Proc. 1995 ASME. Intl Computers in Engineering Conf., pp. 699-712

AN EFFICIENT SYSTEM FOR GEOMETRIC ASSEMBLY SEQUENCE GENERATION AND EVALUATION

Bruce Romney
Cyprien Godard
Michael Goldwasser
G. Ramkumar
Computer Science Department
Stanford University
Stanford, California

ABSTRACT

In this paper, we present a software system which can automatically determine how to assemble a product from its parts, given only a geometric description of the assembly. Incorporated into a larger CAD tool, this system, the Stanford Assembly Analysis Tool (STAAT), could thus provide immediate feedback to a team of product designers about the complexity of assembling the product being designed. This would be particularly useful in complex assemblies where each designer may not be fully aware of the impact of his design changes on the assemblability of the product as a whole. STAAT's underlying data structure is an efficient version of the non-directional blocking graph (NDBG), a compact representation of the blocking relationships in an assembly. STAAT implements several techniques using this structure, under a unified approach in which the same software "machinery" can analyze the product under different assembly constraints. In initial experiments conducted on relatively small polyhedral assemblies of 20 to 40 parts and 500 to 1500 faces, using one-step translational motions, STAAT generated assembly sequences much more quickly than did previous NDBG-based systems. We are working now on extending both these results and the underlying theory to more sophisticated cases.

INTRODUCTION

In many industries, the product development process is characterized by insufficient consideration of the assembly process by product designers. As a result, difficulties in assembling a product are often not discovered until a prototype is constructed. This is especially true when a designer working on one subsystem makes changes which inadvertently affect the assemblability or insertability of another subsystem. It would be helpful if the CAD system used to design a product were able to provide quick feedback regarding the difficulty of assembling it. This feature would shorten the cycle time between the enactment of design changes and the discovery of their effects on assemblability. Such a CAD system would also be useful in such diverse areas as robotic assembly, service and maintenance, the production of "do-it-yourself" instructional manuals and videos, and part retrieval and recycling. Furthermore, by imparting to the system some notion of the costs of different assembly steps, a manufacturer could use it to optimize the layout of a factory's assembly lines.

We have taken a major step forward toward the realization of this goal with the creation of STAAT, the Stanford Assembly Analysis Tool. STAAT is a stand-alone software system able to perform automatic assembly-sequence generation and complexity analysis, given only a geometric description of the proposed assembly. At present, STAAT is only able to handle certain types of assembly motions, namely, single-step translations; however, the underlying theoretical framework for STAAT extends to other classes of motions as well, including one-step translations with rotations, and n-step translations. Work is presently underway on applying this more advanced reasoning to a practical system (and, in fact, a new but related procedure to handle translations with rotations has just recently been developed (Guibas et al., 1995)). Nonetheless, within the context of assembly plans using one-step translations, STAAT represents a significant advance in speed and efficiency over previous work.

Our motivation for this system is threefold: to build a useful prototype for industry, to find new and useful algorithms in geometric reasoning for assembly, and to validate these and previous algorithms by implementing them in STAAT.

RELATED WORK

Many of the early assembly-reasoning systems were interactive ones, querying the user for geometric-reasoning information and generating assembly sequences from the answers (Bourjault, 1984; DeFazio and Whitney, 1987).

The problem of *automatically* generating assembly sequences is, in its full generality, an extraordinarily difficult one, recently shown to be NP-complete in both the two-dimensional (Kavraki and Kolountzakis, 1995) and three-dimensional (Kavraki et al., 1993) cases. As a result, much of the past and present work in this area focuses on restricted variants of the problem. One common thread that appears in most of this work is the strategy of "assembly by disassembly", in which an assembly sequence is generated by starting with the completed product and working backwards through disassembly steps.

Woo and Dutta (1991) created a system to remove one part at a time via single-step translations in $O(n \log n)$ average time, where n is the number of faces in contact. To do this they used the notion of a "disassembly tree", showing what parts in the assembly must be removed before what other parts. They also showed how their method could be used to find the optimal sequence to access and remove a given part for a restricted class of assemblies.

A number of systems exist to perform disassembly using translational motions only along the major axes (Miller and Hoffman, 1989; Hoffman, 1990; Lee and Shin, 1990; Subramani and Dewhurst, 1991). Hoffman (1990), in particular, created a system which can handle curved surfaces as well as flat ones, generates multi-step part trajectories, and also considers subassembly stability under gravity. Later, Hoffman (1991) extended his analysis to translational or rotational trajectories which are not necessarily along the major axes, by making educated guesses about possible local motions in a trial-and-error fashion.

Using computational-geometry approaches, Pollack et al. (1988) wrote an algorithm to separate two polygons by a sequence of translations; and Valade (1985) wrote one to separate two three-dimensional polyhedra by multiple translations. Not surprisingly, perhaps, the latter algorithm is quite slow (1 hour of CPU time for the example provided).

Wolter (1989), Homem de Mello and Sanderson (1989; and Sanderson, 1990), and Lin and Chang (1993) each used a basic approach similar to the one we shall describe here. Directions of possible part removal are suggested by contact information between the parts (or sometimes by the types of parts, *e.g.,* screws). The feasibility of the motions is then determined by "sweeping", or projecting, the parts in the proposed direction.

However, we believe that our work, by condensing much of the local analysis into a single compact data structure (the NDBG), provides a significantly faster means of performing these computations. This is evidenced by our system's relatively fast running times (reported below).

The work described in this paper is largely an extension of the work of Wilson (1992). Wilson introduced the NDBG data structure and described the means by which it could be used to reason about disassembly. In this paper, however, we present a significant improvement to this structure, and a system which can not only determine if assembly/disassembly is possible, but also seek to optimize particular figures of merit for the assembly process. Finally, because our system is implemented in C, whereas Wilson's is in Common Lisp, its computation times are reduced even further.

PRELIMINARIES

Before discussing STAAT in further detail, it is first necessary to define the following terms:

- A *disassembly graph* is an AND/OR graph representing all possible ways in which a product can be disassembled (or, conversely, assembled) (Homem de Mello and Sanderson, 1986). Each node in the graph represents a subassembly, and each AND'ed pair of children for that node represents two subassemblies into which the given subassembly can be decomposed. When there are multiple ways to break apart a subassembly, there are multiple pairs of children OR'd together.
- The *number of hands* required to perform a given step in an assembly sequence is defined as the number of subassemblies that are moving with respect to one another. For example, if a single subassembly is being removed from the rest of the assembly, then two hands are required: one for the moving subassembly, and one for the fixed subassembly. If the fixed subassembly is merely resting on a table, then, the table acts as a "hand" in this context.
- A *binary* assembly sequence is one which requires two hands, *i.e.,* no step requires three or more subassemblies to move in different directions simultaneously.
- A *monotone* assembly sequence is one in which each subassembly, once constructed, is final; that is, it is not modified by subsequent operations. For example, the latch assembly in Fig. 1 has no monotone binary assembly sequence. The only possible assembly sequence for this structure requires inserting P_2 all the way into P_1, inserting that subassembly into P_3, and then partially removing P_2 again. Because this last step changes the P_1-P_2 subassembly, the sequence is non-monotone.
- A *linear* assembly sequence is one in which each step involves the insertion of a single part into the rest of the assembly. The assembly sequence shown in Fig. 2, for example, is *not* linear because the step represented by the long arrow requires the insertion of a two-part subassembly into a three-part subassembly. An assembly for which a linear assembly sequence exists is called a *linear assembly*.
- A *stack* assembly sequence is one in which all the motions occur in a single direction, usually up-and-down. Circuit boards, for example, often have stack assembly

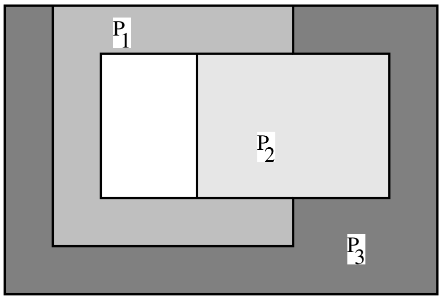


FIGURE 1: AN ASSEMBLY WITH NO MONOTONE BINARY ASSEMBLY SEQUENCE (WOLTER, 1988).

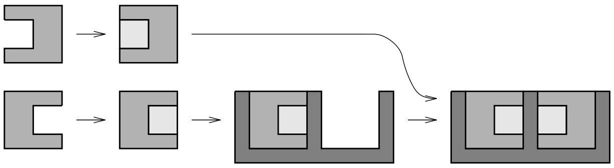


FIGURE 2: A NON-LINEAR ASSEMBLY SEQUENCE.

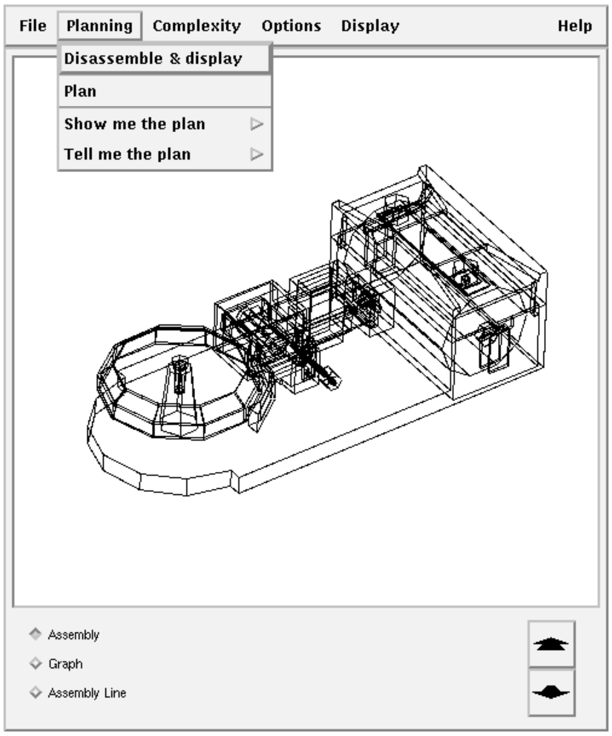


FIGURE 3: A TYPICAL DISPLAY SCREEN FOR STAAT.

sequences. An assembly for which a stack assembly sequence exists is called a *stack assembly*.

- *Local* motion is motion over an infinitesimal distance. A subassembly is said to be *locally free* if it can move an infinitesimal distance without colliding with other parts.
- *Global* or *extended* motion is motion over an infinite distance. A subassembly is said to be *globally free* if it can be removed to infinity.

For more details on the preceding terms, the reader is referred to Wilson (1992).

DESCRIPTION OF THE SYSTEM

Capabilities

STAAT, at present, is a stand-alone system whose input is a geometric description of the assembly. A typical display screen appears in Figure 3. When instructed to do so, STAAT computes a sequence of steps necessary to disassemble the given assembly; these steps can then be reversed to produce an assembly sequence. STAAT can either illustrate this sequence graphically (see Fig. 4), or do its analysis "quietly" and simply present its conclusions.

In addition to the assembly sequences themselves, STAAT can produce a number of complexity measures for the assembly process. The computation of the assembly sequences can be geared towards optimizing a particular complexity measure. Some examples of these measures are:

- **Linearity.** STAAT can restrict its analysis to linear assembly sequences, and thereby quickly determine if the assembly is a linear one. Linear assemblies have the advantage that they can be built using one long assembly line, in which each "station" on the assembly line adds a single part. This is helpful because multiple assembly lines may require a great deal of coordination to ensure that completed subassemblies arrive for final assembly at the same rate and at the same time.
- **Stack.** STAAT can also quickly determine if the product is a stack assembly requiring just up-and-down part insertions. If so, then it will not need to be rotated during its construction. This reduces both product cost and production time.
- **Ease of part removal.** STAAT can attempt to find the shortest partial disassembly sequence to extract a given part. This is an indication of how difficult it would be to service that part. For small assemblies, this computation can be done through an exhaustive search of the disassembly graph, producing the absolutely optimal removal sequence; for larger assemblies, more heuristic, non-guaranteed techniques are required.
- **Ease of fine-motion planning.** Ideally, when a part is inserted, there should be some physical features to align it, but not so many constraints as to make it difficult to insert (for example, in a 3x3 array of blocks, one would usually not insert the middle block last). Using a cost function to quantify these considerations, STAAT

can produce assembly sequences which attempt to make component insertion as easy as possible.

Other measures of complexity which we are presently investigating (but have not yet implemented in STAAT) are:

- What is the **minimum number of directions** of motion required for this assembly? This is a generalization of the stack-assembly question.
- Given two parts P_1 and P_2, does an assembly sequence exist that **inserts** P_1 **before** P_2? What if there are multiple "A-before-B" requirements—can a sequence be found to accommodate all of them?

Note that the relative importance of each of these complexity measures will depend on the particular application involved. For example, linearity is often useful for small assemblies, but for products with many parts, designers may instead want highly non-linear assembly sequences, to maximize parallelism in construction. Therefore, the authors do not presume to propose a single figure of merit for what constitutes a "good" assembly sequence. Rather, our purpose is to generate as many such measures as possible, and leave it to individual designers to choose among them.

STAAT can be instructed to compute either a single disassembly sequence or the entire disassembly graph. (See Figs. 5a and b.) Computing the full graph is useful for small assemblies because once it has been computed, STAAT can quickly find the absolutely optimal assembly sequence, according to the above or other criteria. With appropriate user-interface facilities, this analysis could even be made interactive, with the user fine-tuning his optimality criteria based on the results of the previous search.

Once an assembly plan has been computed, the system can also produce a schematic diagram of an assembly line one could use to manufacture the product. (See Fig. 6.)

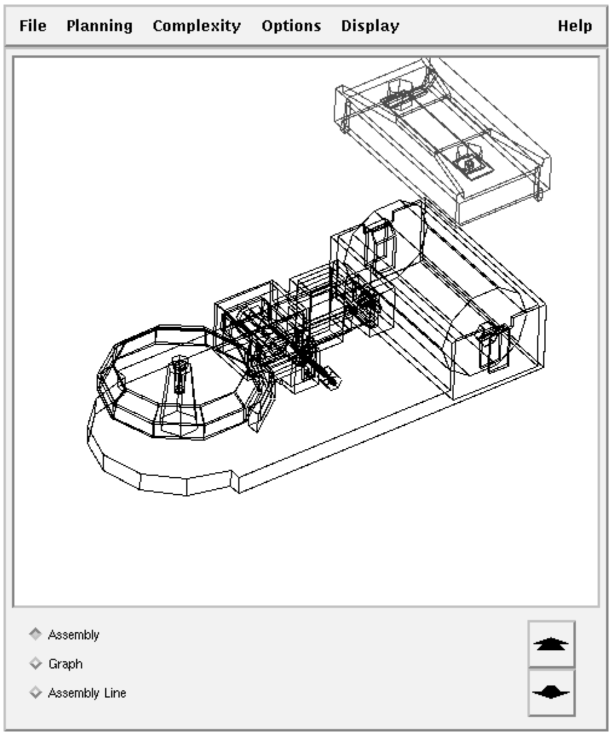


FIGURE 4: STAAT ILLUSTRATING A DISASSEMBLY SEQUENCE.

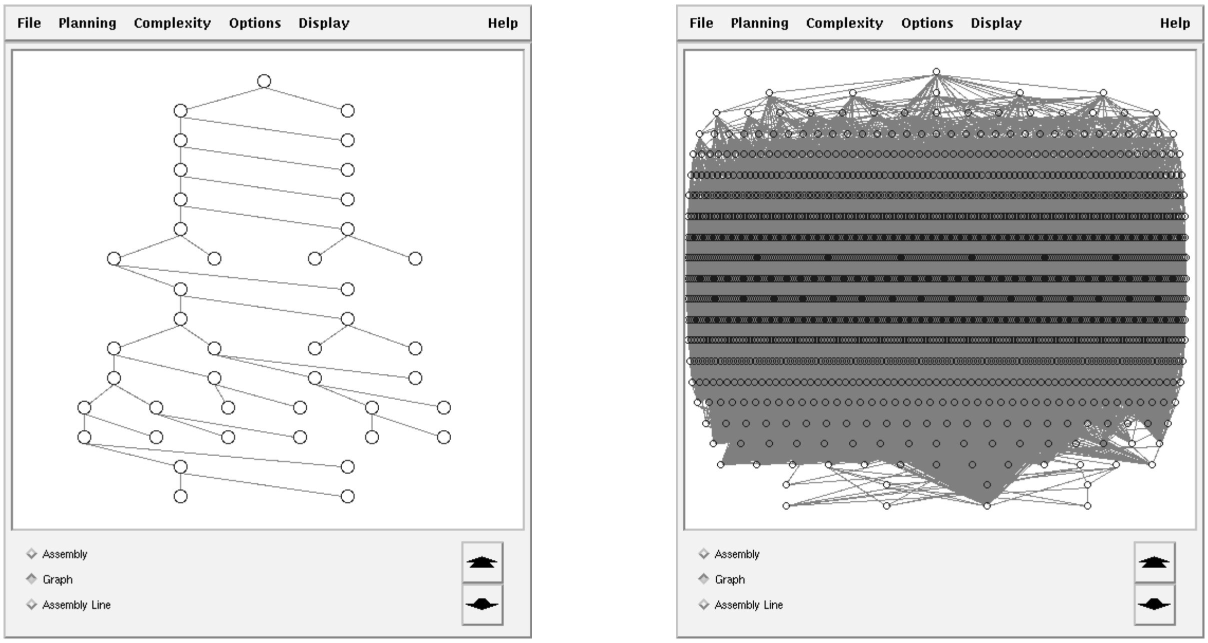
Restrictions

STAAT is presently limited to assembly sequences which are **binary** and **monotone**, and composed of **single-step translations**. The assemblies on which it operates are **polyhedral** and defined by **exact geometry**, *i.e.,* with no tolerances. (Recent results in assembly planning with tolerances are described by Latombe and Wilson (1994).) Additionally, the parts are considered **free-flying objects** in space; that is, no consideration is given to grasping concerns, the tools necessary to handle the parts, or physical stability. These are certainly important issues which would have to be addressed, for instance, in a robotic assembly system. However, they are beyond the scope of this particular project.

The only exception to the above is that if requested, STAAT is able to restrict its analysis to sequences with connected subassemblies; this is useful because connectedness is often a necessary (but not sufficient) condition for stability.

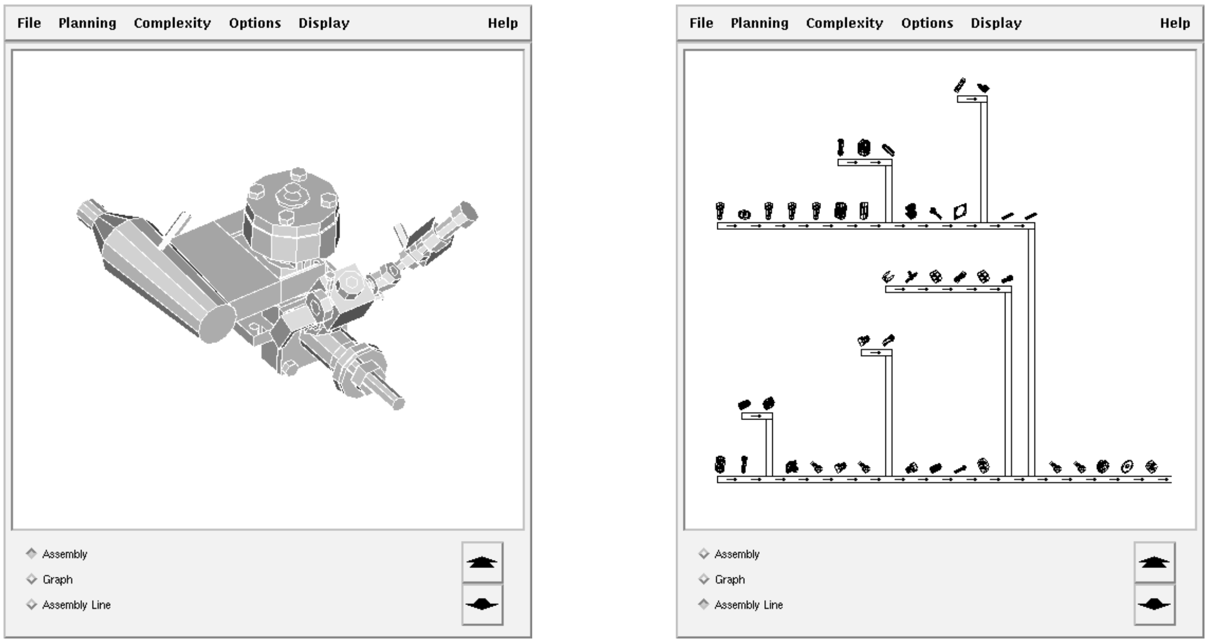
System Overview

STAAT's basic operating procedure is illustrated in Fig. 7. We shall shortly describe each of its main steps in more detail.



(a) **(b)**

FIGURE 5: AND/OR GRAPHS FOR THE BELL ASSEMBLY OF FIG. 3: (a) THE GRAPH GENERATED TO PRODUCE A SINGLE ASSEMBLY SEQUENCE; (b) THE ENTIRE AND/OR GRAPH.



(a) **(b)**

FIGURE 6: (a) A 42-PART MODEL AIRCRAFT ENGINE ASSEMBLY; (b) A (NON-OPTIMIZED) SET OF ASSEMBLY LINES THAT MIGHT BE USED TO MANUFACTURE IT.

However, very briefly, its behavior is as follows:

Find contacts. Take the CAD data for the parts and analyze their geometry to detect contacts. The algorithm used here is a fairly simple one, based on the surface representation of the parts. Admittedly, there may be other algorithms which are more efficient; however, contact detection is not the main focus of our efforts, and we shall not go into the details of our procedure here.

Find local translational freedom (LTF) cones. Combine the contact information into an LTF cone for each pair of parts, which gives the directions in which one part is locally free to move with respect to the other.

Build NDBG. Combine the LTF cones into the Non-Directional Blocking Graph, the central data structure describing all the blocking relationships in an assembly.

Find DBG. For a given direction of motion, compute the Directional Blocking Graph, a directed graph describing what parts are blocking what other parts, for local motions.

Find locally-free subassemblies. Given a direction of motion and a DBG, find a subassembly that is locally free in that direction.

Sweep. In order to examine *global* freedom, project all parts in the given direction. If the "shadow" of a moving part intersects the "shadow" of a stationary part, and the stationary part is in front of the moving part, then there will be a collision; try another motion.

Execute. Once a feasible action has been found, use this action to split the assembly into two subassemblies. Then recurse.

By performing the above steps under different constraints (*e.g.,* only remove one part at a time), we can compute the complexity measures described earlier.

The Non-Directional Blocking Graph is the data structure which is the key to all this analysis. Therefore, it is appropriate that we begin our more detailed discussion of STAAT with an explanation of this structure and how it works.

The Non-Directional Blocking Graph

For many classes of part motion (*e.g.,* infinitesimal translations, translations with rotations, single extended motions, n-step translations, etc.), there exists a *Non-Directional Blocking Graph*, or NDBG, which summarizes the blocking relationships within an assembly for that class of motion. So far in this project, only the NDBG for single-step translations has been implemented, but the theoretical foundation for this structure extends to other classes of motion, as well. As noted earlier, work is presently underway on finding useful algorithms to apply this theory in a practical system, with some promising early results (Guibas et al., 1995).

Even within the realm of NDBGs for single-step translations, there are two types of NDBGs available: one for infinitesimal translations and one for extended translations. In this paper we shall focus primarily on the infinitesimal version (complemented by sweeping operations for global analysis). However, we have also recently made progress on implementing extended NDBGs, as described below in a later section. Because extended NDBGs are guaranteed to yield disassembly solutions when they exist, we ultimately plan to use them as the primary

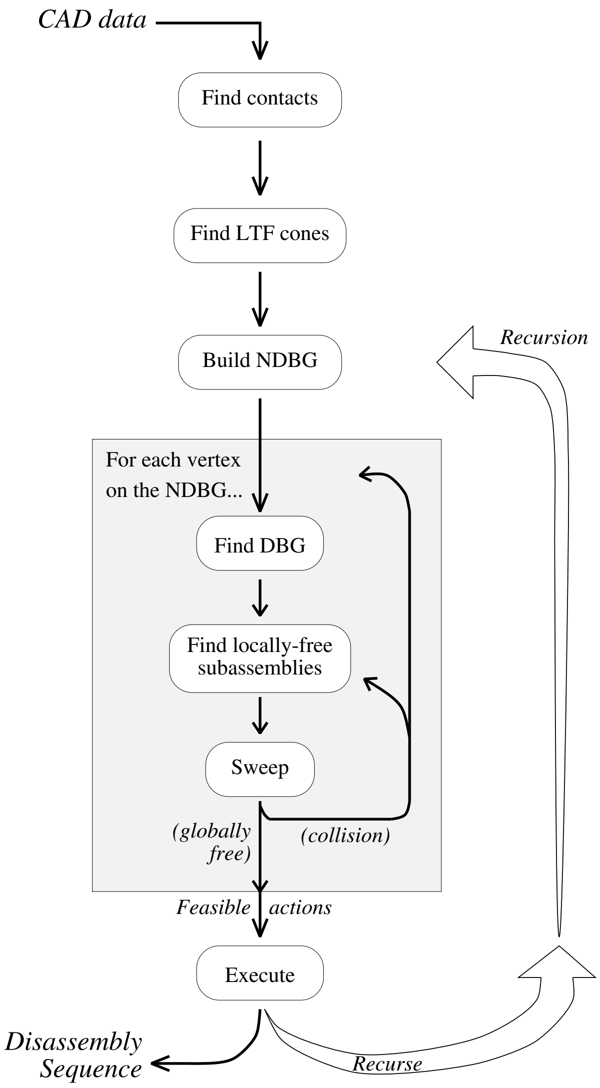


FIGURE 7: STAAT'S BASIC OPERATING PROCEDURE FOR GENERATING ASSEMBLY/DISASSEMBLY SEQUENCES.

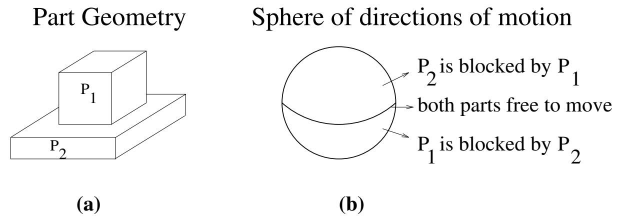


FIGURE 8: IN THIS TWO-PART ASSEMBLY, THE SPHERE OF POSSIBLE DIRECTIONS OF MOTION IS DIVIDED INTO THREE REGIONS.

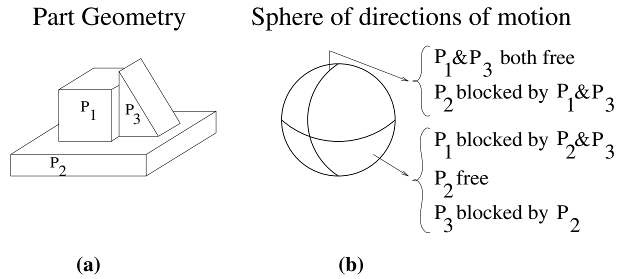


FIGURE 9: ADDING A THIRD PART SUBDIVIDES THE SPHERE EVEN FURTHER, ACCORDING TO THE BLOCKING RELATIONSHIPS AMONG THE PARTS.

means of performing disassembly.

We shall begin this discussion, however, by describing the NDBG for infinitesimal translations.

Given a part floating in three-dimensional space, the set of directions in which it can translate can be described by a sphere. Now suppose we have two parts, as shown in Fig. 8a. The sphere of directions of motion is now divided into three regions (Fig. 8b):

1. the top hemisphere, for directions in which P_1 is free to move, but P_2 is blocked by P_1;
2. the bottom hemisphere, for directions in which P_2 is free to move, but P_1 is blocked by P_2; and
3. the "equator" of the sphere, for directions in which both parts are free to move.

Continuing with this example, if we add a third part, as in Fig. 9a, the sphere of directions is divided even further, as shown in Fig. 9b. The new great circle on the sphere is caused by the new plane of contact between P_1 and P_3.

In each region of this sphere, there exists a different blocking relationship among the parts. However, if we "walk" around *within* a region, the blocking relationship does not change (for local motions). This is illustrated in Fig. 10. And this feature is the key to the usefulness of this arrangement: *By partitioning along the contact planes, we have discretized the continuous sphere of possible directions of motion into a finite number of regions, such that the local blocking relationship within each region is constant.* Furthermore, the number of these regions is polynomial in the number of planes of contact. This is a much more manageable discretization than the approach often seen in the literature, of generating all possible subassemblies (of which there are an exponential number) and testing them one by one.

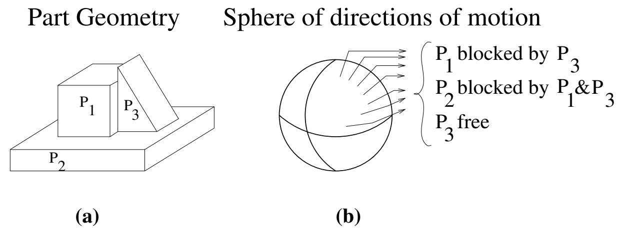


FIGURE 10: WITHIN EACH REGION OF THE SPHERE, THE BLOCKING RELATIONSHIP IS CONSTANT.

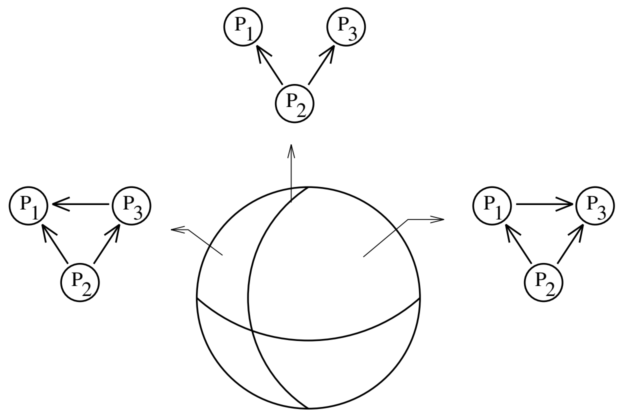


FIGURE 11: THE NON-DIRECTIONAL BLOCKING GRAPH FOR THE ASSEMBLY SHOWN IN FIG. 9a.

Typically, we represent the blocking relationships using a directed graph (known as a *Directional Blocking Graph*, or DBG), as shown in Fig. 11. The sphere and its associated DBGs are together known as the *Non-Directional Blocking Graph*.

Now, the preceding discussion was limited to infinitesimal motions; however, as noted above, it is also possible to create an NDBG for extended motions. In this NDBG, the cells on the sphere represent regions with a constant blocking relationship for *extended* motions. This type of NDBG is discussed further in a later section. It is also possible, theoretically at least, to generate an NDBG which encompasses rotational as well as translational motions. However, since these motions have six degrees of freedom, such an NDBG would be a five-dimensional sphere in six-dimensional space, and is beyond the scope of this paper.

Again, for more details on NDBGs and DBGs, the reader is referred to Wilson (1992).

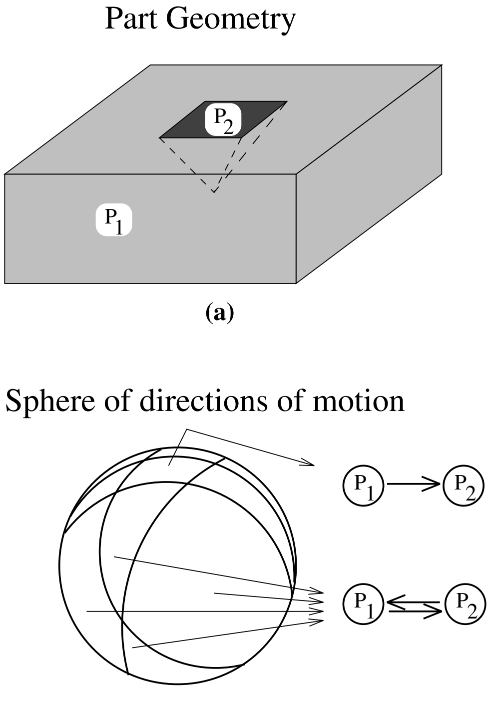


FIGURE 12: WHEN THERE CAN BE MULTIPLE CONTACTS BETWEEN PARTS, THE DBG'S MAY BE THE SAME ACROSS DIFFERENT REGIONS OF THE NDBG.

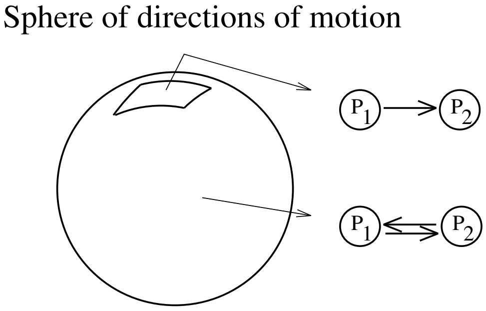


FIGURE 13: THESE ARE THE ONLY DIVISIONS THAT REALLY MATTER IN THE NDBG OF FIG. 12b.

A New, More Efficient NDBG

An improvement can be made to the NDBG structure just described. Consider the assembly of Fig. 12a, which, unlike the previous assemblies, contains multiple contacts between the same two parts. As can be seen in Fig. 12b, many regions on the resulting NDBG sphere all correspond to the same DBG. For purposes of separating these two parts, the NDBG could actually be simplified to something like Fig. 13. The reason why the NDBG of Fig. 12b is less efficient is that its boundaries delimit the regions where different contacts would be penetrated, whereas what we are interested in is where *any* contacts between these parts would be penetrated.

The "new" NDBG of Fig. 13 can be created if we combine all of a part's contacts into a *local translational freedom cone*, or LTF cone, before proceeding with the NDBG construction. An LTF cone is the subset of directions, from the entire sphere of directions, over which a given part can move locally without colliding with another given part. For example, in Fig. 14a, the four planes of contact between parts P_1 and P_2 reduce the possible motions of P_2 to just the cone of directions shown in Fig. 14b. This cone, then, is the LTF cone for P_2 for collisions with P_1.[1]

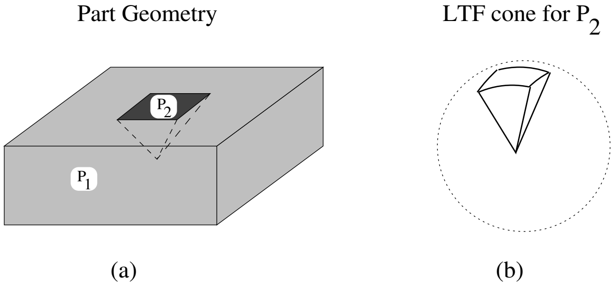


FIGURE 14: AN EXAMPLE OF A LOCAL TRANSLATIONAL FREEDOM (LTF) CONE.

[1] For a more rigorous definition of LTF cones, the reader is referred to Wilson (1992). Wilson, however, did not use LTF cones to simplify the NDBG, as we are doing here.

By merging all the LTF cones from all pairs of parts in contact, we obtain a much more efficient NDBG than if we use the contacts directly. Compare, for example, Figs. 15a and 15b, which show actual NDBGs for the assembly of Fig. 6a: one computed directly from the contacts, and one computed using LTF cones. As these figures illustrate, the elimination of the unneeded arcs leads to a tremendous reduction in NDBG complexity (and hence, a large improvement in computational efficiency). Moreover, since each arc in the new NDBG represents the boundary of an LTF cone, we know that the local blocking relationship *will* change as we go from one cell to the next. Thus, we have taken all the contact constraints from all the parts and distilled them into just those regions on the directional sphere which matter. This makes the approach presented here quite scalable to large assemblies.

Furthermore, the new NDBG has the advantage of being able to accommodate cylindrical surfaces as well as polyhedral ones. This is because a contact between cylindrical surfaces—for example, a peg in a hole—creates an isolated point (or perhaps two) on the NDBG sphere, in the direction(s) in which the peg can be moved; these points are allowable in the improved NDBG but not in the old one. It is our understanding that such cylindrical contacts represent a great majority of the curved-surface contacts in practical assemblies.

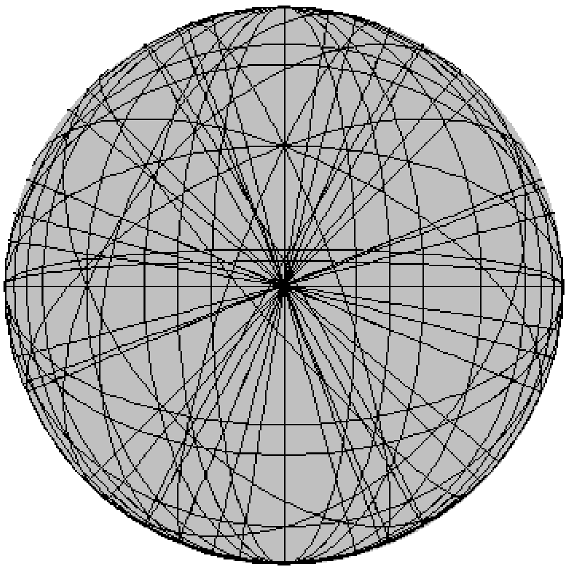
In a later section, we shall describe in more detail how the new type of NDBG is represented in memory.

Traversing the NDBG

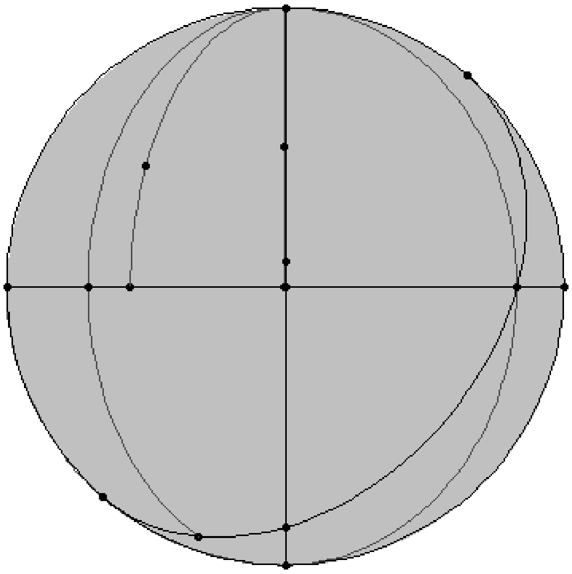
Once the NDBG has been computed, STAAT enters into a loop as it traverses the NDBG looking for directions in which to pull out subassemblies. The primary directions in which it looks are the so-called *NDBG vertices*, which are the points on the NDBG sphere where two or more arcs intersect. As Wilson (1992) showed, these directions are the most likely ones in which to find locally-removable subassemblies. This is because given two parts P_1 and P_2 in contact along some plane A (as in Fig. 8), any motion along A will allow P_1 and P_2 to slide past each other, whereas outside of A, one part will always be blocking the other. Therefore, local freedom is maximized along the contact planes, and by extension, at the NDBG vertices, where multiple contact planes intersect.

Now, it may well be that the directions of maximum *local* freedom are not those of of maximum *global* freedom. Consider, for example, the assembly in Fig. 16a. Locally, the best directions to move in are $\pm\hat{y}$, since both parts are free to move an infinitesimal distance in those directions. However, as can be seen, separating the parts to infinity in these directions would not be possible, because of protrusions in the larger part which we could not possibly have detected in our contact-based analysis. Hence, there may be cases where checking the NDBG vertices for removable subassemblies is not sufficient, and it is necessary also to examine directions in the middle of the *NDBG faces* (the regions on the sphere bounded by the arcs).

Note that even if we do so, we are still not absolutely guaranteed to find a solution, because for every specific direction we happen to check, there could be some small distant obstacle which blocks motion in that particular direction. The solution to



(a)



(b)

FIGURE 15: THE NDBG FOR THE ENGINE ASSEMBLY OF FIG. 6a, (a) COMPUTED DIRECTLY FROM THE CONTACTS, AND (b) COMPUTED FROM THE LTF CONES.

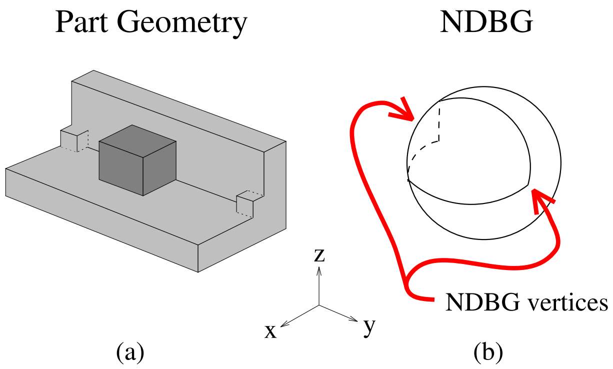


FIGURE 16: ALTHOUGH THE NDBG VERTICES GIVE THE DIRECTIONS OF MAXIMUM LOCAL FREEDOM, THERE MAY BE GLOBAL CONSTRAINTS WHICH PREVENT EXTENDED MOTION IN THOSE DIRECTIONS.

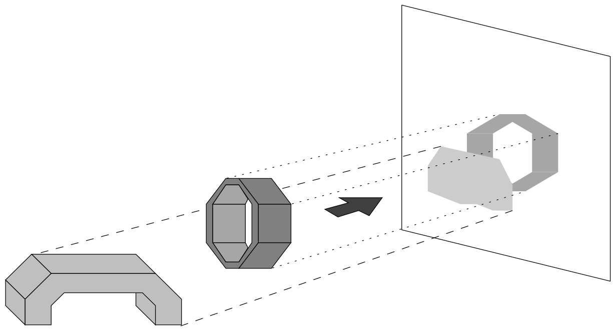


FIGURE 17: BY PROJECTING PARTS IN A PROPOSED DIRECTION OF MOTION, WE CAN DETERMINE IF THERE WILL BE A COLLISION BETWEEN THEM. THIS "GLOBAL" ANALYSIS COMPLEMENTS OUR EARLIER LOCAL ANALYSIS, WHICH GAVE US THE PROPOSED DIRECTION IN THE FIRST PLACE.

this problem is to use the extended NDBG. However, in practice, using the local NDBG works quite well, and in the remainder of this section we shall continue to discuss this approach.

Building and Using the DBG

The first task, once a candidate direction has been identified, is to generate the *Directional Blocking Graph* (DBG) for that direction (see Fig. 11). The DBG is a directed graph which explicitly represents what parts are blocking what other parts for local motion in the given direction. The first DBG constructed is obtained by comparing the proposed direction to the various LTF cones. Thereafter, as we traverse the NDBG, we update the DBG incrementally to reflect new blocking relationships (a much less expensive process than building the DBG from scratch each time). For example, every time we enter an LTF cone, some "A is blocking B" constraint is relaxed; every time we exit one, such a constraint is re-imposed.

Once we have the DBG, we use it to identify locally-removable subassemblies. These correspond to sets of nodes from which there are no outgoing arcs leading to other nodes. A graph-searching algorithm for finding such subassemblies is described by Wilson (1992); it, in turn, relies on a strong-connectedness algorithm given by Aho et al. (1985).

Every locally-free subassembly found in this way is then passed to a *sweeping* module, which determines if the subassembly can be removed to infinity. This is done by "sweeping", or projecting, the subassembly in the given direction (see Fig. 17). If the projection of any moving part overlaps the projection of any stationary part, and the stationary part is in front of the moving part, then we know that there will be a collision. In this case, we go back and try again with either a new subassembly or an entirely new candidate direction and DBG.

Execution and Recursion

In this manner, we obtain a list of one or more actions with which to partition the assembly. In the event that we have more than one feasible action, our choice among them will depend on our overall objective. If we simply wish to determine, yes or no, whether disassembly is possible, then the choice is arbitrary. However, if, say, we are trying to extract a given part quickly, then we might (heuristically) choose the action which removes that part with a minimum of other parts next to it.

Once an action is selected, we use it to break up the assembly into two subassemblies, and recursively disassemble those as indicated in Fig. 7. As STAAT proceeds with the disassembly, it builds up the AND/OR disassembly graph explicitly in its memory. If, at some point, a subassembly is found which cannot be disassembled, STAAT may be instructed to "back up" the AND/OR graph and try to find another sequence of actions to bypass that particular subassembly. (Normally, this would be ineffective since adding parts back onto an inseparable subassembly only yields another inseparable subassembly. However, if the user has specified that all subassemblies be connected, then it may well be that the additional parts could render the assembly separable. In any event, this "try-try-again" method is merely a user option.)

Upon completion, the sequence of actions can be reversed to produce an *assembly* plan. This plan can be outputted with a graphical demonstration, and/or with assembly instructions in plain English.

A CLOSER LOOK AT THE SYSTEM

Representation of the NDBG

A geometry software package was developed in connection with STAAT, to construct and maintain the NDBG as an arrangement of polygons on a spherical surface. The input for this module is a set of LTF cones (described earlier). Each LTF cone produces a polygon on the unit sphere (see fig. 18). Possible degenerate polygons include full spheres, full hemispheres, single arcs, and isolated vertices.

The collection of polygons partition the sphere into a set of

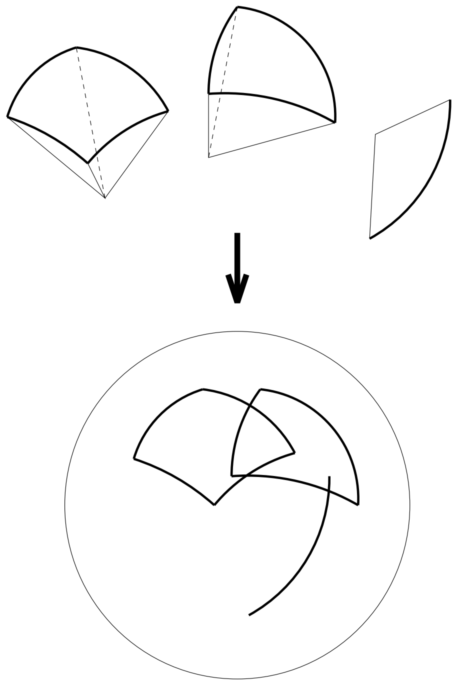


FIGURE 18: LTF CONES ARE MERGED TOGETHER TO FORM THE NDBG. HOWEVER, THIS ARRANGEMENT OF POLYGONS ON THE SPHERE IS SOMEWHAT DIFFICULT TO REPRESENT AND MANIPULATE AS-IS.

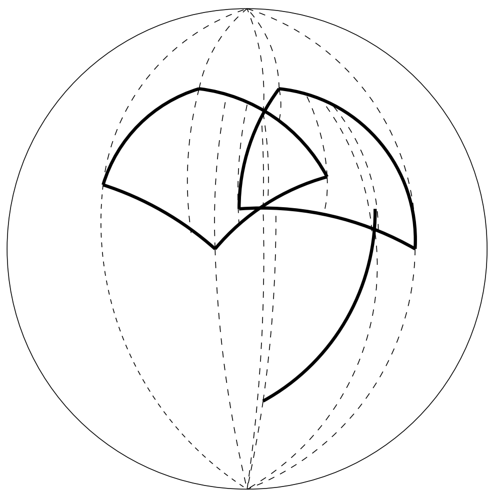


FIGURE 19: BY EXTENDING VERTICAL "THREADS" FROM EACH VERTEX, WE DECOMPOSE THE ARRANGEMENT INTO TRAPEZOIDS AND TRIANGLES, WHICH ARE EASIER TO REPRESENT.

faces; however the boundaries of these faces may involve arbitrarily many edges and arbitrarily many connected components. To simplify the required data structures, we use a popular technique, introduced by Chazelle and Incerpi (1984), to vertically decompose the faces of the arrangement into trapezoids.

First, however, we artificially break the sphere into two hemispheres, and choose an arbitrary pair of antipodal points on the boundary of the hemispheres to be the "poles". Although we maintain the arrangement directly on the hemispheres, each hemisphere can be viewed as a standard plane via a stereographic projection, and so we are able to apply techniques for maintaining arrangements in the plane.

Now we are able to decompose the arrangement into trapezoids. From every vertex, we extend a vertical "thread" upwards and downwards (towards the poles) until an edge of the arrangement is reached. (See fig. 19.) At this point, the original faces have been broken up into pieces with the guarantee that each remaining face is bounded by exactly two edges, and at most two vertical threads. For this reason, these faces are called *trapezoids*. In addition, because the number of threads introduced is bounded by twice the number of vertices, the asymptotic complexity of the new arrangement has remained the same. A more detailed explanation of trapezoidal decomposition is given by Chazelle and Incerpi (1984).

The algorithm used for construction is a basic randomized incremental approach, similar to those of Mulmuley (1993) or Seidel (1991). For complete details of this section of the software, the reader is referred to Goldwasser (1994).

Extended NDBGs and Minkowski sums

In this section, we define "extended" NDBGs and describe how they are constructed using Minkowski sum computations. We also provide a few details about the algorithm that is used.

Recall that until now we have been dealing with NDBGs for infinitesimal translations. As noted earlier, though, the extended NDBG is defined exactly the same way, but substituting extended motions instead of infinitesimal motions. That is, the extended NDBG partitions the sphere of directions into regions with constant blocking relationships for *extended* motions.

One advantage of using extended NDBGs to determine a feasible motion is that the extended blocking relationships are immediately known and stored explicitly. Thus, we do not need to sweep each subassembly to check for collisions. This can be a significant savings because when we use infinitesimal NDBGs, a large number of sweeps (up to an exponential number, in the worst case) may have to be tried before we successfully separate the subassembly. Profiling of the STAAT executable confirms that a significant amount of time is indeed spent doing sweeping checks.

Another important advantage of using extended NDBGs is that because *all* the extended blocking relationships are known and stored, we are guaranteed to find a valid disassembly motion when it exists, simply by searching the extended-NDBG sphere.

The extended NDBG can be easily constructed from a classical geometric object called the *Minkowski sum* (related closely to the *Convolution*) of two polyhedra representing the parts. More precisely, we use the *Minkowski difference* of the two polyhedra:

Definition. The *Minkowski difference* R of two polyhedra $P, Q \subset R^3$ is the set of vectors $\vec{r}$ such that Q, translated in $\vec{r}$, intersects with P.

The Minkowski difference R is known to be a multiply-connected generalized polyhedron with holes.

The extended NDBG can be constructed from R using the following method. An extended motion by part P_i in a particular direction α is feasible if and only if an infinite ray starting from the origin in the direction α does not contain any vector in the Minkowski difference of P_i with any other part P_j. In other words, motion along α is feasible if and only if the projection of R onto the unit sphere of directions does not cover the point on the sphere in direction α. Therefore, the projection of the Minkowski difference on the unit sphere gives us the generalized extended-NDBG polygon on the unit sphere.

We compute the Minkowski difference using a simple technique of convolving pairs of faces and vertices. Furthermore, for efficiency our algorithm makes special use of the fact that only the projection of the Minkowski difference is needed, and not the entire structure. Given two polyhedra P and Q, we take each vertex of P and convolve it with a face of Q, and vice-versa. (The Minkowski difference R also contains faces that are convolutions between two edges, but we do not need to compute these because they do not add any extra area to the projection of R on the unit sphere.) We use some heuristics here to cut down on the number of faces to deal with. As the convolution faces are computed, they are projected straight-away onto the unit sphere. We then compute the union of all of the faces using a 2-D *Bentley-Ottman* style line sweep technique (Preparata and Shamos, 1985). We save on efficiency by doing a 2-D union computation to find the projection directly rather than finding the 3-D Minkowski difference and projecting it explicitly. For more details on some of the geometric techniques used in the algorithm, the reader is referred to Ramkumar (1994).

We compute the Minkowski difference of each pair of parts in this manner and store the resulting extended-NDBG polygons for later use in the disassembly process. At present, our implementation of this technique is relatively new, but after some further testing we expect it to replace the sweeping module described earlier.

IMPLEMENTATION

STAAT is implemented in C and uses the Motif toolkit in its user interface. It runs on either a DEC alpha or a DEC 5000 workstation. We are also presently adapting it for use on a Sun SPARCstation 10.

The assembly data format, which was developed in-house, is a straightforward boundary representation of a polyhedral assembly. Faces may be non-convex and may contain holes. The data format also supports cylindrical and circular features, for future expansion of STAAT.

A separate stand-alone program can convert IGES 5.1 data files into STAAT's format; this process transforms any curved surfaces into a triangular mesh.

EXPERIMENTAL RESULTS

Experiments were conducted to measure STAAT's running times using infinitesimal NDBGs plus sweeping. The results are shown in Table 1. Three computation times were measured:

- The time to partition the root node, *i.e.,* break the top-level assembly into two subassemblies;
- The time to find a complete disassembly sequence for the assembly; and
- The time to compute the entire AND/OR graph for the assembly. For the larger assemblies, this is not feasible in practice because of the large size of the graph.

As can be seen, the running times are significantly better than those reported by Wilson (1992), often by a full order of magnitude. Much of this is probably due to the fact that STAAT is implemented in C, and Wilson's GRASP is in Common Lisp; however, we also believe that the more efficient NDBG structure contributes significantly to the speedup. The running times shown also compare quite favorably to those reported by Lin and Chang (1993) ($\sim$ 2 minutes for a 14-part assembly). In the near future, we hope to obtain CAD data files from industry, with which to conduct further tests of STAAT.

At the time of this paper's submission, the extended NDBG module has not been fully incorporated into the main system, so no comparable running times are presently available using that method.

CONCLUSION

We have presented a system, STAAT, to automatically determine how to assemble or disassemble a product using single-step translations, given only a description of the parts' geometry. This is accomplished through the use of the non-directional blocking graph, a compact data structure summarizing all the blocking relationships in an assembly for a given class of motions (in this case, single-step translations). Two types of NDBGs were presented: one for infinitesimal motions, which is the primary version used by our system today, and one for extended motions, which when fully incorporated will be more complete and give STAAT the guarantee of finding disassembly solutions when they exist.

In addition to finding the assembly sequences themselves, STAAT can also be used to compute various complexity measures for the assembly process, and to find assembly sequences which attempt to optimize these measures.

The results presented here represent a significant improvement over previous work along these lines. Recent related work by other researchers in our laboratory (Guibas et al., 1995; Latombe and Wilson, 1994; Wilson et al., 1995) extends the ideas described above to include rotation and toleranced assemblies. Other possible directions of future work include multi-step translations, the handling of simple curved surfaces (*e.g.,* cylinders and spheres) as well as flat ones, and the discovery and computation of new measures of assembly complexity.

TABLE 1: RUNNING TIMES FOR THE VARIOUS ASSEMBLIES, IN SECONDS, EXCLUDING TIME TO FIND CONTACTS. (GRASP FIGURES ARE FROM WILSON (1992).)

Assembly	Number of Parts	STAAT (DEC alpha)			STAAT (DEC 5000)			GRASP (DEC5000)		
		Node	Sequence	Graph	Node	Sequence	Graph	Node	Sequence	Graph
Bell	17	0.03	0.47	8.4	0.12	1.6	25	0.9	16	522
Bell	22	0.17	0.6	53	0.6	2.3	157	1.4	21	2186
Engine	12	0.15	1.0	6.7[2]	0.5	4.0	24[2]	0.5	30	42
Engine	30	0.20	3.0	—	0.7	11	—	3.6	112	—
Engine	42	0.13	3.8	—	0.5	14	—	8.5	189	—

ACKNOWLEDGEMENTS

Work on this paper has been supported by a grant from the Stanford Integrated Manufacturing Association (SIMA), and by NSF/ARPA Grant IRI-9306544. Additionally, work by the first author has been supported by an NSF graduate research fellowship; work by the second author has been supported by Matra Datavision, Les Ulis, France; and work by the third and fourth authors has been supported by NSF Grant CCR-9215219.

REFERENCES

Aho, A. V., Hopcroft, J. E., and Ullman, J. D., 1985, *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, pp. 222-226.

Bourjault, A., 1984, "Contribution à une approche méthodologique de l'assemblage automatisé: élaboration automatique des séquences opératoires", Ph.D. thesis, Faculté des Sciences et des Techniques de l'Université de Franche-Comté.

Chazelle, B., and Incerpi, J., 1984, "Triangulation and shape-complexity", *ACM Transactions on Graphics*, vol. 3, no. 2, pp. 135-152.

DeFazio, T. L., and Whitney, D. E., Dec. 1987, "Simplified generation of all mechanical assembly sequences", *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 6, pp. 640-658.

Goldwasser, M., 1995, "An implementation for maintaining arrangements of polygons", *Communication* in *Proceedings, 11th Annual ACM Symposium on Computational Geometry*, (to appear).

Guibas L. J., Halperin, D., Hirukawa, H., Latombe, J.-C., and Wilson, R. H., 1995, "A simple and efficient procedure for polyhedral assembly partitioning under infinitesimal motions", *Proceedings, IEEE International Conference on Robotics and Automation*, (Nagoya, Japan), (to appear).

Hoffman, R., 1990, "Automated assembly planning for B-rep products", *Proceedings, 1990 IEEE International Conference on Systems Engineering*, (Pittsburgh, PA), pp. 391-394.

Hoffman, R., 1991, "A common sense approach to assembly sequence planning", *Computer-Aided Mechanical Assembly Planning*, Kluwer Academic Publishers, Boston, pp. 289-313.

Homem de Mello, L. S., and Sanderson, A. C., 1986, "AND/OR graph representation of assembly plans", *IEEE Transactions on Robotics and Automation*, vol. 6, no. 2, pp. 188-199.

Homem de Mello, L. S., and Sanderson, A. C., 1989, "A correct and complete algorithm for the generation of mechanical assembly sequences", *Proceedings, 1989 IEEE International Conference on Robotics and Automation*, (Scottsdale, AZ), vol. 1, pp. 56-61.

Kavraki, L., Latombe, J.-C., and Wilson, R. H., 1993, "On the complexity of assembly partitioning", *Information Processing Letters*, vol. 48, pp. 229-235.

Kavraki, L., and Kolountzakis, M., 1995, "Partitioning a planar assembly into two connected parts is NP-complete", *Information Processing Letters*, (to appear).

Latombe, J.-C., and Wilson, R. H., 1995, "Assembly sequencing with toleranced parts", *Proceedings, Third ACM/IEEE Symposium on Solid Modelling and Applications*, (Salt Lake City, UT), (to appear).

Lee, S., and Shin, Y. G., 1990, "Assembly planning based on geometric reasoning", *Computers and Graphics*, vol. 14, no. 2, pp. 237-250.

Lin, A. C., and Chang, T.-C., 1993, "3D MAPS: Three-Dimensional Mechanical Assembly Planning System", *Journal of Manufacturing Systems*, vol. 12, no. 6, pp. 437-456.

Miller, J. M., and Hoffman, R. L., 1989, "Automatic assembly planning with fasteners", *Proceedings, 1989 IEEE International Conference on Robotics and Automation*, (Scottsdale, AZ), vol. 1, pp. 69-74.

Mulmuley, K., 1993, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice Hall, New York.

Pollack, R., Sharir, M., and Sifrony, S., 1988, "Separating two simple polygons by a sequence of translations", *Discrete and Computational Geometry*, vol. 3, no. 2, pp. 123-136.

Preparata, F. P., and Shamos, M. I., 1985, *Computational Geometry, An Introduction*, Springer-Verlag, New York, pp. 278-290.

Ramkumar, G. D., 1994, "Efficient algorithms to compute convolutions of non-convex polyhedra", Manuscript, Department of Computer Science, Stanford University, Stanford, CA.

Sanderson, A. C., 1990, "Automatic generation of mechanical assembly sequences", *Geometric Modeling for Product Engineering*, North-Holland, Amsterdam, pp. 461-482.

Seidel, R., July 1991, "A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons", *Computational Geometry The-*

[2] It should be noted that for the bell assemblies, only connected subassemblies were generated, while for the engine assemblies, STAAT generated *all* possible subassemblies, because the engines have no disassembly sequence using only connected subassemblies.

ory and Application, vol. 1, no. 1, pp. 51-64.

Subramani, A. K., and Dewhurst, P., 1991, “Automatic generation of product disassembly sequences”, *Manufacturing Technology CIRP Annals*, vol. 40, no. 1, pp. 115-118.

Valade, J.-M., 1985, “Geometric reasoning and automatic synthesis of assembly trajectory”, *Proceedings, 1985 International Conference on Advanced Robotics*, (Tokyo, Japan), pp. 43-50.

Wilson, R., 1992, “On Geometric Assembly Planning”, Ph.D. thesis, Stanford University, Stanford, CA.

Wilson, R. H., Kavraki, L., Lozano-Pérez, T., and Latombe, J.-C., 1995, “Two-Handed Assembly Sequencing”, *The International Journal of Robotics Research*, MIT Press, (to appear). Also available as Technical Report STAN-CS-93-1478, Stanford University, Stanford, CA.

Wolter, J. D., 1988, “On the Automatic Generation of Plans for Mechanical Assembly”, Ph.D. thesis, The University of Michigan.

Wolter, J. D., 1989, “On the automatic generation of assembly plans”, *Proceedings, 1989 IEEE International Conference on Robotics and Automation*, (Scottsdale, AZ), vol. 1, pp. 62-68.

Woo, T. C., and Dutta, D., May 1991, “Automatic disassembly and total ordering in three dimensions”, *Transactions of the ASME*, vol. 113, no. 2, pp. 207-213.