Stanford Artificial Intelligence Laboratory
Memo AIM-280

May 1976

Computer Science Department
Report No. STAN-CS-76-555

MONTE CARLO SIMULATION OF TOLERANCING IN DISCRETE PARTS MANUFACTURING AND ASSEMBLY

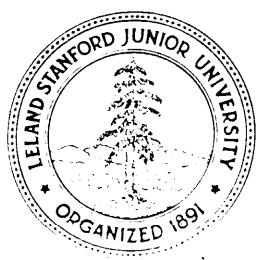
by

David D. Grossman

Research sponsored by

National Science Foundation

COMPUTER SCIENCE DEPARTMENT
Stanford University



Stanford Artificial Intelligence Laboratory
Memo AIM-280

May 1976

Computer Science Department
Report No. STAN-CS-76-555

MONTE CARLO SIMULATION OF TOLERANCING IN DISCRETE PARTS MANUFACTURING AND ASSEMBLY

by

David D. Grossman

ABSTRACT

The assembly of discrete parts is strongly affected by imprecise components, imperfect fixtures and tools, and inexact measuremets. It is often necessary to design higher precision into the manufacturing and assembly process than is functionally needed in the final product. Production engineers must trade off between alternative ways of selecting individual tolerances in order to achieve minimum cost, while preserving product integrity. This paper describes a comprehensive Monte Carlo method for systematically analysing the stochastic implications of tolerancing and related forms of imprecision. The method is illustrated by four examples, one of which is chosen from the field of assembly by computer controlled manipulators.

Permanent address: Computer Sciences Department, IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.

This research was supported by the National Science Foundation under Contract NSF APR74-01390-A02. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, NSF, or the U. S. Government.

INTRODUCTION

The assembly of discrete parts is a major fraction of industrial production. The role of computers in this field has been limited primarily to production and inventory control, computer aided design, and programming numerically controlled machine tools. Very little progress has been made in applying computers "to the problem of simulating assembly processes, in spite of the fact that such simulation offers the possibility of considerable savings over the alternative cost of building pilot production lines.

When one examines other large industrial fields one finds that computer simulation is a much more widely used tool. There are basically two reasons, however, why this tool has not been extensively applied in discrete parts assembly. First, because assembly is not a scientific discipline, experience is formulated as a set of ad hoc principles rather than as a mathematical theory. Although such principles may be set forth in textbooks,[1] it is difficult to embody them in computer simulations. This situation is in sharp contrast, for example, to the way differential equations can be used to model complex chemical processes. The second reason is that assembly. environments contain an immense variety of dissimilar objects. This aspect of assembly is in sharp contrast, for instance, to nuclear physics simulations where all neutrons behave in the same way.

The only obvious unifying principle in discrete parts assembly is that in S-dimensional space no two objects may intersect. This fact suggests a formulation of the simulation problem in terms of set theory, an approach which is being taken in research on parts description at the University of Rochester.[2,3,4] Set theoretic representations are good for determining if a given point is inside a particular set, but performance difficulties arise on problems involving pairs of 'sets. For example, the question of whether or not a piston intersects a motor block is difficult to answer because it is likely to cause a lengthy search for a point contained in both sets. Compounding this difficulty is the fact that assembly involves continuous motion of the discrete parts, so that it is desirable to be able to solve set intersection problems at every instant of time. The computational algorithms would not be hard to formulate, but the execution times would be extremely long, even on the fastest computers in the world. For this reason, simulation of the full assembly process is intractable, although simulation of special classes of assembly problems is still a practical and achievable goal.

From among the many aspects of assembly which could conceivably be modeled, this paper is concerned with the implications of tolerancing and imperfection. In the literature on this subject, dimensional tolerancing has come to mean specifying the tolerances of parts in mechanical drawings. A national standard has been established which defines the meanings of tolerancing symbols in drawings[5] and textbooks have been written to explain the use of these symbols.[6] The emphasis on drawings, however, tends to obscure the underlying reasons for being concerned with tolerances. The issue is not so much what 3.000±.005 cm means but rather *why* the designer chose to specify this tolerance in the first place.

There are three factors which enter into specifying tolerances in drawings. First, the discrete part which is described must ultimately be assembled into a product which is expected to have some function, and the tolerance may be needed to provide this function. For instance, it is highly desirable that each chamber of a Colt revolver align accurately with the barrel. Secondly, the part may be required to have certain tolerances in order that the assembly process itself be feasible. For example, in order to-assemble an automobile engine, the holes in the gasket must align with those in the block. Also, it is often necessary to have very accurate parts to avoid jamming vibratory feeders. In fact, it is often necessary to design higher precision into the assembly process than is functionally needed in the final product. Finally, tolerances may be assigned to correspond to the capabilities of the manufacturing method chosen., Tolerances achievable by sheet metal stamping would not be the same as those achievable on a numerically controlled machine tool, and it would be foolish to assign tolerances in a drawing which would give unreasonably small yields.

The product designer uses his expertise in product design, assembly, and manufacturing to specify tolerances in the drawing which are both adequate and achievable. An excellent textbook has been published which describes the considerations 'involved in this process.[7] , The process is complicated because the design criteria depend on the combined tolerances, rather than on the tolerances individually. Typically, the designer must trade off between alternative ways of selecting individual tolerances in order to achieve some resultant tolerance with minimum cost. Unfortunately, the 3-dimensional relationships involved are usually too tedious to allow a'rigorous mathematical treatment in all but the simplest cases. The designer therefore uses a great deal of intuition in reaching a decision. Finally he writes down a number like 3.000±.005 cm and throws away all the information which went into this decision.

A recent paper from General Motors describes a system which enables product designers to specify a set of individual parts tolerances and simulate the stochastic properties of interesting resultant tolerances.[8] The system is based on the Monte Carlo method, a simulation technique which is well known and has' been widely used in many other applications.[9] The existence of the GM paper shows that a need exists for simulation tools in the field of parts tolerancing. The problem of tolerancing is sufficiently hard, and the stakes are sufficiently high, that intuition is no longer a satisfactory method for specifying parts tolerances.

The approach taken in the GM work is to provide an interactive system in which the user can obtain high statistics very quickly. In order to achieve execution speed, the user must explicitly provide all the equations which tell how the resultant tolerances depend on the individual tolerances. The system models just the positions and orientations of a few features of the part, rather than the entire part shape. This system is apparently proving quite useful to GM designers.

Aside from the GM work, the only other published papers relating to modeling parts tolerances are those from the university of Rochester, where a language called PADL for

representing a class of discrete parts is being developed.[2,3,4] The hope is that PADL descriptions can someday be used to generate programs for numerically controlled machine tools which can make the parts.

The topic of parts representation without regard to tolerancing has been studied by Binford, Agin, and Nevatia,[10,11,12] Braid,[13,14] Baumgart,[15,16] Grossman,[17] and Lieberman and Lavin.[18,19] Although none of these parts modeling schemes was designed with tolerancing in mind, both the Baumgart and Grossman approaches offer a natural way of adding Monte Carlo procedures to simulate tolerances. As the author of one of these papers, my choice of which of the two systems to use for the current work was highly biased. I chose to use my own system solely because I am much more familiar with it.

Although the balance of this paper describes a specific implementation of Monte Carlo tolerancing within a parts representation system, many of the issues discussed are implementation independent. The point of this paper is not simply to give a blueprint for a specific way of simulating tolerances but rather to show that such a system is possible, to expose some of the design issues, and to give examples of ways in which the system might be used.

The simulation method described in this paper most closely resembles that of the GM paper, but there are several mapr differences. Whereas the CM system computes the resultant tolerances of indivrdual parts from tolerances specified in mechanical drawings, the current work is much more comprehensive. It allows one to simulate the propagation of tolerances all the way from the manufacturing process right through the assembly process. Also, while the GM work requires that the user explicitly supply formulas for the resultant tolerances as functions of the individual tolerances, the current work provides system routines which automatically perform these sorts of operations numerically. This provision is particularly useful because in many situations the relevant formulas can not be derived in closed form. On the other hand the GM system is Interactive, runs at high speed, and yields high statistics answers, while the current system runs in batch mode, executes much more slowly, and therefore yields much poorer statistics.

The next section of this paper reviews the main features of my earlier publication on representing parts by PL/I procedures and explains how this system can easily be applied to the Monte Carlo simulation of parts tolerances. This method is then illustrated by four specific examples, one of which is chosen from the field of assembly by computer controlled manipulators. The reason for chosing this example is that this research was carried out as part of continuing manipulator projects at the IBM T. J. Watson Research Center and the Stanford University Artificial Intelligence Laboratory. However, it is important to stress that the simulation techniques described here are applicable not only in the domain of computer controlled assembly, but also in the much wider domain of manufacturing and assembly as they exist in industry today, using conventional equipment and procedures. The paper closes wrth a discussion of research areas appropriate for extension of the Monte Carlo tolerancing method.

MONTE CARLO METHOD

Distributions

The basic idea of any Monte Carlo calculation is to generate an ensemble of models which simulates an ensemble of real entities.[9] The statistical properties of the real entities may then be simulated by studying the corresponding properties of the models. Such simulation is useful when purely analytical methods cannot be found.

For the case of' discrete parts manufacturing and assembly, the real entities consist of three-dimensiona! objects at a workstation. These objects include component parts and their features, tools and fixtures, measuring instruments, and automation equipment up to the level of complexity of transfer lines and computer controlled manipulators. For all of these objects, the primary attributes to be modeled are shape, position, and orientation.

In simulating statistical distributions of shape, position, and orientation attributes, it is necessary to define the meaning of expressions of the form 3.000±.005 cm. One possible definition would be a normal distribution with a mean of 3.000 cm and a standard deviation of which ,005 cm is some small integral multiple. This choice would allow dimensions to fall outside the specified range, albeit infrequently. Another possibility would be to have a distribution which goes rigorously to zero outside the specified range. Inside the range, the distribution could be uniform, or peaked at 3.000 cm, or bimodally peaked at 2.995 cm and 3.005 cm. The distribution function might also be skewed if, for example, a part has been manufactured in a fixture which is showing signs of progressive wear.

The ANSI dimensioning and tolerancing standards do not specify what statistical distribution is implied by expressions of the form 3.000±.005 cm.[5] This omission is actually necessary, because' the shape of the distribution function depends on the manufacturing process, so that the choice of this shape is best left to the production engineer. In the system described in this paper, an arbitrary choice was made to restrict the class of allowed distributions to be either uniform or normal. This choice was made for the sake of convenience and does not represent any inherent limitation in the method.

Part Ensembles

In most parts modeling systems the user describes each part in terms of numbers which are entered directly into a data structure. This data structure, therefore, represents a particular *instance* of a part rather than an *ensemble* of similar parts. For the Monte Carlo simulation of tolerances, however, it is necessary that the parts modeling system provide some simple means of representing ensembles. What is needed, therefore, is a system in which the user describes parts not in terms of *numbers* but in terms of *parameters* that are assigned numerical values when a part is instantiated. The advantage of such a system for this Monte Carlo simulation is that a random number generator may be used to assign values to these

parameters.

The use of parameters to characterize arbitrary attributes of parts is one of the principle features of the Procedural Geometric Modeling System (PGMS) developed earlier by this author." [7] This modeling system was therefore used for the current study. The reader is referred to the earlier publication for details concerning the way in which PGMS represents 3-dimensional objects as PL/I procedures. A brief summary of the main features of this system are included here for the sake of completeness. Further features wilt be explained in subsequent sections of this paper as the need arises.

In PGMS, a hypothetical part whose name is "widget" and which has two attributes might be invoked by the calling sequence

```
CALL SOLID(WIDGET,A,B);
```

The generic widget itself would be represented by a PL/I procedure whose entry point is named WIDGET. This procedure would describe how the widget is hierarchically constructed out of its component subparts. These subparts might be positive SOLID's or negative HOLE's For example,

```
WIDGET. ENTRY (A,B);
CA LL SOLID(CUBOID,A ,A ,B);
CALL HOLE(CUBOID,A,A/2,B-10);
RETURN,
```

A library of parts procedures already exrsts which starts with the primitive POINT and includes such objects as LINE, CUBOID, CONE, WEDGE, CYLNDR, and HEMISPH. More complicated objects have also been coded, up to the level of complexity of IMM, which represents the IBM Research mechanical manipulator, and SUARM, which represents the Stanford University arm.

In addition to parts procedures, PCMS provides routines to perform transformations in 3-dimensional space. For example, if the generic widget were translated by C units along the Y-ax is and then rotated by D degrees about the X-axis, the calling sequence would be

```
CALL YTRAN(C);
CALL XROT(D);
CALL SOLID(WIDGET,A,B);
```

A particular instance of a widget would be invoked by assigning values to the parameters. For example,

```
CALL YTRAN( 12);
CA LLXROT(30);
CA LL SOLID( W IDGET,3.000,16.5);
```

An ensemble of 500 similar widgets would be represented by the calling sequence

```
DO I= 1 TO 500;
    CALL YTRA N( 12+RAND(-0.1,+0.3));
    CA LL XROT( 30+GA USS(2.5));
    CALL SOLID(WIDGET,3.000+RAND(-.005,+.005),16.5+RAND(-.2,+.2));
    END;.
```

where the function RAND(X,Y) returns a random number uniformly distributed on the interval from X to Y, and the function GAUSS(Z) returns a random number normally distributed with mean 0 and standard deviation Z.

Semantics

Once an ensemble of parts has been represented, PCMS provides a way to derive properties from the representation. This process is referred to as attaching semantics to the representation. The first step is to code a semantic routine which can compute a desired property, For example, the routine TOTVOL shown below adds up the volume of all positive and negative CUBOID's in any object.

```
TOTVOL: PROCEDURE (NODE,X,Y,Z);
DECLARE NODE. ENTRY;
IF NODE=CUBOID THEN VOLUME=VOLUME+POLARITY*X*Y*Z;
RETURN;
END TOTVOL;
```

Next, calls to system routines BEGIN, EXEC, and END are used to attach these semantics to the system and the part procedure of interest is executed. In the case of the ensemble of 500 widgets, one could print the volume of each widget with the following code.

```
DO I= 1 TO 500;
    VOLUME=0;                    /*INITIALIZE VOLUME*/
    CALL BEGIN(5000);            /*ALLOCATE STORAGE:::/
    CALL EXEC(TOTVOL);           /*ATTACH SEMANTICS:::/
    CALL YTRAN( 12+RAND(-0.1,+0.3));
    CALL XROT( 30+GAUSS(2.5));
    CALL SOLID(WIDGET,3.000+RAND(-.005,+.005),16.5+RAND(-.2,+.2));
    CALL END;                    /*DEALLOCATE STORAGE*/
    PUT SKIP DATA (VOLUME);      /*PRINT WIDGET VOLUME*/
    END;
```

Generalizing from this example, one can easily see how to 'provide semantics to display histograms of almost any desired properties of the ensemble. What is probably not clear from this example is the fact that for more realistic parts, the hierarchy of subpart calls

involves so much computation that execution is usually rather slow. For instance, when the procedure for the Stanford arm is executed on an IBM 370/168 running the VM time sharing system with 120 users, each instantiation takes about 6 seconds of virtual CPU time and 1 minute of elapsed time. Derrving the properties of an ensemble of 500 Stanford arms would therefore require about 8 hours of elapsed time. This number is prohibitively long for casual use of the system. However, 8 hours of elapsed time in simulating a complex mechanism would certainly not be excessive if the derived properties were to reveal a design deficiency which would have taken months to correct had the hardware been built first.

Another fact which is not clear from the example above is that parts of typical complexity require the allocation of several hundred thousand bytes of intermediate storage The Stanford arm procedure, for instance, requires nearly 300K of storage The reason behind thrs need for intermediate storage relates to the detailed implementation of PGMS, a topic which is discussed in my prior publication and which will be omitted here.

EXAMPLES

Rivet-Hole Bracket

The first example chosen to illustrate Monte Carlo tolerancing in PGMS is similar to the wet-hole bracket used as the example in the GM paper. A few changes were made because the original drawrng shows only a partial view of the bracket in two dimensrons, while in PGMS it is desirable to model the part completely and in three dimensions.

The modified rivet-hole bracket may be represented by the following code:

```
RHBRAK: ENTRY (X1,Y1,RAD1,X2,Y2,RAD2,ANG,THICK,LENG,NSECT);
DECLARE RHBFRAME(4,4) FLOAT;
CALL STORE(RHBFRAME);
CALL SOLID(WEDGE,THICK,LENG,ANG,1);        /*BRACKET*/
CALL XYZTRAN(X1,Y1,0);
CALL HOLE(CYLNDR,THICK,RAD1,NSECT);        /*HOLE 1*/
CALL RECALL(RHBFRAME);
CALL XYZTRAN(X2,Y2,0);
CALL HOLE(CYLNDR,THICK,RAD2,NSECT);        /*HOLE 2*/
RETURN,
```

The call in the above code to the PGMS routine STORE is used to save the current coordinate frame in the local array RHBFRAME. Subsequently, the current frame is translated from the corner of the bracket to the position of the first hole. The current frame is then returned to the bracket corner by the RESTORE routine, so that it may subsequently be translated to the position of the second hole.

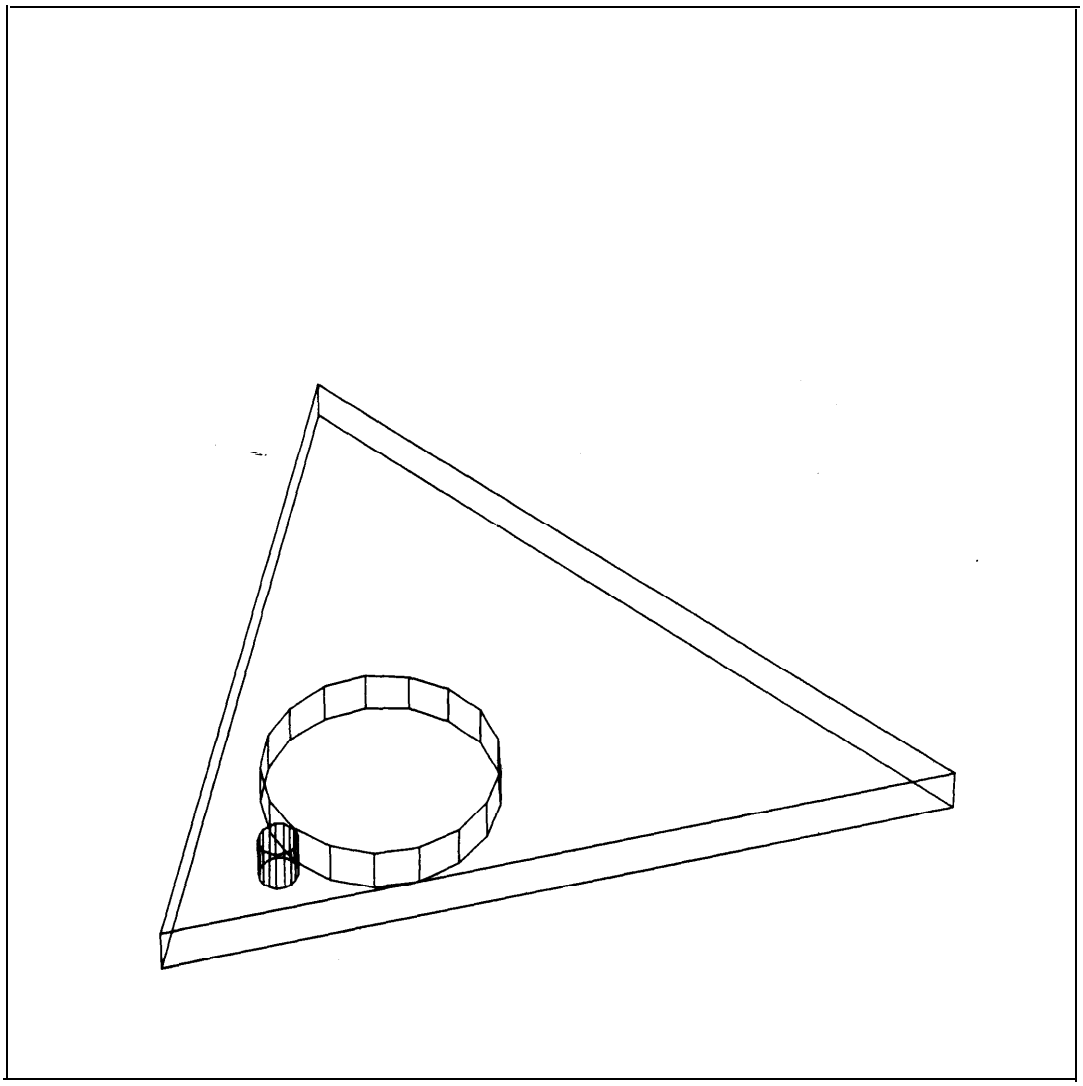


Figure 1: Drawing of Rivet-Hole Bracket

```
RHBRAK ( )
    WEDGE (1)
        GLINE (1,1)
            LINE (1,1,1)
                POINT (1,1,1,1)
                POINT (1,1,1,2)
        GLINE (1,2)
            LINE (1,2,1)
                POINT ( 1,2,1,1)
                POINT (1,2,1,2)
        CLINE (1,3)
            LINE (1,3,1)
                POINT (1,3,1,1)
                POINT (1,3,1,2)

            . . . (a total of 9 GLINE'S)

    CY LNDR (2)
        GLINE (2,1)
            LINE (2,1,1)
                P O I N T (2,1,1,1)
                POINT (2,1,1,2)

            . . . (a total of 3*NSECT GLINE'S)

    CYLNDR (3)
        .GLINE (3,1) .
            LINE (3,1,1)
              . POINT (3,1,1,1)
                POINT (3,1,1,2)

            . . . (a total of 3*NSECT GLINE'S)
```

Figure 2: Rivet-Hole Bracket **Subpart Hierarchy**

The ten parameters of this procedure represent the seven dimensions subject to tolerancing, the part thickness and length, and the number of sectors used in approximating the cylindrical holes by polyhedra. The effect of this polyhedral approximation can be seen in Figure 1 which was generated by attaching a standard graphics semantic routine to the RHBRAK procedure,

The RHBRAK procedure represents a subpart hierarchy of 40+24*NSECT nodes as indicated in Figure 2. At the top level, the RHBRAK consists of a solid WEDGE and two CYLNDR holes. The WEDGE in turn is composed of nine GLINE's (general lines), each of which is made out of one LINE with two end POINT's. Every level in this hierarchy can be referred to by a unique *subaddress*, also shown in Figure 2. For instance, the LINE along the bottom left edge of the RHBRAK has a subaddress of (1,3,1). The importance of these subaddresses will become clearer in the discussion which follows.

In the GM paper, the designer is concerned with the clearance between the two holes and the clearances between the second hole and the edges of the part. In order to study these resultants, the following semantic routine might be used.

```
BRAKRES: PROCEDURE (NODE,X 1,Y 1,RAD 1,X2,Y2,RAD2);
DECLARE (RGHTEDGE,LEFTEDGE,HOLE1,HOLE2) POINTER;
DECLARE NODE ENTRY;
IF NODE-RHB,RAK THEN DO;
      CALL DEFINE (RGHTEDGE, 1,2,1);
      CALL DEFINE (LEFTEDGE, 1,3,1);
      CALL DEFINE (HOLE 1,2);
      CALL DEFINE (HOLE2,3);
      CLEAR 1-DISTOO(HOLE1,HOLE2)-RAD I-RAD2;
      CLEAR2-DISTOX(HOLE2,RGHTEDGE);
      CLEAR3-DISTOX(HOLE2,LEFTEDGE);
      END;
RETURN;
END BRAKRES;
```

The DEFINE routine of PCMS is used to associate a PL/I pointer variable with any previously specified frame in the part hierarchy. The first argument in the call to DEFINE gives the name of the pointer variable and the subsequent arguments give the subaddress in the part hierarchy. Encoding these subaddresses requires that the user have a manual which summarizes the subpart hierarchy generated by each procedure in the part library and shows drawings of the basic volume shapes. Understanding subaddresses is currently the most tedious aspect of PCMS.

The function DISTOO invoked in this semantic routine returns the distance from Origin to Origin (00) of the two specified frames. The function DISTOX returns the distance from Origin to X-axis (OX) of the two specified frames. In order to have written this code it is necessary to have known that every LINE runs along the X-axis of its frame, and that every

CYLNDR runs along' the positive Z-axis of its frame. Thus CLEAR1, CLEAR2, and CLEA R3 are the desired clearances. It can be seen from this example that the polyhedral approximation has absolutely no effect on the statistical properties of these clearances.

Finally, a short program may be written to attach these semantics to the system and print the three clearances for each of 500 rivet-hole brackets.

```
DO I-l TO 500;
    CALL BEGIN(50000);
    CALL EXEC(BRAKRES);
    CALL SOLID(RHBRAK,1.325+GAUSS(.005/3),      /*X1*/
                      .875+GAUSS(.005/3),       /*Y1*/
                      .2+RAND(-.0075,.0075),    /*RAD1*/
                      2.525+GAUSS(.005/3),      /*X2*/
                      1.615+GAUSS(.005/3),      /*Y2*/
                      1.2+RAND(-.0075,.0075),   /*RAD2*/
                      67+RAND(-.25,+.25),       /*ANG*/
                      0.25,                     /*THICK*/
                      8.0,                      /*LENG*/
                      1);                       /*NSECT*/
    CALL END;
    PUT SKIP DATA (CLEAR1,CLEAR2,CLEAR3);
    END;
```

Because execution time varies roughly in proportion to the total number of nodes in the subpart hierarchy, NSECT has been set to 1 here. This simulation of 500 rivet-hole brackets takes about 3 minutes of CPU time on an IBM 370/168.

For this example, using PGMS to model tolerances is somewhat more difficult than using the GM system, largely because the tedium of understanding subaddresses outweighs that of writing down a few trigonometric formulas. As the examples become more complicated, however, the subaddress problem remains about constant, while the trigonometry problems become much worse. The overall balance therefore swings in favor of PGMS.

Box Manufacture

This example of Monte Carlo tolerancing is concerned with a manufacturing process in which 4 holes are drilled into a rectangular box. The holes are made by a gang drill with drill bits held in four separate chucks, while the box is held in a fixture attached to the drill bed. The box is 12 cm long, 8 cm wide, and 4 cm high, and the four corner holes have radius 3 mm and depth 2.5 cm and are nominally 1 cm from each edge.

Tolerance errors in the positions of the holes are generated because the fixture may be translated or rotated slightly in the plane of the drill bed, and each of the four drill chucks

may be radially displaced slightly from its nominal position. To make the example somewhat more interesting, it will be assumed that the rotational error in positioning the fixture is about an axis which runs through a corner rather than the center of the box.

Each of the drill bits may be modeled as a cylinder which has been radially displaced by RADERR in a random direction from its desired position.

```
DRILBIT: ENTRY (RADERR,LENG,RAD,NSECT);
CALL ZROT(RAND(0,360));
CALL XTRAN(RADERR);
CALL SOLID(CYLNDR,LENG,RAD,NSECT);
RETURN;.
```

An ensemble of boxes manufactured by this process may then be represented as a cuboid with holes cut out by the four drill bits.

```
RECTBOX: ENTRY (X,Y,Z,LENG,RAD,NSECT,
                XERR,YERR,ANGERR,RADERR);
CALL SOLID(CUBOID,X,Y,Z);                          /*BLOCK*/
CALL ZROT(ANGERR);
CALL  XYZTRAN(XERR,YERR,Z-LENG);
CALL XYZTRAN(1,1,0);
C A L L HOLE(DRILBIT,RADERR,LENG,RAD,NSECT); /*HOLE 1*/
CALL  XTRAN(X-2);
C A L L HOLE(DRILBIT,RADERR,LENG,RAD,NSECT); /*HOLE 2*/
CALL  YTRAN(Y-2);
C A L L HOLE(DRILBIT,RADERR,LENG,RAD,NSECT); /*HOLE 3*/  . .
C A L L XTRAN(2-X);
CALL  HOLE(DRILBIT,RADERR,LENG,RAD,NSECT);  /*HOLE 4*/
RETURN;
```

The next step is to code a semantic routine which can derive the coordinates of the four holes with respect to the coordinate system of the box.

```
HOLFIND:  PROCEDURE  (NODE);
DECLARE (HOLE 1,HOLE2,HOLE3,HOLE4) POINTER;
DECLARE  NODE  ENTRY;
IF NODE-RECTBOX THEN DO;
      CALL DEFINE (HOLE 1,2);  CALL ORIGIN (HOLE 1,POS 1);
      CALL  DEFINE (HOLE2,3); CALL ORIGIN (HOLE2,POS2);
      CALL  DEFINE (HOLE3,4); CALL ORIGIN (HOLE3,POS3);
      CALL  DEFINE (HOLE4,5); CALL ORIGIN (HOLE4,POS4);
      END;
RETURN;
END  HOLFIND;
```

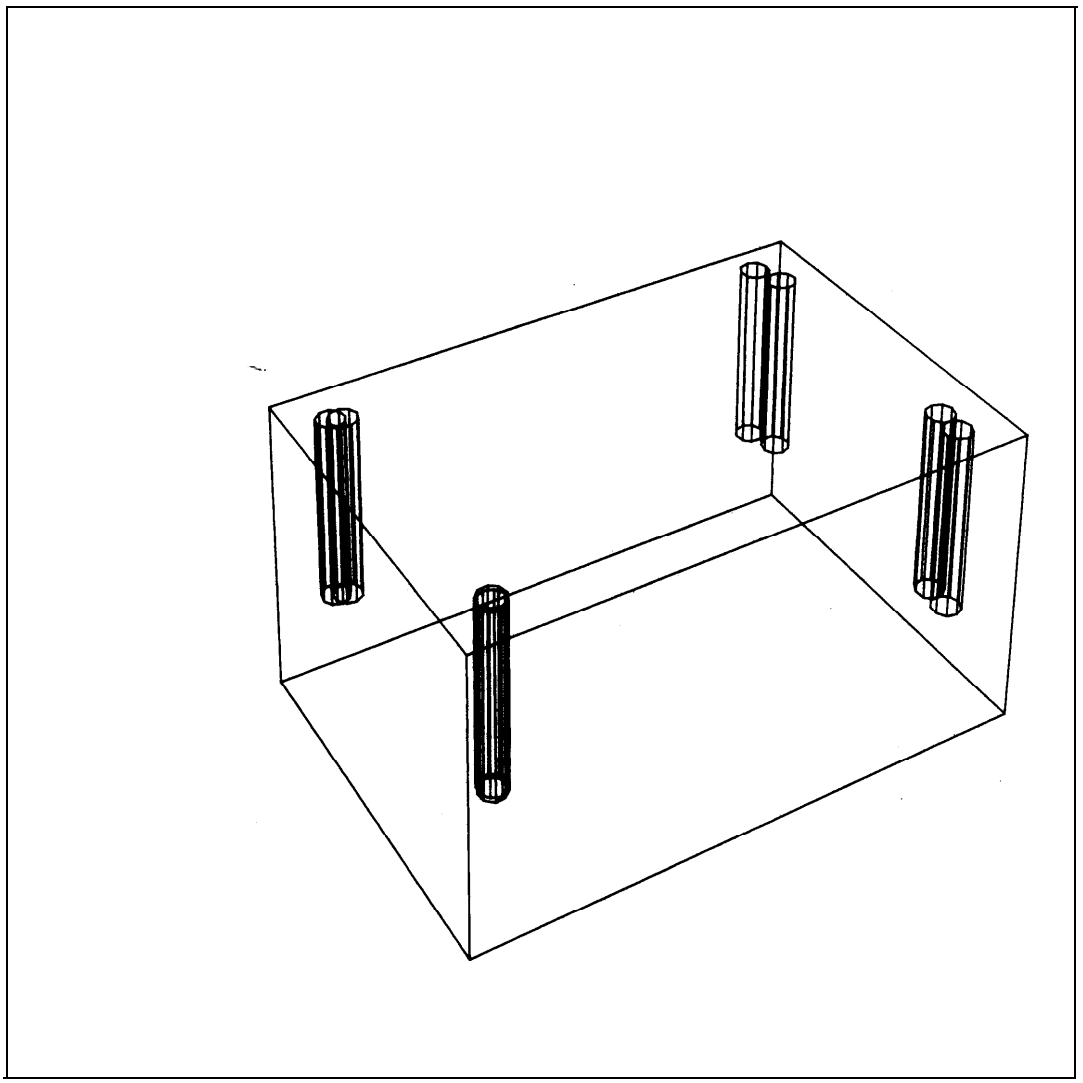


Figure 3: DoubleExposure Drawing of Rectangular Box

The PCMS routine ORIGIN returns the origin vector associated with the frame of the object pointed to by the first argument. Finally, the locations of each of the four holes, in an ensemble of 500 boxes may be printed by attaching these semantics and executing the RECTBOX.

```
DECLARE (POS 1(3),POS2(3),POS3(3),POS4(3)) FLOAT;
 DO I-1 TO 500;
     CALL  BECIN(80000);
     CALL EXEC(HOLFIND);
     CALL SOLID(RECTBOX,12,8,4,        /*X,Y,Z*/
                2.5,0.3,1,             /*LENG,RAD,NSECT*/
                GAUSS(0.1/3),          /*XERR*/
                GAUSS(0.1/3),          /*YERR*/
                RAND(-2.5,2.5),        /*ANGERR*/
                GAUSS(0.05/3));        /*RADERR*/
     CALL END;
     PUT SKIP DATA (POS 1,POS2,POS3,POS4);
     END;
```

Execution time is about 8 minutes on an IBM 370/168. A "double-exposure" drawing showing overlapping views of two boxes in the ensemble appears in Figure 3. This drawing was generated by attaching a standard graphics semantic routine and calling the RECTBOX procedure twice. The fact that graphics are produced so easily within PGMS is of considerable help in verifying that the simulation is working properly.

One aspect of this simulation which is perhaps unrealistic is that the fixture is perturbed for each box. in the ensemble. In an actual manufacturing operation, on the other hand, the fixture would be locked in place, The statistical distributions obtained in the actual, manufacturing operation would therefore be narrower than those derived from this simulation.

What has been simulated here is an ensemble of boxes produced by Independent setups as opposed to an ensemble produced by a *fixed* setup. In most cases of batch production, this simulation would be good enough for all practical purposes. One can imagine situations, however, in which the independent setup assumption is not appropriate. For instance, if pairs of consecutive boxes were to be attached to one another, the fact that both were produced on the same setup might be important. For this case, the code would have to be changed to simulate pairs of boxes instead of single boxes.

Actually, the box would probably be manufactured by trying a succession of setups until one was found which yielded satisfactory boxes, and this setup would then be retained for the remainder of the batch. Simulating the resulting ensemble is possible within PGMS, but it entails modeling the conditions used to determine whether or not the setup is satisfactory. Modeling conditional decisions is discussed briefly in the section of this paper dealing with extensions of the Monte Carlo method.

Box and Lid Assembly

This example is concerned with attaching a lid to the box of the previous example. The lid is 12 cm by 8 cm by 0.5 cm thick and is assumed to have been manufactured in the same manner as the box. At assembly time, a fixture is used which holds the lid rigidly in place on top of the box in such a way that the edges line up perfectly. The issue is whether or not the holes in the lid are aligned sufficiently well with those in the box to allow four screws to be inserted.

A procedure which represents both the box and its lid is shown below.

```
BOXNLID: ENTRY (X,Y,ZBOX,ZLID,LENG,RAD,NSECT,
                XERRB,YERRB,ANGERRB,RADERRB,
                XERRL,YERRL,ANGERRL,RADERRL);
CALL SOLID(RECTBOX,X,Y,ZBOX,LENG,RAD,NSECT,        /*BOX*/
                XERRB,YERRB,ANGERRBB,RADERRB);
CALL ZTRAN(ZBOX);
CALL SOLID(RECTBOX,X,Y,ZLID,ZLID,RAD,NSECT,        /*LID*/
                XERRL,YERRL,ANGERRL,RADERRL);
RETURN;
```

The next step is to code a semantic routine which computes the alignment errors for each of the four pairs of holes.

```
ALIGNER: PROCEDURE (NODE);
DECLARE (BOX,LID) POINTER;
DECLARE NHOLE BINARY FIXED;
DECLARE NODE ENTRY;
IF NODE-BOXNLID THEN DO;
     FOR NHOLE=1 TO 4 DO;
          CALL DEFINE (BOX, 1,NHOLE+1,1);
          CALL DEFINE (LID,2,NHOLE+1,1);
          ERROR(NHOLE)=DISTOZ(BOX,LID);
          END;
     END;
RETURN;
END ALIGNER;
```

The subaddresses in these DEFINE statements identify frames of corresponding CYLNDR holes in the box and lid. The function DISTOZ returns the distance from the Origin to Z-axis (OZ) of these two frames.

If it is assumed that the assembly process is unsuccessful whenever any of the four screw hole misalignments exceeds 2 mm, a simple procedure can be written to determine the number of successful assemblies in an ensemble of 500 boxes and lids.

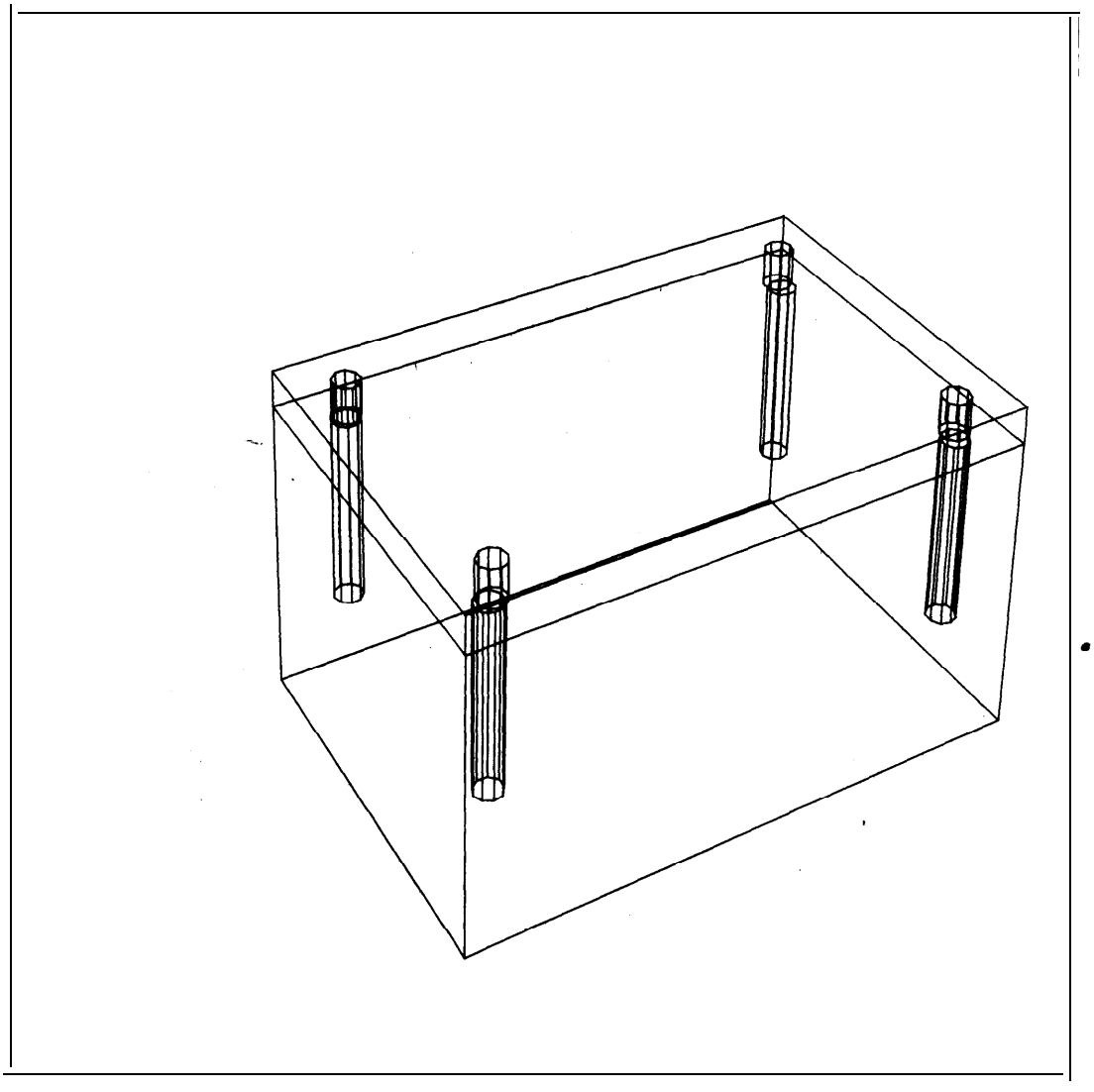


Figure 4: Drawing of Unsuccessful Box and Lid Assembly

[18]

```
DECLARE ERROR(4);
DO I- 1 TO 500;
      NSUCCESS=0;
      CALL BEGIN( 120000);
      CALL  EXEC(ALICNER);
      CALL SOLID(BOXNLID,12,8,4,0.5,        /*X,Y,ZBOX,ZLID*/
                 2.5,0.3,1,                 /*LENG,RAD,NSECT*/
                 GAUSS(0.1/3),              /*XERRB*/
                 GAUSS(0.1/3),              /*Y ERR B*/
                 RAND(-2.5,2.5),            /*ANGERRB*/
                 GAUSS(0.05/3),             /*RADERRB*/
                 GAUSS(0.1/3),              /*XERRL*/
                 GAUSS(0.1/3),              /*Y ERR L*/
              R  A  ND(-2.5,2.5),           /*ANGERRL*/
                 GAUSS(0.05/3));            /*RADERRL*/
      CALL END;
      IF ERROR(1)<.2
         & ERROR(2)<.2
         & ERROR(3)<.2
         . & ERROR(4)<.2 .
         THEN NSUCCESS=NSUCCESS+ 1;
      END;
PUT SKIP DATA (NSUCCESS);
```

When this program is executed, it determines that 27% of the assemblies would be successful. A bout 10 minutes of CPU time are required to obtain this result using an IBM 370/168. A drawing of one of the unsuccessful assemblies is shown in Figure 4.

Since in principle the lids are symmetric, it is also possible to generate an ensemble in which the lids have been randomly flipped upside down or rotated 180 degrees in the horizontal plane between the time of manufacture and the time of assembly. Such an ensemble simulates the common industrial practise of throwing freshly manufactured parts into a tote bin. The simulation then yields 19% successful assemblies. The reason why this percentage is much lower than the previous one is related to the fact that the rotational error in the fixture was **assumed** to be about an axis which ran through a corner of the box rather than through its center.

Stanford Arm

The final example is taken from the field of computer controlled manipulators. Currently, two manipulator arms are being used at the Stanford University Artificial Intelligence Laboratory to study problems in industrial automation. Figure 5 shows a drawing of one of these arms holding a power screwdriver and a screw. Although the arm had been modeled much earlier by Baumgart,[16] this picture was obtained by using PGMS procedures instead.

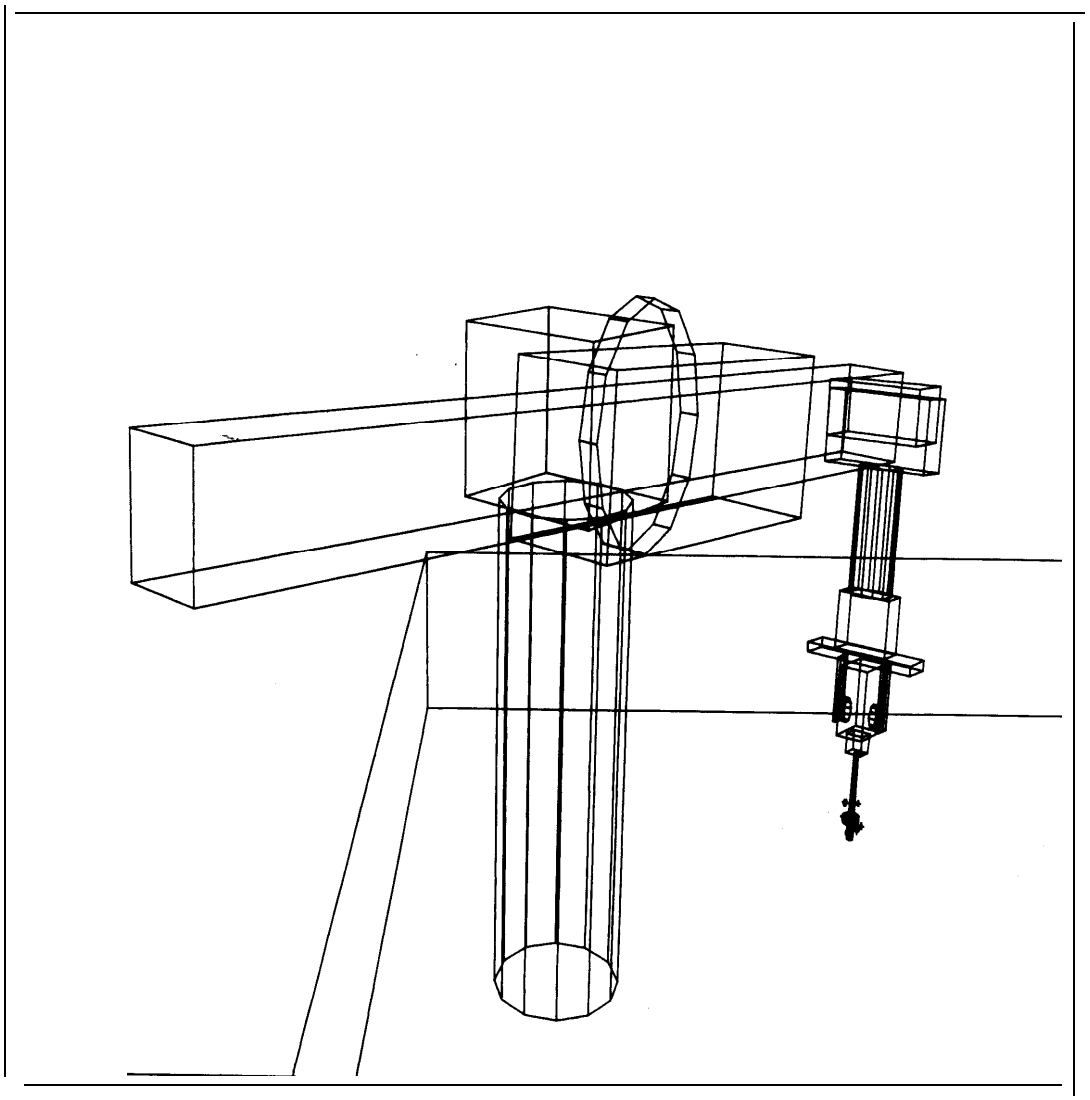


Figure 5: Drawing of Stanford Arm

[20]

In advanced manipulator applications, it is frequently necessary to perform *inspection* to measure the locations of objects or even simply to determine whether an object is present or missing. For instance, since a screw can easily fall off a screwdriver, it may be desirable to verify that the screw is actually still on the end of the screwdriver.

Both touch sensing and computer vision have been_ used in the past to perform this type of inspection.[20] Currently, Bolles is working on a more systematic approach to doing inspection by computer vision.[21] One of the main problems encountered in this endeavor relates to the fact that the location of the end of the screwdriver is not known precisely by the program, because of backlash and compliance in the manipulator. The vision program, therefore, can not simply look at the nominal location of the screw, Instead, it must search the image, over a finite region whose extent depends on the tolerance errors of the manipulator joints.

The purpose of this example is to show that it is possible to do a Monte Carlo simulation of as complex an object as a manipulator, without having to write down the trigonometric formulas for the location of its gripper as a function of all the joint angles. An ensemble of 10 Stanford arms may--be modeled simply by coding

```
DO I=1 TO 10;
    CALL SOLID(SUA RM,-41+RAND(-2,2),  /*JOINT ANGLE 1*/
                    -92+RAND(-2,2),    /*JOINT ANGLE 2*/
                    15+RAND(-.2,.2),   /*JOINT ANGLE 3*/
                    -90+RAND(-2,2),    /*JOINT ANGLE 4*/
                    90+RAND(-2,2),     /*JOINT ANGLE 5*/
                    O+RA ND(-2,2),     /*JOINT ANGLE 6*/
                    1.5);              /*GRIPPER OPENING*/
    END;
```

It is only slightly more difficult in PGMS to model an ensemble of arms, each of which is holding a screwdriver with a screw. A semantic routine may then be supplied to draw the first object in this ensemble, and for all subsequent objects to draw a little asterisk at the location of the tip of the screw, as shown in Figure 5. Alternatively, semantics may be provided to compute the parameters of an error ellipse in the image plane, so that a vision program will know what region must be searched to verify the presence of the screw.

EXTENSIONS

In all four of the preceding examples, the simulation of tolerancing was used to derive *independent* distributions of resultant properties. It is also possible to derive *conditional* distributions of resultant properties. The need for considering conditional distributions arises primarily whenever there are steps in the manufacturing and assembly process which

involve conditional actions. Actually, such actions are quite common in discrete parts production, although they tend to be overlooked because these steps are usually implicitly assumed.

For instance, one expects an assembly worker to know without being told that

IF the lid doesn't fit
THEN throw it out and try another one
ELSE attach it

Alternatively, the worker might ignore any requirement of interchangeability and save the nonfitting lid until a matching box was found. In either case, the statistical properties of the resulting assemblies would no longer be the same. This fact is true whether or not the conditional instructions are stated explicitly.

Not all conditional actions have the simple form IF . . . THEN . ..ELSE. For example, the assembly process might involve sliding the lid until it is aligned with the box. This step would move each lid by a different amount, depending on the initial misalignment of that particular lid and box.

In a PGMS tolerancing simulation, the addition of steps which simulate conditional actions is a straightforward process, provided that these actions can be stated in the form of procedures which involve spatial transformations no parse than rotations and translations by well defined amounts. For an IF . . . THEN . . . ELSE action, one simply adds the appropriate IF . . . THEN . . . ELSE clause to the program. A problem arises, however, that there are conditional actions which can not be easily expressed in the form of well defined procedures.

A common and insidious example of such actions relates to the way parts are often chamfered to make' the assembly process easier. As the assembly is performed, the chamfers force parts into slightly different positions and alter their subsequent statistical properties. The effect of a chamfer in locating a single pin can be expressed fairly easily in the form of a procedure, but for more than one pin the effect of chamfering becomes very difficult to state explicitly.

The effect of chamfers is .a specific case of a general process which may be called fitting or *accommodation.* Case studies performed at the Charles Stark Draper Laboratory indicate that in typical industrial assemblies, roughly 15% of the steps involve accommodation.[22] Although this process is industrially important, it is very difficult to simulate except in the simplest situations. For instance, it is well known that the way to attach a lid to a box is to put all four screws in loosely and then tighten them, rather than tightening each one immediately. Unfortunately, even in this case it is not known how to express the exact process of accommodation in the form of a well defined procedure.

However, it is possible to *approximate* many accommodation processes.. For example, in the

box assembly one can say that the first screw to be loosely inserted produces a translation of the lid such that its hole aligns with the corresponding box hole. The second screw produces a rotation of the lid which makes the vector from the first to the second lid hole align with the corresponding box vector, followed by a translation of the lid along this vector to make the two alignment errors equal and opposite. The third screw only produces a translation orthogonal to the previous vector, and the fourth screw has no effect. Clearly, this procedure is only. an approximation to what really happens, but the chances are that it is a good enough approximation for most practical purposes. An alternative approximation would be to say that each successive screw produces a transformation of the lid to a new position such that the sum of the squares of of the alignment errors is minimized. In either of these cases, one can easily add to PGMS proocedures which simulate the approximate accommodation process.

Another extension of Monte Carlo tolerancing would be to simulate the process of making measurements with imperfect measuring tools. For example, suppose a computer vision system is used to locate the position of a hole in a part so that a manipulator can insert a screw. This measurement is limited by the camera resolution, which may be on the order of one picture element in the scanning array. The measurement is also limited by pan and tilt errors in aiming the camera. Projecting the camera errors from the image plane back to the actual hole in three-dimensional space will generally give an elongated region within which the location of the hole can not be resolved. If several features of a part are located in this manner, the position and orientation of the part itself may be derived. All of these steps can be simulated within PGMS.

It is also possible to simulate part imperfections of a much grosser nature than those normally considered in tolerancing. For instance, Agin has written a computer vision program which inspects lamp bases for displaced or *missing* grommets.[23] In order to simulate an ensemble of lamp bases with an appropriate range of defects, one could represent the generic lamp base by a routine with parameters specifying whether or not the grommets are present.

```
LAMPBAS: ENTRY (GROM 1,X 1,Y1,GROM2,X2,Y2);
CALL XYZTRAN(X 1,Y 1,0);
IF GROM1=1
      THEN CALL SOLID(GROMMET);
CALL XYZTRAN(X2-X1,Y2-Y 1,O);
IF GROM2=1
      THEN CALL SOLID(GROMMET);
RETURN;
```

Cross defects of this type are quite common in industry. The most familiar example is that roughly 2% of all machine screws are ordinarily defective. Some have no heads, while others have no slots or no threads. The defective fraction may be reduced by preinspection, but for most applications the additional cost can not be justified. It is therefore worth emphasizing the fact that errors of these types can also be simulated ‘within a Monte Carlo parts

tolerancing system.

CONCLUSION

This paper has described a Monte Carlo approach to the simulation of tolerancing and other forms of imprecision in discrete' parts manufacturing and assembly. An implementation of the method, based on the Procedural Geometric Modeling System developed earlier by this author, is illustrated by four specific examples, one of which was chosen from the field of assembly by computer controlled manipulators.

There appears to be a pressing need for simulation techniques relating to discrete parts manufacturing and assembly. The assembly process is strongly affected by imprecise components, imperfect fixtures and tools, and inexact measurements. It is often necessary to design higher precision into the manufacturing and assembly process than is functionally needed in the final product, Production costs are highly dependent on specified tolerances and the resultant product yields.

The technique described in this paper can provide production engineers with a systematic way of analyzing the stochastic implications of tolerancing and other forms of imprecision.

ACKNOWLEDGEMENT

This paper was partially motivated by Russell Taylor's work on high level languages for computer con trolled manipulators. One of his programs determines allowed loci of workpieces by resolving symbolic geometric constraints. The paper was also motivated by the computer vision research of Bob Bolles. One of his programs calculates the region of an image to be searched for a desired feature of a workpiece that has been displaced slightly from its nominal position. Discussions with Taylor and Bolles in the early stages of this work have proved very valuable. Their results, incidentally, will be published soon as part of their doctoral dissertations.

This work was performed at the Stanford AI Lab, as part of the Computer Integrated Assembly Systems project headed by Tom Binford. I want to thank Peter Will, manager of the Automation Research project at the IBM T. J. Watson Research Center, from which I was on sabbatical leave, for making my year at SAIL possible.

Finally, I want to acknowledge the logistical assistance of Mike Blasgen and Larry Lieberman in this work.

REFERENCES

[1] W. V. Tipping, *An Introduction to Mechanical Assembly,* Business Books, London, England, 1969.

[2] *An Introduction to PADL,* Production Automation Project Technical Memorandum TM-22, University of Rochester, December 1974.

[3] A. A. G. Requicha, N. M. Samuel, and H. B. Voelcker, *Part and Assembly Description Languages* - II, Production Automation Technical Memorandum TM-20a, University of Rochester, revised November 1974.

[4] *Discrete Part Manufacturing: Theory and Practice,* Production Automation Project Technical Report TR-l-1, University of Rochester, 1974.

[5] *Dimensioning and Tolerancing,* American National Standards Institute Report ANSI Y 14.5-1973, published by IEEE, New York, 1973.

[6] Lowell W. Foster, *Geometric Dimensioning and Tolerancing: A Working Guide,* Addison-Wesley Publishing Co., Reading, Massachusetts, 1970.

[7] Earlwood T. Fortini, *Dimensioning For Interchangeable Manufacture,* Industrial Press Inc., New York, 1967.

[8] Harold W. Gugel, *Monte Carlo Simulation With Interactive Graphics,* GM Research Publication GMR- 153 1, General Motors Corporation Research Laboratories, Warren, Michigan, October '1974.

[9] John M. Hammersley and David C. Handscomb, *Monte Carlo Methods,* Wiley, New York, 1964.

[10] Gerald J. Agin, *Representation and Description* of Curved *Objects,* Stanford Artificial Intelligence Laboratory Memo AIM-173 and Stanford University Computer Science Report STAN-CS-305, October 1972.

[11] Gerald J. Agin and Thomas 0. Binford, *Computer Description* of *Curved Objects,* Third International Joint Conference on Artificial Intelligence, Stanford, August 1973.

[12] Ramakant Nevatia, *Structured Descriptions of Complex Curved Objects for Recognition and Visual Memory,* Stanford Artificial Intelligence Laboratory Memo AIM-250 and Stanford University Computer Science Report STAN-CS-464, October 1974.

[13] I. C. Braid, *Designing With Volumes,* Cantab Press, Cambridge, England, 1974.

[14] I. C. Braid, *The Synthesis of Solids Bounded by Many Faces,* Communications of the ACM, Volume 18, Number 4, p. 209, April 1975.

[15] Bruce G. Baumgart, *Winged Edge Polyhedron Representation,* Stanford Artificial Intelligence Laboratory Memo AIM- 179 and Stanford University Computer Science Report STAN-CS-320, October 1972.

[16] Bruce G. Baumgart, *GEOMED,* Stanford Artificial Intelligence Laboratory Memo AIM-232 and Stanford University Computer Science Report STAN-CS-4 14, May 1974.

[17] David D. Grossman, *Procedural Representation of Three-Dimensional Objects,* IBM Research Report RC-5314, T. J. Watson Research Center, Yorktown Heights, N. Y., March 14, 1975; to be published in IBM Journal of Research and Development.

[18] Mark A. Lavin and Laurence I. Lieberman, *A System for Modeling Three-Dimensional Objects,* IBM Research Report RC-5765, T. J. Watson Research Center, Yorktown Heights, N. Y., December 17, 1975; to be published in IBM Journal of Research and Development.

[19] Mark A. Lavin, *MODFEAT: A System for Naming Polyhedral Features of Three-Dimensional Objects,* IBM Research Report RC-5764, T. J. Watson Research Center, Yorktown Heights, N. Y., December 17, 1975.

[20] Robert Bolles and Richard 'Paul, *The Use of Sensory Feedback in a Programmable* Assembly *System,* Stanford Artificial Intelligence Laboratory Memo AIM-220 and Stanford University Computer Science Report STAN-(X-396, October 1973.

[21] Robert C. Bolles, *Verification Vision Within a Programmable Assembly System: An Introductory Discussfan,* Stanford Artificial Intelligence Laboratory Memo AIM-275 and Stanford University. Computer Science Report STAN-CS-75-537, December 1975.

[22] J. Nevins, D. Whitney, S. Drake, D. Killoran, M. Lynch, D. Seltzer, S. Simunovic, R. M. Spencer, P. Watson, and A. Woodin, *Exploratory Research in industrial Modular Assembly,* Charles Stark Draper Laboratory Report R-921, Cambridge, Massachusetts, December 1, 1974 to August 31, 1975.

[23] C. Rosen, D. Nitzan, G. Agin, G. Andeen, J. Berger, J. Eckerle, G. Gleason, J. Hill, J, Kremers, B. Meyer, W. Park, and A. Sword, *Exploratory Research in Advanced Automation,* Stanford Research Institute Project 2591 Report 2, Menlo Park, California, August 1974.