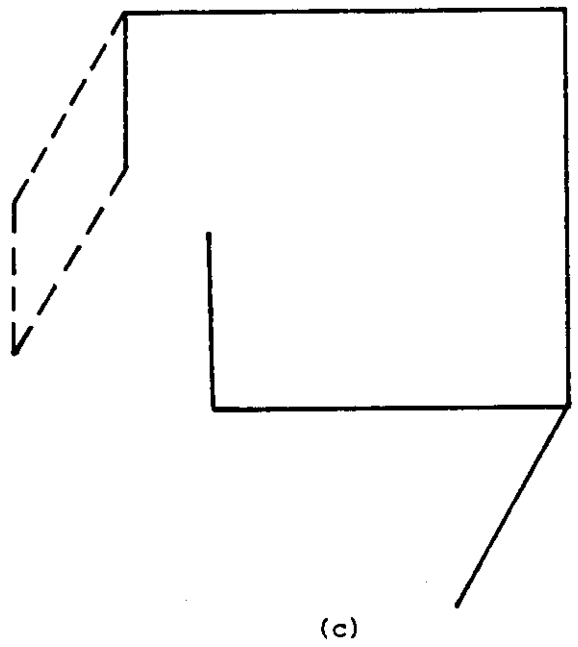


Figure 24: The wire-frame object in (b) is to be merged with the model object in (a). The confirmed edges of the model object (indicated by solid lines) are *e1, e2, e3, e4*, and *e12*; the confirmed vertices (indicated by circles) are *v1, v2*, and *v3*. Dashed lines represent hypothesized edges. (c) The result after merging.

the line equation of the new edge is obtained by a least squares fit to all the points on the two initial edges (see Fig. 25d, where the dotted lines are the initial edges, and the solid line is the new edge). The non-vertex end point of the new edge is the projection of the non-vertex end point of the longest initial edge onto the line constraining the new edge. This new end point is labeled as confirmed. The edge is then added to the model object and the two initial edges are deleted. Note that the vertex end point of this edge need not lie on the line constraining the edge. In Fig. 24, this situation occurs between edges *e12* and *e104*.

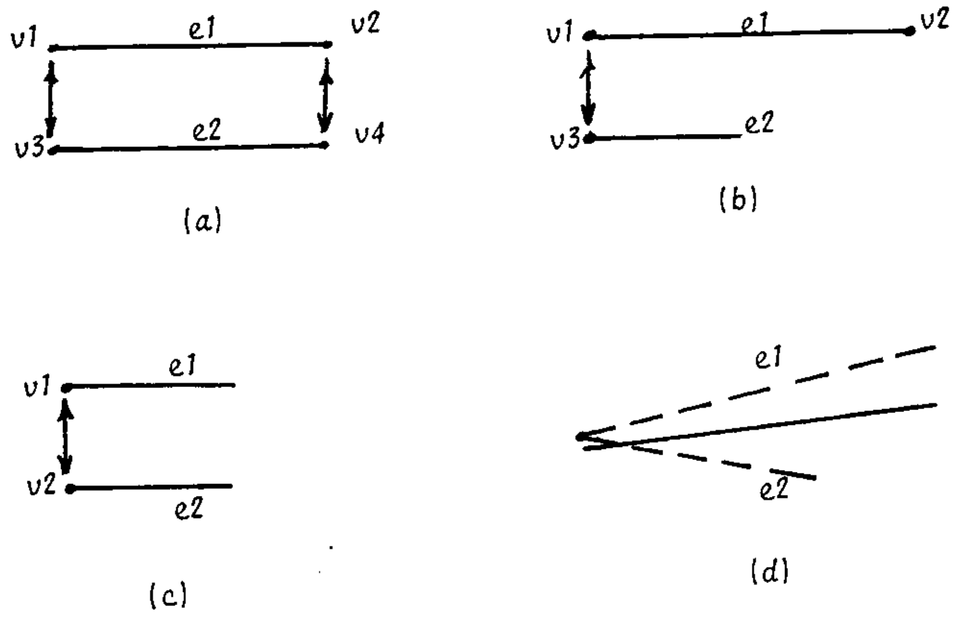


Figure 25: Merging edges. Two edges to be merged may either (a) share both their vertices, or (b) and (c) share one vertex. (d) Result of merging edges in situation (c).

Before merging, a model edge may contain either one confirmed vertex or two confirmed vertices. If it contains one confirmed vertex, then all hypothesized edges and vertices in the model that recursively depend on this edge are deleted. Hypothesized faces that have lost minimal support are also deleted. In Fig. 24, this occurs for the edges *e4* and *e12*. The hypothesized elements in the figure that recursively depend on, say, *e4* are the vertices *v4* and *v7*, and the edges *e5, e10, e9* and *e11*. If a model edge to be merged contains two confirmed vertices (e.g., *e2* and *e3* in Fig. 24), no hypothesized elements need be deleted since all necessary deletions were made when the vertices of the edge were merged.

After all corresponding elements of the two objects have been merged, the edges and vertices remaining in the wire-frame object that were not merged are added to the model object, and the wire-frame object is deleted. In Fig. 24, this step involves adding the edge *e103* to the model.

Finally, the plane equation is recalculated for each face in the model object which had edges and vertices that were modified or deleted. Fig. 24c shows the final configuration of the object after the merging process.

This object is incomplete and must be completed using the techniques described in previous sections.

8.4. Results of Merging

When these procedures are applied to the wire-frame data in Fig. 22 and the scene model in Fig. 19, we obtain the updated scene model shown in Fig. 26. The updated version has two important improvements over the initial version. First, the updated model contains more buildings since new wire-frame data, some of which represent new buildings, have been incorporated into the initial model. Second, for many buildings described in both versions of the model, the positions of vertices and edges are more accurate in the updated version. This is because many hypothesized vertices and edges are replaced by accurate ones obtained from the new data, and many confirmed vertices and edges are merged with corresponding ones in the data by "averaging" their positions, generally decreasing the amount of error.

The shape of the large hole in the roof of one of the buildings has changed from a rectangle in the initial model to an almost triangular quadrilateral in the updated version. When compared with the source images in Fig. 1, the rectangular shape would seem more accurate. However, the positions of the edges and vertices that form the hole are more accurate in the updated model in the sense that they are more faithful to the wire-frame descriptions derived from the images.

This experiment demonstrates how information provided by each additional view allows the model to be incrementally made more complete and accurate.

9. Summary

The 3D Mosaic system is a vision system that incrementally acquires a 3D model of a complex scene from multiple images. This paper has focused on the representation, construction, and updating of the model. Each view of the scene undergoes either monocular or stereo analysis. This results in a 3D wire-frame description that represents portions of edges and vertices of objects in the scene. The model is incrementally constructed and updated from the wire frames using task-specific knowledge. This process involves generating, adding, and deleting hypotheses about the structure of the scene. At any point along its development, the model represents the current understanding of the scene and may be used for tasks such as matching, display generation, planning paths through the scene, and making other decisions dealing with the scene environment. Examples have been presented showing how the system interprets complex aerial photographs of the Washington, D.C. area.

Acknowledgement

Takeo Kanade, Fumi Komura, Shigeru Kuroe, and Duane Williams have worked with me on this project and have provided much support and encouragement.

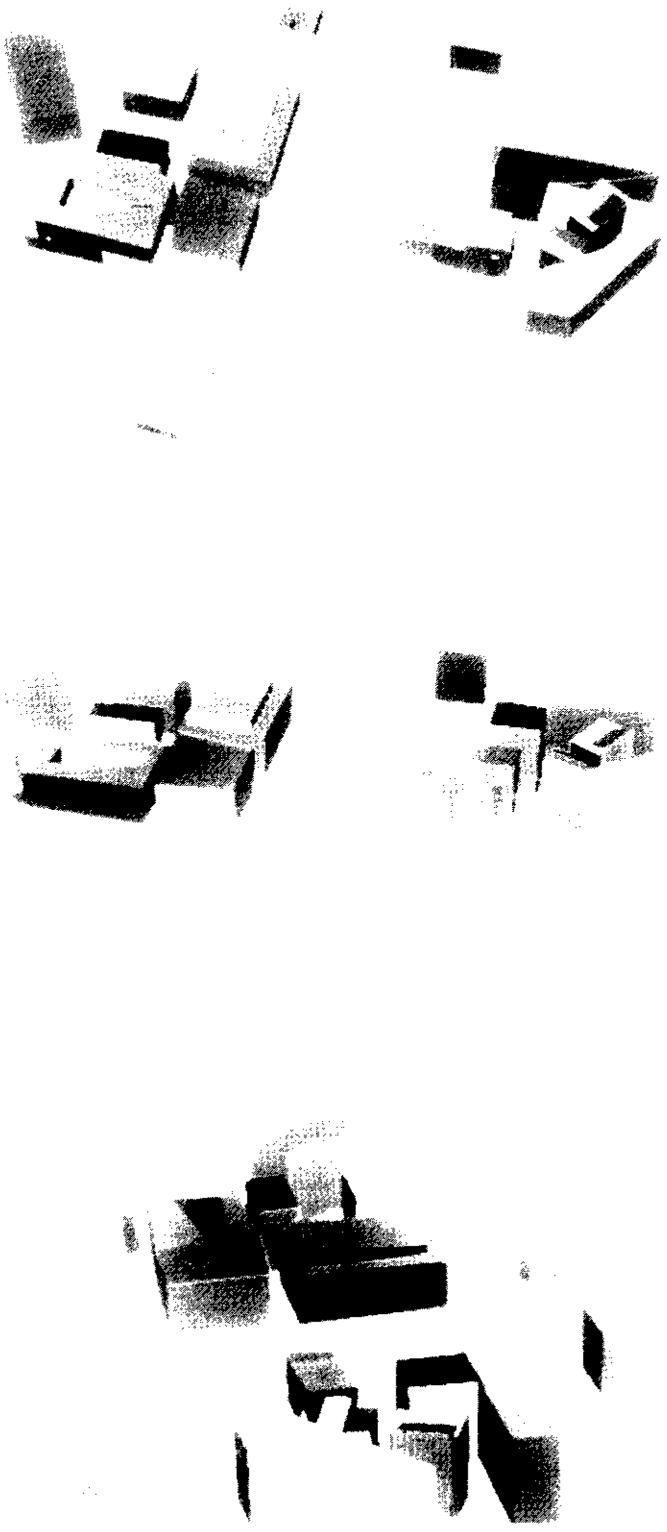


Figure 26: Perspective views of buildings derived by incorporating the wire-frame data in Fig. 22 into the model in Fig. 19.

References

[Baer, Eastman, and Henrion 79]
Baer, A., Eastman, C., and Henrion, M.
Geometric Modelling: a Survey.
Computer-Aided Design 11:253-272, September, 1979.

[Baker 77] Baker, H. H.
Three-Dimensional Modelling.
Proc. IJCAI-77 :649-655, August, 1977.

[Barrow, Bolles, et al. 77]
Barrow, H. G., Bolles, R. C., Garvey, T. D., Kremers,J. H., Tenenbaum, J.M., and Wolf, H. C.
Experiments in Map-guided Photo Interpretation.
Proc. IJCAI-77 :696, August, 1977.

[Baumgart 74] Baumgart, B. G.
Geometric Modeling for Computer Vision.
Technical Report STAN-CS-74-463, Department of Computer Science, Stanford University, Stanford, CA, October, 1974.

[Bourne, Milligan, and Wright 82]
Bourne, D. A., Milligan, R., Wright, P. K.
Fault Detection in Manufacturing Cells Based on Three Dimensional Visual Information.
Proc. SPIE , May, 1982.

[Doyle 79] Doyle, J.
A Truth Maintenance System.
Artificial Intelligence 12 :231-272, 1979.

[Doyle 81] Doyle, J.
Three Short Essays on Decisions, Reasons, and Logics.
Technical Report STAN-CS-81-864, Department of Computer Science, Stanford University, Stanford, CA, May, 1981.

[Eastman and Preiss 82]
Eastman, C. M., and Preiss, K.
A Unified View of Shape Representation and Geometric Modeling.
Israel Conference on CAD , January, 1982.

[Herman 83] Herman, M.
Monocular Reconstruction of a Complex Urban Scene in the 3D MOSAIC System.
Proc. ARPA Image Understanding Workshop :318-326, June, 1983.

[Herman and Kanade 84]
Herman, M., and Kanade, T.
The 3D MOSAIC Scene Understanding System: Incremental Reconstruction of 3D Scenes from Complex Images.
Technical Report CMU-CS-84-102, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, February, 1984.

[Herman, Kanade, and Kuroe 83]
Herman, M., Kanade, T., and Kuroe, S.
The 3D MOSAIC Scene Understanding System.
Proc. Eighth International Joint Conference on Artificial Intelligence :1108-1112, August, 1983.

[Herman, Kanade, and Kuroe 84]
Herman, M., Kanade, T., and Kuroe, S.
Incremental Acquisition of a Three-Dimensional Scene Model from Images.
IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-6(3):331-340, May, 1984.

[Martin and Aggarwal 83]
Martin, W. N. and Aggarwal, J. K.
Volumetric Descriptions of Objects from Multiple Views.
IEEE Trans. on Pattern Analysis and Machine Intelligence PAMI-5(2):150-158, March, 1983.

[McKeown 83] McKeown, D. M.
MAPS: The Organization of a Spatial Database System Using Imagery, Terrain, and Map Data.
Technical Report CMU-CS-83-136, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, July, 1983.

[Moravec 80] Moravec, H.P.
Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.
Technical Report CMU-RI-TR-3, The Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, September, 1980.

[Potmesil 83] Potmesil, M.
Generating Models of Solid Objects by Matching 3D Surface Segments.
Proc. IJCAI-83 :1089-1093, August, 1983.

[Requicha 80] Requicha, A. A. G.
Representations for Rigid Solids: Theory, Methods, and Systems.
Computing Surveys 12(4):437-464, December, 1980.

[Rubin 80] Rubin, S.
Natural Scene Recognition Using Locus Search.
Computer Graphics and Image Processing 13:298-333, 1980.

[Underwood 75] Underwood, S. A. and Coates, C. L.
Visual Learning from Multiple Views.
IEEE Transactions on Computers (6):651-661, June, 1975.