Solution space navigation for geometric constraint systems

Meera Sitharam Adam Arbree Yong Zhou
Naganandhini Kohareswaran

Abstract. We study the well-documented problem of systematically navigating the potentially exponentially many roots or realizations of well-constrained, variational geometric constraint systems. We give a scalable method called the ESM or Equation and Solution Manager that can be used both for automatic searches and visual, user-driven searches for desired realizations. The method incrementally assembles the desired solution of the entire system and avoids combinatorial explosion, by offering the user a visual walkthrough of the solutions to recursively constructed subsystems and by permitting the user to make gradual, adaptive solution choices.

We isolate requirements on companion methods that are essential and desirable for efficient, meaningful solution space navigation. Specifically, they permit (a) incorporation of many existing approaches to solution space steering or navigation into the ESM; and (b) integration of the ESM into a standard geometric constraint solver architecture. We address the latter challenge and explain how the integration is achieved. Additionally, we sketch the ESM implementation as part of an opensource, 2D and 3D geometric constraint solver FRONTIER developed at the University of Florida.

Keywords. Root selection for geometric constraint systems. Well-constrained systems. Underconstrained and Overconstrained systems. Constraint graphs. Cyclical and 3D geometric constraint systems. Variational geometric constraint solving. Decomposition of geometric constraint systems. Degree of Freedom analysis. Conceptual design. Feature-based and assembly modeling.

University of Florida. Work supported in part by NSF Grant CCR 99-02025, NSF Grant EIA 00-96104. Corresponding author: sitharam@cise.ufl.edu.

1. INTRODUCTION

Geometric constraint systems have been studied in the context of variational constraint solving in CAD for nearly 2 decades For recent reviews of the extensive literature on geometric constraint solving see, e.g, [Hoffmann et al. 1998a; Kramer 1992; Fudos 1995b; Sitharam 2004c].

A *geometric constraint system* consists of a finite set of geometric objects and a finite set of constraints between them. The constraints can usually be written as algebraic equations and inequalities whose variables are the coordinates of the participating geometric objects. For example, a distance constraint of d between two points (x_1, y_1) and (x_2, y_2) in 2D is written as $(x_2 - x_1)^2 + (y_2 - y_1)^2 = d^2$. Most of the constraint solvers so far deal with 2D constraint systems, although some of the newer approaches including [Hoffmann et al. 1997; 1998b; 2001a; 2001b] [Hoffmann et al. 1999; Bruderlin 1986; Owen] [Sitharam 2004c; Hoffmann et al. 2004; Sitharam and Zhou 2004b; Sitharam et al. 2004; Lomonosov and Sitharam 2004; Oung et al. 2001; Sitharam 2004b], extend to 3D constraint systems.

A *solution or realization* of a geometric constraint system is the (set of) real zero(es) of the corresponding algebraic system. In other words, the solution is a class of valid instantiations of (the position, orientation and any other parameters of) the geometric elements such that all constraints are satisfied. Here, it is understood that such a solution is in a particular geometry, for example the Euclidean plane, the sphere, or Euclidean 3 dimensional space. A constraint system can be classified as *overconstrained*, *well-constrained*, or *underconstrained*. Well-constrained systems have a finite, albeit potentially very large number of *rigid* solutions; their solution space is a zero-dimensional variety. Underconstrained systems have infinitely many solutions; their solution space is not zero-dimensional. Overconstrained systems do not have a solution unless they are *consistently overconstrained*. Parts of a system could be (consistently) overconstrained, but the entire system could be underconstrained. Systems that are not underconstrained are therefore well or *well-overconstrained* and are called *rigid* systems.

Geometric constraint solvers need to deal with the standard and well documented problem of controlling the combinatorial explosion of the set of solutions of well-constrained or well-overconstrained systems. This goes hand in hand with classifying and steering through multiple generic solutions in a conceptually meaningful manner. Such systems may have exponentially many solutions in the number of geometric primitives. For example, even a simple ruler and compass construction in 2D permits 2 solutions for each constructed point (reflections are considered distinct since they cannot be obtained by a physical Euclidean transformation such as rotation).

One standard approach [Fudos 1995a; Fudos and Hoffmann 1996a; Owen] attempts to automatically pick a solution that is as close as possible to the appearance of the input sketch. Typically, chirality (or relative orientation) of the geometric primitives in the sketch is used as the guide. A related approach uses explicitly defined "navigation" constraints or "declarative solution selectors" provided by the user. The formal basis for this began with [Hendrickson 1992], and some of the more recent methods are surveyed in [Bettig and Shah 2003]. The use of genetic algorithms for dealing with these extra constraints are given in [Joan-Arinyo et al. 2003]. In general, these constraints could be consistent overconstraints (also called *redundant* constraints); chirality or relative orientation constraints; and other constraints related to chirality (by oriented matroid theory [Bjorner et al. 1993]) such as intersection or nonintersection of convex hulls or subsets of the points. They reduce to specific polynomial inequalities obtained from determinants of matrices of indeterminates.

A second approach gives the user an indexing [Bouma et al. 1995], of the solution space using a Decomposition Recombination (DR) plan of the constraint system as a guide. These DR-plans are generated by most constraint solvers to efficiently guide the algebraic-numeric solvers. Specifically, they break down the large polynomial system that arises from the entire geometric constraint system into small subsystems that can be realistically handled by algebraic-numeric solvers, since the latter typically have exponential time complexity.

The methods of [Bouma et al. 1995] provide steering for simple, 2D constraint systems that are triangle decomposable, or have linear DR-plans, similar to ruler and compass constructible systems, and extend immediately to 3D systems that permit solving for one geometric primitive at a time.

In this case, each such addition provides two "bifurcation" choices that are typically two reflections of the newly solved-for primitive. However there are some drawbacks. (i) Such DR-plans exist only for very special constraint systems (ii) Even for those systems, the method is either not scalable or requires the user to choose the of reflections for each geometric element apriori, without knowing whether such a solution exists; (iii) The method provides little or no visual guidance and does not provide the option of choosing reflections sequentially, adaptively, where later choices depend on earlier ones; (iv) Moreover, even in the case of triangular decomposable systems, in (consistently) overconstrained cases that occur often in practice, triangular decompositions cannot in general incorporate a user's conceptual feature decomposition, potentially denying the user's prefered navigation path. This problem is explained in detail in [Sitharam and Zhou 2004a].

1.1 Contribution and Organization

In Section 2, we give well-defined essential and desirable requirements on the companion methods for the ESM to be effective and briefly discuss literature on known companion methods that meet these requirements. In addition, the conceptual structures used by these companion methods are also crucial to the ESM and are described here.

In Sections 3, and 4 we present the main contribution: a third, mixed approach for interactively and visually navigating the solution space of the maximal well or well-constrained subsystems of general 2D or 3D systems G. It is an efficient, scalable, user friendly method, called *equation and solution manager (ESM)*. The method is modular and integrable in that it can be used by any geometric constraint solver that satisfies a well-defined set of basic requirements.

The advantages of this clear *modularization* of the ESM method are the following. (i) It can be integrated into *any* - currently standard - geometric constraint solver architecture as shown in Figure 1 [Oung et al. 2001], provided the companion methods satisfy these requirements: such an integrated implementation can be found in the FRONTIER 3D geometric constraint solver opensource software [Sitharam 2004b]. (ii) It can then easily incorporate many existing approaches to solution space steering or navigation, such as those mentioned earlier in this Section. The companion methods are the following.

–A *good* DR (decomposition-recombination) plan(ner) for general 3D geometric constraint systems, at least those that are generically not underconstrained.

–A *combinatorial cluster-system optimizer* (CCO) that converts each subsystem S (*cluster*) in a good DR-plan into a stable, independent system of constraints, whose algebraic complexity -i.e., number and degree of variables - has been optimized as far as combinatorially possible, i.e., without algebraic or numeric computation.

–An finite precision, interval or subdivision based *algebraic-numeric polynomial system solver* (Algebraic-Solver) that outputs all real solutions (intervals) of a stable, independent system of polynomial equations and inequalities with interval coefficients. While such solvers typically run for exponential time, they are used here only for solving small systems of low algebraic complexity.

–A graphical user interface (GUI) that has standard text and 2D graphic I/O capabilities, and basic 3D capabilities to display solutions the user, and communicate back his/her choices.

In Section 5, we give the conclusions.

2. COMPANION METHOD REQUIREMENTS

In order to describe the the ESM method and how it can be integrated into *any* geometric constraint solver as in Figure 1, it is necessary to give well-defined *essential* and *desirable* requirements on the companion

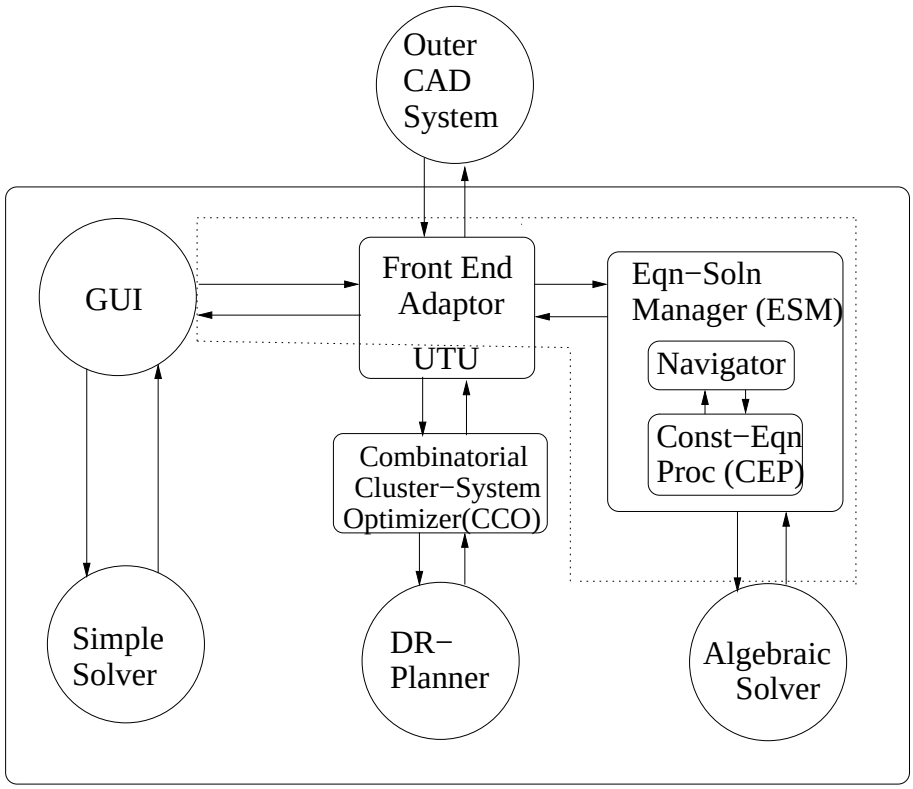

Fig. 1. A standard geometric constraint solver architecture: our contribution within dotted lines

methods - mentioned in the Introduction - for the ESM method to be effective.

Furthermore, the conceptual structures underlying these companion methods, such as geometric constraint graph, cluster, DR-plan etc., also underly the ESM and are described here. We additionally briefly discuss literature on known companion methods that meet these requirements.

2.1 DR-planner

In order to define DR-plans which are the underlying structure used by the ESM and in order to give the DR-planner companion method requirements, we first need some preliminaries about combinatorial analysis of geometric constraint systems. A *geometric constraint graph* $G = (V, E, w)$ corresponding to geometric constraint system is a weighted graph with vertex set (representing geometric objects) V and edge set (representing constraints) E; $w(v)$ is the weight of vertex v and $w(e)$ is the weight of edge e, corresponding to the number of *degrees of freedom (dofs)* available to an object represented by v and number of degrees of freedom removed by a constraint represented by e respectively.

For example, Figure 2 shows a 2D constraint systems and their respective dof constraint graphs. Figure 3 shows a 3D constraint systems whose graph have vertices of weight 3 (points) and edges of weight 1.

Note that the constraint graph could be a *hypergraph*, each hyperedge representing a constraint involving any number of vertices. A vertex induced subgraph represents a subsystem of the entire constraint system. The subgraphs can be classified as being *underconstrained, well-constrained, or well-overconstrained* - the latter two are called *rigid* subgraphs or *clusters*. The meaning is that the corresponding subsystems have the

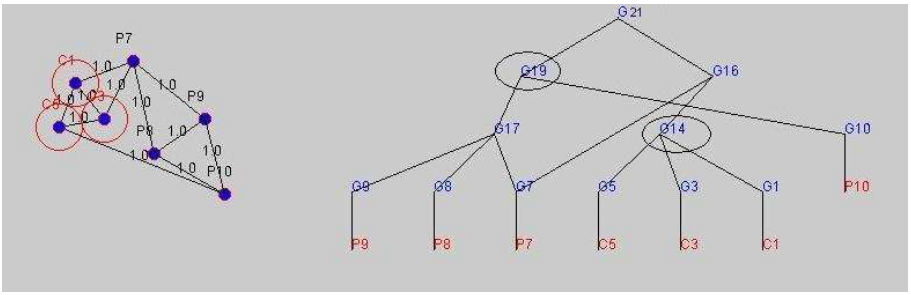

Fig. 2. 2D example, constraint graph, DR-plan.

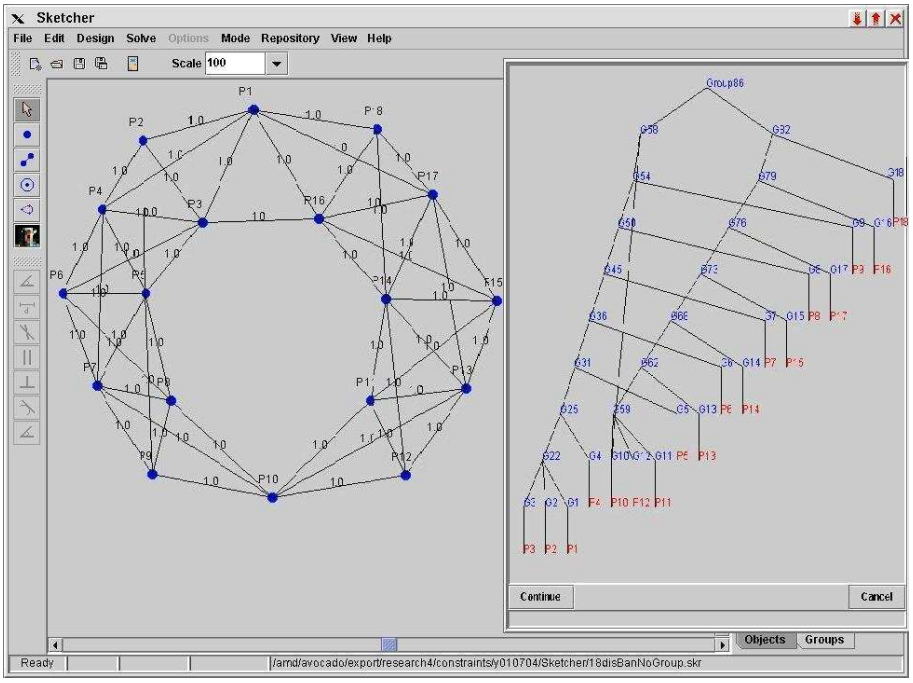

Fig. 3. 3D constraint system (with "bananas" type generic constraint dependence not detectable by a simple dof count), sketch on a 2D canvas; it is well-constrained for the given set of distances; corresponding DR-plan

corresponding constrainedness properties for *generic* parameter values. In many cases, this classification of the subgraphs can be done purely combinatorially, using the dof weights, but ignoring geometric parameters such as distance, angle etc. attached to the constraints.

2.1.1 *Formal definition of DR-plan(ner)s and basic properties.* As mentioned in the introduction, any effective constraint solver combinatorially develops a plan for recursively decomposing the constraint system into small subsystems, whose solutions - obtained from the algebraic/numeric solver - can be recursively recombined by solving other small subsystems. Such a recombination is straightforward, *provided all the subsystems generically have a finite number of solutions, i.e., they are generically rigid.*

The DR-planner is typically a graph algorithm that outputs a *decomposition-recombination plan (DR-*

plan) of the constraint graph. In the process of combinatorially constructing the DR-plan in a bottom up manner, at stage i, it locates a well-constrained subgraph or cluster S_i in the current constraint graph G_i, and uses an abstract *simplification* of S_i to to create a transformed constraint graph G_{i+1}. While we rely on a rough correspondence between well-constrained subgraphs or clusters and *rigid* subsystems, the exact nature and limitations of this correspondence is a complex issue, especially in 3D, and is discussed later (in Section 2.1.2 under desired properties of DR-planners for navigation).

Formally, a DR-plan of a constraint graph G is a directed acyclic graph (DAG) whose nodes represent rigid clusters in G, and edges represent containment. The leaves or sinks of the DAG are all the vertices (primitive clusters) of G. The roots or sources are *all* the maximal clusters of G. For well or well-overconstrained graphs, the DR-plans have a single source, and for underconstrained graphs, the DR-plans have multiple sources. There could be many DR-plans for G. See Figures 2, 3, Generally, for overconstrained clusters, a DR-plan is required to also contain information about which sets of overconstraints can be removed while retaining the rigidity of the cluster. These are called *reducible* overconstraints. Reducible overconstraint *directly associated* with a cluster C refer to those that connect primitive elements occurring in different child clusters of C. Overconstraints that lie within any child cluster C_i are associated with that C_i, not with C.

2.1.2 *DR-plan(ner) requirements.* In this section, we describe essential and desirable properties that a good DR-plan(ner) should have for it to be useful in conceptual navigation of the solution space of general, cyclic 2D or 3D systems. Furthermore, we discuss to what extent these requirements are met by existing DR-planners, in order to justify their role as companion methods for the ESM method presented here.

Generality

It is *desirable* to be able to apply the ESM method to a large class of 2D and 3D constraint systems. For the method to work for any such class, it is *essential* that the DR-plan's nodes should correspond to rigid clusters or generically rigid subsystems, and all choices of reducible constraints of overconstrained clusters should be available.

We now discuss to what extent this requirement is met by current DR-planners. Recall that these clusters and their reducible constraints are picked by the DR-planner using just the constraint graph, i.e., using purely combinatorial properties of the constraint system.

However, to date, there is no known, tractable, characterization of generic rigidity of distance constraint (sub)systems for 3 or higher dimensions, based purely on combinatorial properties of the constraint graph [Whiteley 1997], [Graver et al. 1993], although several conjectures exist. Moreover, there are no known combinatorial characterizations of 2D rigidity, when other constraints besides distances are involved.

It should be noted that in 2D, Laman's theorem [Laman 1970], gives such a characterization if all geometric objects are points and all constraints are distances. However, the standard generalization of the Laman property for 3D, also called *dof-rigidity* is inadequate: while all rigid systems are dof-rigid, the converse is not the case in 3D. Standard counterexamples are systems that contain constraint dependences or inexplicit overconstraints hidden in so-called "bananas" or "hinges" [Graver et al. 1993; Crapo 1979; 1982]. See Figure 3. In fact, these are the only known types of counterexamples.

The *module-rigid* Frontier vertex algorithm described in [Sitharam and Zhou 2004b] and implemented in [Sitharam 2004b] is the first known polynomial time DR-planner that is not fooled by any known type of constraint dependence, specifically, the "bananas" or "hinge" type constraint dependences. It gives a DR-plan whose nodes are so-called module-rigid clusters. It is an open question whether there are any nonrigid subgraphs that are module-rigid. No counterexamples are known.

The module-rigid Frontier vertex algorithm is crucially based on the dof-rigid Frontier vertex algorithm analyzed in [Lomonosov 2004], [Lomonosov and Sitharam 2004], [Sitharam 2004c]. While the latter does not detect bananas and other implicit constraint dependences, it is the first algorithm that gives a *complete* DR-plan, i.e., a *complete decomposition* of the entire graph and each cluster into *maximal* proper subclusters,

in the presence of explicit overconstraints. A generalized version of this decomposition is the key starting point for *detecting* module-rigidity [Sitharam and Zhou 2004b] and is also crucial for obtaining a tractable DR-plan that aids efficient solving (discussed next).

The basic idea for the dof-rigid Frontier vertex DR-planners (without a complete formal analysis and without the fully general completeness property) was presented in [Hoffmann et al. 2001b]. The method for obtaining all possible sets of reducible constraints for overconstrained clusters of dof-rigid Frontier vertex DR-plans, both for 2D and 3D, and modifying the DR-plan once they have been removed, is presented in [Hoffmann et al. 2004].

Tractability and Completeness

The *size* of a cluster in a DR-plan is its fan-in (it represents the size of the corresponding subsystem, once its children are solved). Since the algebraic-numeric solvers take time exponential in the size of the subsystems they solve, and the number of solutions is also typically exponential, minimizing the size of a DR-plan is *essential* to the ESM method presented here. An *optimal* DR-plan is one that minimizes the maximum fan-in. It is shown in [Lomonosov 2004], [Lomonosov and Sitharam 2004], that the problem of finding the optimal DR-plan of even a 2D distance constraint graph is NP-hard, and approximability results are shown only in special cases. Nonapproximability results are not known.

To get around this difficulty, we use the following alternative property. A *tractable* DR-plan for systematic navigation should ensure that each cluster C should be accompanied by a a small set of its children C_i that form an *optimal covering set* of maximal clusters properly contained in C. A *covering set* of clusters is one whose union contains all geometric elements within C. The size of the optimal covering set for C is the size of the cluster C, i.e., its fan-in in the DR-plan. For scalability of the ESM method, we restrict ourselves to constraint graphs for which this size is typically constant independent of the original graph size. The *optimality* here refers not only to the size, but also the algebraic complexity of the active constraint system for solving C, given the solutions of its child clusters. This optimization is the function of the combinatorial cluster-system optimizer (CCO) method (see Figure 1) described later in this section. Note that in order to choose the optimal covering set of child clusters for a cluster C, the CCO needs as input a generalized complete decomposition of C into maximal proper subclusters, which was already seen to be an essential requirement for the generality of a DR-planner, now seen to be essential for tractability as well.

Another crucial property of a DR-plan is its *width* i.e, *number* of clusters in the DR-plan. It is *essential* that this be small, preferably linear, certainly polynomial, in the size of G: this reflects the complexity of the planning process and also affects the complexity of the solving process that is based on the DR-plan. The module-rigid and dof-rigid Frontier vertex DR-planners analyzed in [Sitharam and Zhou 2004b], [Lomonosov 2004], [Lomonosov and Sitharam 2004], [Sitharam 2004c], and implemented in [Sitharam 2004b], are the only ones that have quadratic (typically linear) width and output complete maximal decompositions of each cluster, whose optimal covering sets have typically constant size.

Incorporating a conceptual feature hierarchy or design decomposition

For the user to effectively navigate the solution space just by inspecting the solutions to subsystems in the DR-plan, it is *desirable* that these subsystems include those (well or well-overconstrained clusters) that occur in an underlying conceptual decomposition. This is a feature, part or subassembly hierarchy that captures design intent, i.e., a partial order, typically also represented as a directed acyclic graph.

This incorporation of such an input decomposition is crucial also in order to allow independent and local manipulation of features, parts, subassemblies or subsystems within their local coordinate systems; for allowing the user to dictate the order of resolution (and solution space inspection) of the features, parts, subassemblies or subsystems. For example, parametric constraint solving can in fact be achieved as a special case, where the order is a complete, total order. In addition such a *solving priority order* arises naturally in the case of module-rigid DR-plans [Sitharam and Zhou 2004b] discussed in Section 2. These include clusters

whose clusterhood depends on first solving other (nondescendant) clusters in the DR-plan.

2.2 Combinatorial Cluster-System Optimizer (CCO)

This method is described in detail in [Sitharam et al. 2004]. Here we give its requirements and output. The CCO method takes its input from the DR-plan (see Figure 1) one cluster C at a time, along with a complete maximal decomposition into subclusters C_i, and complete information on reducible overconstraints directly associated with each C_i and with C. (See definitions relating to the DR-planner). The CCO method's output consists of:
(i) a covering set of clusters among the C_i's that will be used for solving C, (ii) a subset of the *cluster overlap* constraints for the chosen covering set which together with a well-constrained set of original constraints, result in a system of algebraic equations (a) that is stable or independent and (b) whose complexity - i.e., number of variables and degree - has been *combinatorially* minimized; typically this complexity is constant, independent of the size of the constraint graph. Note that the actual algebraic system for solving C is never generated or manipulated by the CCO method. All the output requirements are *essential* for tractability of solution and thus for the ESM method presented here, and are proven in [Sitharam et al. 2004], except for the Requirement (ii)(a) which is proven in [Sitharam and Zhou 2005].

2.3 Algebraic Solver

This could be a purely algebraic or numeric solver. An *essential* property of the Algebraic Solver (see Figure 1) is that it should be reasonably efficient in practice; numerically stable; output all real solutions to the input polynomial system; uses interval arithmetic, i.e., can deal with interval values for the coefficients of the polynomials; and can search for solutions within specified intervals for each variable. *Desirable* properties depend on types of navigation constraints that we permit: incorporating overconstraints during solving to prune solution space; dealing with semi-algebraic sets, i.e., dealing with polynomial inequalities during solving. The solver of [Gaukel 2003] satisfies all of these requirements.

2.4 Graphical User Interface (GUI)

Interaction with the user through the GUI is central to the ESM method, and benefits from a well-designed GUI. The GUI requirements however are easy to describe: adequate text and menu interaction as well as 2D and 3D canvases to enable the following. The GUI is used to input: the constraint system and a conceptual decomposition or feature hierarchy, (these could have been partially solved and stored from an earlier session), any updates to these, and interactive user input during the constraint parsing and equation building stages and realization space navigation. The GUI is also used to modify the object, constraint and feature repertoire or constraint-to-equation parse tree. The GUI is used to output: the DR-plan of the input constraint system consistent with the input conceptual decomposition, the system of equations as they are being parsed in stages from the constraints, the subsystem solutions corresponding to the nodes of the DR-plan, a realization of the partially solved system as the navigation proceeds, and the final realization. A good example is the GUI implemented in [Sitharam 2004b], [Kohareswaran 2003] and shown in the figures in Section 3 and 4.

3. NAVIGATING THE SOLUTION SPACE: THE ESM METHOD AND DATASTRUCTURES

This section describes the overall *Equation and Solution Manager (ESM)* navigation method and the common, augmented DR-plan datastructure - used by this method and its companion methods - that aids efficiency. The ESM takes as input the constraint graph, the DR-plan output of *any* DR-planner as in Section 2; a stable and combinatorially optimized system of constraints for each cluster in the DR-plan - i.e., the output of *any* combinatorial cluster-system optimizer as in Section 2; and finally navigation constraints,

such as redundant or consistent overconstraints and chiralities. It generates an algebraic system of equations and inequalities for each cluster. Then, using its access to *any* algebraic-numeric solver as in Section 2, it offers the user a tractable, interactive visual walk-through of the solution space by outputting to the GUI - and storing in the augmented DR-plan datastructure - the partial realizations or solutions of the cluster subsystems in an efficient manner. Interactive input to the method includes the user's choice of solutions to the clusters and additional navigation constraints added on by the user, using which the ESM method recursively assembles the desired solution of the entire system.

The method provides fully flexible backtracking for redoing the user's choices starting from *any* cluster. Finally, the method can be run fully automatically, without any user intervention, to output a realization of the entire constraint system, including navigation constraints that are input apriori.

3.1 ESM Pseudocode

The Solveforest routine can be merged into the Solvecluster routine. It is not necessary, except for clarity, and for dealing with the case where the DR-plan has several roots - in case the input constraint system was underconstrained.

Solveforest (set S of clusters)

```
      [1) At top level of recursion S represents
      a complete decomposition of given system into
      maximal, well-overconstrained subsystems.
      2) At lower levels of recursion, S represents
      an optimal covering set output by CCO]

   For each cluster C in S,
   Solvecluster(C)

   Return user-chosen solution U(C) for each cluster C in S.
      [efficiently stored in an augmented DR-plan datastructure,
       described below]
```

Solvecluster (C)

```
      [The input is CCO's output for C:
       1) at bottom level of recursion, a well-constrained system involving
       primitive geometric elements;
       2) at higher levels, an
       optimal covering set S of solved child clusters Ci, and a stable,
       independent set of constraints between the Ci's, partitioned into
       tree overlap constraints and  non tree constraints]
```

```
Solveforest (set S of Ci's)

Repeat
    CEP(C)

        [Gets from user additional navigation constraints for C.
         1) In some iterations, especially the first,
         it creates algebraic system Alg(C)
         for solving C, using the user-chosen solution for each Ci and
         the non-tree constraints in the stable set of constraints
         between the Ci's, output by the CCO.
         2) In some later iterations, it prunes any already existing set Sol(C)
         using the most recently input navigation constraints.
         3) It Interacts with user further for massaging
         this algebraic system to reduce complexity]

    Algebraic-Numeric Solver(Alg(C))

        [Returns solution set Sol(C) for C: rotation
         of each Ci w.r.t. its parent in the
         tree of overlap constraints given by CCO for C.
         The coordinate system of C is the same as
         the home or root cluster of the tree
         of cluster overlap constraints  output by the CCO:
         the home cluster's
         rotation/translation is fixed to be
         the identity.]

    Communicate the set Sol(C) visually to user via GUI

Until user is satisfied and picks a solution U(C)

Return user-chosen solution  U(C)
```

This visual walk-through is an interactive process, permitting the usual 3D visual tools such as panning, zooming, rotation to visualize the various solution possibilities, which cannot be fully illustrated by the figures here. The interested reader is encouraged to download and run the (opensource) software [Sitharam 2004b]. See Figures 4, 5, 7, 8, 9, for an illustration.

3.2 Augmented DR-plan Datastructure for Efficient Navigation

A cluster object represents a DR-planner's simplification [Hoffmann et al. 2001b; Lomonosov and Sitharam 2004; Oung et al. 2001; Sitharam 2004b; 2004c; 2004a] of a well-constrained or a well-overconstrained subgraph, as well as its original subgraph.

Figure 6 shows the contents of the cluster.

1. *Hierarchy field.* These cluster objects are hierarchical and contain pointers to their sub-DR-plan, a directed acyclic graph (DAG), i.e, the DAG interrelationships are stored within each cluster C as a list of its immediate children C_i.

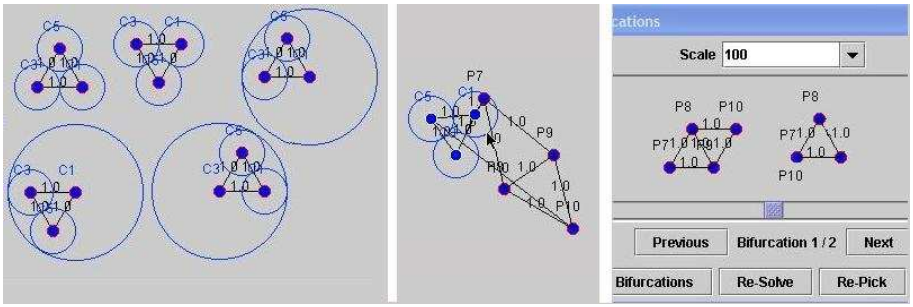

Fig. 4. Left: display of all solutions of the cluster formed by the 3 circles in constraint system of Figure 2. Middle: partially solved sketch after choosing one of them; Right 2 solutions of the diamond of points and distances

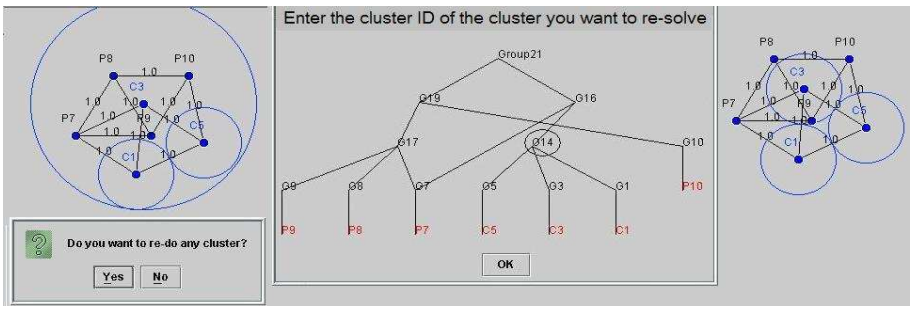

Fig. 5. Left: complete solution of root cluster of Figure 2 after choosing one of the solutions for each of the child clusters (the 3 circles and the diamond of Figure 4). Right: new solution of root cluster after backtracking to repick a different solution of the 3 circle cluster

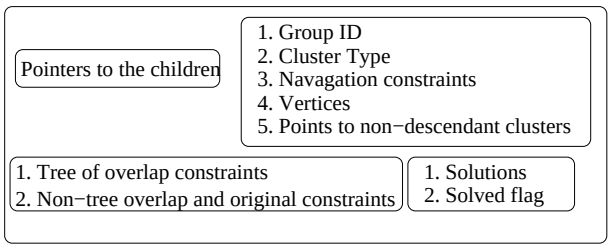

Fig. 6. The structure of the cluster.

2. *Combinatorial descriptive fields.* The cluster C contains more descriptive fields, 1) a group ID corresponding to a feature, if any, that corresponds to this cluster C in the input feature hierarchy. Note that not every feature in the input feature hierarchy is a cluster; however, every cluster feature must appear in the output DR-plan, if the DR-planner satisfies the desirable feature incorporation property discussed in Section 2. 2) Another important descriptive field is the cluster type that indicates whether this cluster is well-constrained, well-overconstrained (combinatorially), and a list of reducible overconstraints directly associated with C. , i.e., does not lie within any of the child clusters of C (see Section 2). 3) Another field contains other navigation constraints that are directly associated with C (these are discussed later in this section). 4) Another field is a list of original vertices or geometric primitives that constitute this cluster.

5) In the case of module-rigid DR-planners such as [Sitharam and Zhou 2004b], as mentioned in Section 2, some clusters become clusters only after other (non-descendant) clusters have been solved, inducing a solving priority order. Thus a cluster contains pointers to such non-descendant clusters.

3. *Fields giving input/output to CCO.* The cluster C stores the input to and the output of the CCO companion method, i.e., a complete set of maximal rigid clusters within C, of which a subset have been picked by the CCO as the children of C. The cluster structure contains a stable, independent system of cluster overlap and original constraints between the (primitive elements of) children of C. These are partitioned into two parts, 1) a tree of overlap constraints, with a home or root cluster and a set of non-tree overlap and original constraints. 2) The original constraints include all non-reducible constraints in C.

4. *Fields describing algebraic solution.* The cluster object has several fields that store the solved degrees of freedom of the cluster. A list of strings is stored, one for each solution of the cluster returned from the algebraic-numeric solver (given fixed chosen solutions to each of the child clusters). When the user selects a desired solution, that string is parsed into a list of actual degree of freedom values for this cluster. Multiple copies of the list are permitted, one for each parent of the cluster in the DR-plan, if they exist. Note that especially when the DR-plan incorporates an input feature hierarchy or partial decomposition, the DR-plan could be a true DAG with child clusters shared by several parent clusters. The final field is a flag that indicates whether the current Cluster C has been solved or not.

Storing Solutions Efficiently. A crucial feature of the datastructure that maintains clarity and efficiency during navigation and backtracking is that each realization of the cluster C is not stored as positions of each geometric element in the cluster - this would cost linear time and space for each child cluster - but rather it is stored as the homogeneous rotation/translations (a constant number of matrix entries) of C's child clusters that resolves the constraints between them. The constraints are the non-tree portion of the independent set of constraints output by the CCO companion method and the rotations are associated with the overlap tree output by the CCO companion method, expressing a child cluster C_i's rotation with respect to another child cluster C_j of C, namely C_i's parent in the tree of overlap constraints output by the CCO companion method. Only when the position of a primitive geometric element v in a cluster C is needed is it actually computed. This could be needed either by the *constraint equation processor (CEP)* of the ESM method described next, in order to construct a stable system of equations for C's parent in the DR-plan, or to prune the solution set of C, or when a particular realization of C needs to be displayed to the user.

The position of a primitive element v in C is obtained by taking a nested product of rotations. (i) The "outer" product is a product taken over any of the paths that lead from the current cluster to a leaf/sink of the DR-plan that corresponds to that geometric element. This path is not unique if the geometric element v is shared by more than one cluster appearing in the DR-plan, but all these paths provide the same, unique position for the geometric element v: this is ensured by the CCO companion method. (ii) Each "inner" product is associated with a single cluster D along the above path: it is a product of rotation matrices that place D within the coordinate system of its parent E in the DR-plan. I.e., it is the product of rotations along the unique path to D from the root or home cluster - in the tree of overlaps for E generated by the CCO companion method (see Section 2).

Thus the entire DR-Plan object is simply a pointer to the root node of a list of clusters as they are described above. In case the graph is underconstrained, the root node is a dummy node, whose children are the actual sources of the DR-plan, i.e, a complete set of maximal clusters. See Section 2.

3.3 Creating and massaging the algebraic system: interactive and editable constraint-to-equation processor (CEP)

The CEP method is a crucial part of the ESM method and has the following functionality:

—It generates a system of equations and inequalities for solving a cluster C, using (a) the output of the

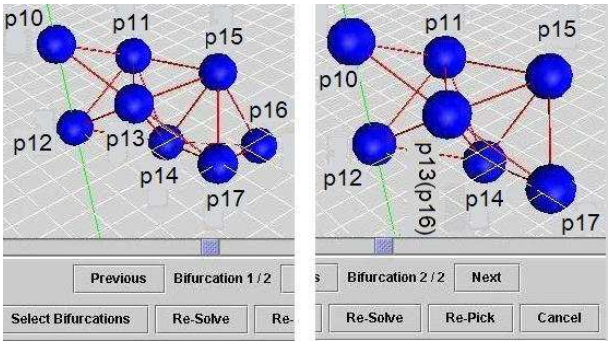

Fig. 7. Navigating by bottom-up traversal of the DR-plan, two 3D realizations of cluster G79, for DR-plan and constraint system in Figure 3.

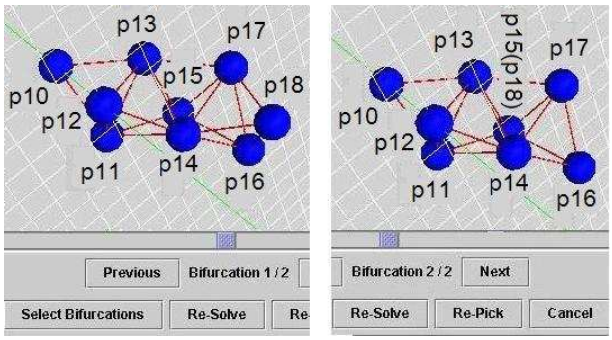

Fig. 8. Navigating by bottom-up traversal of the DR-plan of Figure 3: two realizations of cluster G82, for left solution choice in Figure 7 for child cluster G79

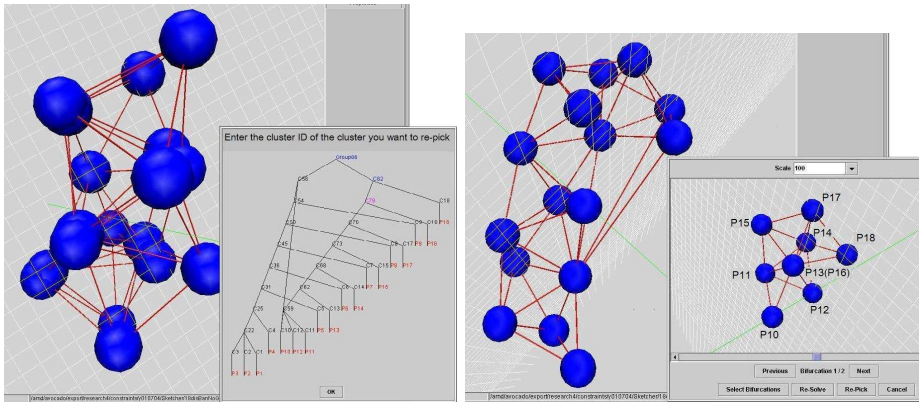

Fig. 9. Left: Complete solution of constraint system in Figure 3 for solution choices in Figures 7, 8 obtained using backtracking. Right: new conformation choice for cluster G82 obtained by picking the other (right) conformation choice for cluster G79 shown in Figure 7; and final solution for new choice

.

CCO companion method; (b) the user-chosen solutions for the child clusters C_i of C; and (c) navigation constraints directly associated with C, including reducible overconstraints which have been isolated by the DR-planner.

—In order to obtain algebraic systems of low complexity, it helps to permit user interaction during the process of building the system corresponding to a cluster. The CEP is additionally "editable" since it permits modifications to the constraint repertoire as well as the constraint parsing mechanism to be *input* by the user systematically and efficiently instead of having to rewrite the code for the ESM. The CEP's user interaction is effected via the GUI text interface, discussed in the communication and integration section.

The CEP has a two stage operation. The first *conversion stage* converts edges or constraints in the *non tree* portion of the output of the CCO into algebraic equation strings understandable by the AlgebraicSolver. These equations are between variables describing the positions of the primitive elements in the child clusters C_i of C. The second *substitution stage* of the CEP expresses the position variables corresponding to a primitive element v in a child cluster C_i in terms of the *variable* rotation and translation of the cluster C_i within C's coordinate system obtained from the *tree* portion of cluster overlaps returned by the CCO.

The CCO's properties described in Section 2 guarantee that the resulting system from the conversion and substitution stages is a stable, independent system of algebraic equations for solving C.

3.3.1 *The Conversion Stage.* This stage of the CEP proceeds in steps, so that at any step, the user can interfere and make substitutions that make the system of equations simpler. The heart of the CEP is a datastructure and look-up system, the constraint-to-equation parse tree (*CEP tree*), which takes the unique characteristics of each edge or constraint and searches an index of user-defined equations strings for a match. The CEP organizes those strings into a n-ary tree representing a natural hierarchy of constraints - this conceptual hierarchy is also mirrored in the representation of constraints used by any standard constraint solver GUI. The use of this tree data structure makes the searching of the constraint list, and conversion of constraint to equation both efficient and more interactive. Moreover, this datastructure can be input or modified by the user, making it fully editable and extensible, as discussed below.

CEP Tree. The CEP tree is organized by equation classes and sub-classes. For instance, consider a distance constraint between two points and a tangency constraint between a two circles. The distance has the equation $(x1-x2)^2+(y1-y2)^2=d^2$, and the tangency has the equation $(x1-y2)^2+(y1-y2)^2=(r1+r2)^2$, where $x1, x2, y1$, and $y2$ are either the locations of the points or the centers of the circles, d is the distance between the points, and r1 and r2 are the radii of the two circles. These equations fall into the same equation class because they are in a sense generated by the same basic template or "grammar." In this example, the tangency equation would be a sub-class of the distance equation since its equation could be created with the substitution of the d variable in the distance equation with the string $(r1+r2)$. This relationship would be represented in the CEP tree by two nodes, one node for each equation form and the tangency node would be a child node of the distance node. To complete the data structure, each node stores (a) a list of edge objects which map to an equation and (b) either an equation or a list of substitutions which create the new child equation from the parent equation. By storing only the substitution information, the process of converting the list of constraints to a list of equations is highly simplified.

When the equation system for a current cluster C is being built one equation at a time, i.e., when single constraints are parsed one at a time, the actual variable names occuring in the output equation are irrelevant during the conversion stage - they will be appropriately modified during the substitution stage described below.

However, the process of parsing a single constraint can be extended to parsing a system of constraints en bloc. This requires more care with preserving identical and distinct variable names that appear in different

constraints. Generating the corresponding equation set involves traversing several paths of the parse tree in lock step, with substitutions being made along the way. The user can interfere at any stage and make substitutions to simplify the entire algebraic system. The user interaction proceeds through a simple GUI text interface discussed in the communication and integration section below.

3.3.2 *The Substitution Stage.* This stage of the CEP re-expresses the position variables - corresponding to primitive element v in a child cluster C_i - which occur in the equation system obtained after the conversion stage. These variables are expressed in terms of (a) the *variable* rotation and translation of the cluster C_i within C's coordinate system obtained from the *tree* portion of cluster overlaps returned by the CCO. and (b) the already solved, *constant* position of v within C_i's local coordinate system. Both (a) and (b) require products of rotation matrices, as in the description of the efficient, augmented DR-plan datastructure. In the case of (b), a nested product is required, with the "outer" product taken over the clusters on any path in the DR-plan from C_i to v. In the case of (a), there is only an "inner product" of the rotation matrices along the path to C_i from the home or root cluster - in the tree of overlap constraints for C returned by the CCO companion method.

3.3.3 *Incorporating Navigation constraints.* Navigation constraints are used not only to prune the search during user directed navigation, but more importantly, to convert the ESM method into a (tractable) automatic search for desired solutions for any cluster of the DR-plan. These constraints can either be given apriori or interactively during a user-directed navigation, to prune unwanted realizations of the DR-plan's clusters encountered thus far.

Leveraging the modularity of the ESM method discussed above - i.e., its separation from the DR-planner, and its applicability to *any* input DR-plan as in Section 2 - permits methods developed along with other DR-planners for incorporating navigation constraints - to be transferred *practically unchanged* to the FA DR-planner and the ESM methods discussed above, for example relational and engineering constraints for triangle decomposable systems given in [Fudos and Hoffmann 1997]. We describe the two other common types of navigation constraints below.

One type of constraint used for navigation or picking out desired solutions for any cluster C of the DR-plan are simply redundant or overconstraints directly associated with C, i.e., connecting primitive elements that lie in different child clusters of C. These are isolated by any DR-planner as in Section 2. There is a well-developed theory of unique realizations and so-called rigidity circuits obtained, for isolating desired solutions from the solution space using redundant constraints [Hendrickson 1992].

The other type of common navigation constraints are chirality [Fudos 1995a; Fudos and Hoffmann 1996b], also called order type or relative orientation constraints. Chirality constraints can be infered from the input sketch. For example, for point objects, these constraints assign signs to the $D+1$ by $D+1$ determinants of the homogeneous coordinate matrices obtained from each set of $D+1$ point objects. I.e, these constraints are $D+1$ degree polynomial inequalities. These correspond to a complete oriented matroid specification [Bjorner et al. 1993]. Alternatively, a *partial* oriented matroid specification could be specified by the user: these assert intersection or separation of pairs of convex hulls of subsets of point objects (called respectively circuits and co-circuits in oriented matroid terminology). For example, two line segments could be constrained to intersect, or a point could be forced to lie in the convex hull of some other set of points. These, too could be written as polynomial inequalities. Chirality constraints, like redundant overconstraints, are also directly associated with a unique cluster C in the DR-plan: the smallest cluster that contains all the primitive elements that participate in the chirality constraint. This also implies that not all of the participating primitive elements fall in any single child cluster of C.

The navigation constraints associated with a cluster C are processed by the CEP using the same Conversion and Substitution stages described above. At the end of these stages, the navigation equations and inequalities

are not in terms of the primitive elements that they involve. Instead, they are in terms of the rotation matrix entries (variables) that place the children C_i of C (in the DR-plan), within C's coordinate system. Thereafter, during each iteration in which the CEP is called for cluster C, (see ESM pseudocode), they are dealt with in one of two ways. If the algebraic system associated with the cluster has not yet been solved (in the first iteration), or the number of solutions is beyond some predefined threshold (in later iterations), and the AlgebraicSolver is capable of handling redundant equations and polynomial inequalities, then these processed navigation equations and inequalities are tacked on to the stable, independent system for C generated as described above, and sent to the AlgebraicSolver. If not, i.e, if either of these conditions does not hold, these navigation equations and inequalities are simply used to prune the solution set for C. Note that since the navigation equations and inequalities have already been processed by the Substitution stage of the CEP, this pruning can be done by directly using the solutions output by the AlgebraicSolver, i.e, using the rotation matrix entry values for the children of C. No computation of the actual positions of the primitive elements in C is required.

3.3.4 *Editability: changing the CEP tree and the constraint repertoire.* The design of the CEP permits the user to flexibly *create* and *modify* the parse tree described above, without hardcoding it.

In creating the parse tree, the first node is a child of a pre-existing dummy node, Root. Newly created nodes can be the children of any previously entered node or Root. After the user has chosen the position of the node within the forming tree, all other necessary information, such as the new label and equation, are obtained. The final step in node creation is to attach a list of the constraint types that will be indexed to this new node. The user input a text list of all the constraint types which map to this node. The user repeats the process of node creation until the entire CEP tree has been entered. The user then saves the entire CEP tree to be used for future parsing.

4. NAVIGATING THE SOLUTION SPACE: ESM'S COMMUNICATION WITH USER AND INTEGRATION WITH COMPANION MODULES

This section describes ESM's integration into a standard constraint solver architecture as in Figure 1 specifically, the dataflow, communication with the companion methods that satisfy the requirements given in Section 2, and a sketch of its implementation in the FRONTIER constraint solver [Sitharam 2004b].

4.1 Dataflow

The ESM must continually interact with the user to select various subsytem solutions and to direct the path towards final solution. All communication between the modules takes place via the UTU or the universal transfer unit, which is also the main driver of the back-end modules. While there are numerous types of information the ESM can pass back to the GUI, their process is always the same. First, the current state of the DR-Plan and all additional communication information is sent to the GUI via the UTU. The GUI reads only the communication information, but stores the residual datastructure information while it processes the communication information. The residual data structure information is passed back to the UTU, if necessary, during the next request, and using it, the UTU can rebuild the current state of the solution process and the ESM can continue where it left off. The detailed actions of the GUI, UTU, and the ESM in each mode of communication during the solving phase are outlined in the next sections.

4.1.1 *Communication during Navigation.* Once the DR-planning phase is over and the DR-plan has been displayed to the user, and when the UTU receives the flag indicating that the user has selected the user-interactive Navigate option (as opposed to the Autosolve option), the UTU simply sends the constraint graph and DR-Plan directly to the ESM. When the ESM receives the continue flag, it begins the solution-navigation process and the visual walk-through or steering through the solution space as explained in Section 3. The

ESM traverses the DR-plan bottom up, as explained above, until it reaches the point where it has several cluster solutions to offer to the user. At this point, the user must select one of several possible solutions for the cluster that the ESM will use to create the final solution for the entire graph. This information must be passed from the ESM back to the Sketcher for display. The ESM sends back two lists of information via the UTU. First, for each solution possibility, the ESM writes out a list of the primitive elements in the cluster. For each element, the ESM writes the object's ID, and a list of its solved position variables. Second, the ESM writes a list of the clusters contained in the current DR-Plan and a boolean flag to indicate whether they have been solved or not. This information allows the user to locate where the solving process is w.r.t. the DR-Plan when making his/her choice of a solution. As a final step, the ESM sets a flag to indicate that the information being sent is a new solution possibility and returns.

The Sketcher displays the solution possibilities to the user, one at a time, in a separate window. The original sketch and a *partially solved sketch* as in Figures 4, 5, incorporating the subsystem solutions chosen so far into the unsolved original sketch are also displayed and allows the user to make a choice of several options described below.

1. Select a solution and continue solution - In this case, the Sketcher simply sets the ESM communication flag to indicate a solution choice and sends the UTU the number of the chosen solution. The ESM uses this to updates the DR-Plan to contain the rotation matrix values for the chosen solution and continues the solution navigation process. Assuming that the user always selects a solution, this process of solving, solution selection, and DR-Plan update would continue until all the subsystems in the DR-Plan have been solved.

2. Choose to backtrack - If the user decides that none of the current solution choices are appropriate, they may chose to edit their earlier selections. For editing, the user has two options: resolve the current cluster with new choices for its children's solutions, or repeat the solving of some previous cluster after making new choices for the solutions of its children. In either case, the Sketcher sends a flag back to the UTU that indicates that backtracking is to occur and an ID of the cluster from which the users wishes to continue solution. The UTU edits the DR-Plan and resets the "solved" flags that indicate that that cluster is the next to be solved.

The ESM will ignore any previous solutions if the fin flag has been unset, and when the UTU calls it with the altered "solved" list, it will continue solving from an earlier point exactly as described above.

3. Choose to update - This final option permits the user to update the input constraint graph and feature hierarchy, because a subsystem is found to have either zero or infinitely many solutions (due to generic or nongeneric constraint dependencies undetected by the DR-planner). The user believes there is something wrong with the input graph itself, rather than the bifurcation choices that have been made earlier. Many modifications of the graph do not require repeating the DR-planning of the system. These changes can be made interactively during solution navigation process and much of the original DR-plan and solutions can be reused. This is one of the advantages of the Frontier vertex based DR-planners [Sitharam 2004c]. The the algorithms, communications and dataflow for such updates are described in [Sitharam 2004a].

Assuming a solution to the current system exists and the user can select subsystem solutions to find it, all communication ends when all the root clusters in the DR-Plan have been solved and the user does not wish to backtrack. At this point, the ESM writes a flag to indicate that a final solution is to be returned. Then it returns a ordered list of all objects, their ID's and the final positions of their degrees of freedom.

4.1.2 *Communication during the Autosolve option.* If the user chose the autosolve as opposed to the navigate option, there is no interaction between the user and the ESM after the DR-Plan has been accepted. The process of auto-solving is the same as the navigate solution option except that when the ESM would request a realization choice from the user, the ESM automatically chooses the first solution that satisfies the

navigation constraints. Backtracking automatically occurs when no solution is found for a cluster C for the current choices of the solutions for the children of C. At the end of the auto-solution process the ESM either returns a final solution exactly as above or it returns a flag that indicates that no valid solution exists.

4.1.3 *Communication between the Constraint-to-Equation Parser (CEP) and the GUI.* The CEP permits the user to both set up the CEP tree, and to interactively build subsystems of equations that the ESM sends to the AlgebraicSolver. Both types of communication take place via the UTU, and through the GUI. The GUI opens a text window for this communication.

The GUI options allow the user to set interactive/automatic mode for the CEP at any point during a session. In the interactive mode of the CEP, the user is again given the option of taking the Entry mode (where the user can edit the parse tree, add new types of constraints etc.,) or the Parse mode (for the navigate option of the solving phase). In the latter case, the user is prompted each time when a subsystem of equations is built by the ESM to send to the AlgebraicSolver. If the autosolve option is chosen, the default choice is to shut off all user interaction with the CEP.

4.2 Implementation within FRONTIER

—The GUI, sometimes just refered to as Sketcher, is written in Java, Java 3D and is the main driver for FRONTIER and is used for all input and output.

—The DR-planner, CCO, UTU, ESM together with CEP, etc., are written in C++. The C++ driver is the UTU, which also coordinates the communication.

—The AlgebraicSolver is off-the-shelf such as [Gaukel 2003] and not discussed here.

Implementing the Communication between the Modules. The DR-plan datastructure is physically shared by the C++ modules, and is communicated, through the UTU, to the Java modules using a Java Native Interface (JNI arrays); however the same datastructure is conceptually mirrored in the object oriented hierarchies used by the GUI's Java modules. The GUI is used for all user input and for all output displays. All information passed between the GUI and the backend modules (C++) is passed through a JNI interface of three arrays, one of integers, one of doubles and one of characters. The GUI writes information into these three arrays, and natively calls the C++ code with these three arrays as parameters. The C++ driver, the UTU, parses these arrays and from them creates (or restores) the augmented DR-Plan data structures described earlier. The UTU then directs the ESM to accomplish the specific request made by the user, described above, in the Dataflow section. The different requests are indicated by the use of an integer flag that is passed as the first integer in the integer array.

5. CONCLUSIONS

We list below the main features of ESM.

- The ESM method *recursively assembles* the desired solution of the system G: it *prunes* the solution space of G by eliminating solutions that are inconsistent with the user's interactive solution choices for well or well-overconstrained subsystems S of G. These are selected by *visually walking the user through* all the solutions - for a current susbsystem S - that are consistent with his/her previous solution choices for subsystems *of* S. These subsystems S are taken from a DR-plan of G. When this DR-plan is in correspondence with a design decomposition of G, the ESM method provides the user conceptually meaningful control.
- The ESM method is typically *fully scalable* in that only a small (constant, independent of $|G|$) number of solution choices are offered to the user at any stage, in a tractable (linear in $|G|$) number of stages.
- The ESM method permits the user to *backtrack* and change his/her previous solution choices at any stage. It also permits additional *navigation constraints* and other inputs by the user that help reduce the solution

choices for the current subsystem S, or help speed up the solving of S. These could be provided apriori or interactively. The CEP permits the user to edit the repertoire of navigation constraint types and the manner by which they are parsed into algebraic equations for efficient solving. Alternatively, the entire ESM method can be switched to *automatic*, i.e., requiring no user interaction. In this case, its efficiency and scalability depend on the tightness of the apriori input navigation constraints.

- Given access to the companion methods in Section 2, the modular ESM method is original, but conceptually simple. It involves significant interaction with the user and with the companion methods. Hence the challenge lies in its clean integration into the geometric constraint solver architecture shown in Figure 1 [Oung et al. 2001], leveraging and naturally extending the data structures and data flow. The original contribution of this manuscript is represented by the dotted balloon in Figure 1. A complete implementation of Figure 1 can be found in the FRONTIER opensource software, [Sitharam 2004b], which is to date the only complete 3D geometric constraint solver available.

REFERENCES

Bettig, B. and Shah, A. 2003. Solution selectors: A user oriented answer to the multiple solution problem in constraint solving. *Journal of Mechanical Design 125,* 3, 445–451.

Bjorner, A., Vergnas, M. L., Sturmfels, B., White, N., and Ziegler, G. 1993. *Oriented Matroids. Encyclopaedia of Mathematics. Vol. 46. Edited by G-C. Rota*. Cambridge University Press.

Bouma, W., Fudos, I., Hoffmann, C. M., Cai, J., and Paige, R. 1995. A geometric constraint solver. *CAD 27*, 487–501.

Bruderlin, B. 1986. Constructing three-dimensional geometric object defined by constraints. In *ACM SIGGRAPH*. Chapel Hill.

Crapo, H. 1979. Structural rigidity. *Structural Topology 1*, 26–45.

Crapo, H. 1982. The tetrahedral-octahedral truss. *Structural Topology 7*, 52–61.

Fudos, I. 1995a. *Constraint solveing for computer aided design*. Ph.D. thesis, Dept. of Computer Sciences, Purdue University.

Fudos, I. 1995b. Geometric constraint solving. Ph.D. thesis, Purdue University, Dept of Computer Science.

Fudos, I. and Hoffmann, C. M. 1996a. Correctness proof of a geometric constraint solver. *Intl. J. of Computational Geometry and Applications 6*, 405–420.

Fudos, I. and Hoffmann, C. M. 1996b. Correctness proof of a geometric constraint solver. *J. Comp. Geometry and Applic. 6*, 405–420.

Fudos, I. and Hoffmann, C. M. 1997. A graph-constructive approach to solving systems of geometric constraints. *ACM Trans on Graphics*, 179–216.

Gaukel, J. 2003. Effiziente losung polynomialer und nichtpolynomialer gleichungssysteme mit hilfe von subdivisionsalgorithmen.

Graver, J. E., Servatius, B., and Servatius, H. 1993. *Combinatorial Rigidity*. Graduate Studies in Math., AMS.

Hendrickson, B. 1992. Conditions for unique graph realizations. *SIAM J. Comput. 21*, 65–84.

Hoffmann, C., Sitharam, M., and Yuan, B. 2004. Making constraint solvers more useable: the overconstraint problem. *to appear in CAD*.

Hoffmann, C. M., Lomonosov, A., and Sitharam, M. 1997. Finding solvable subsets of constraint graphs. In *Springer LNCS 1330*, S. G., Ed. 463–477.

Hoffmann, C. M., Lomonosov, A., and Sitharam, M. 1998a. Geometric constraint decomposition. In *Geometric Constraint Solving*, Bruderlin and R. Ed.s, Eds. Springer-Verlag.

Hoffmann, C. M., Lomonosov, A., and Sitharam, M. 1998b. Geometric constraint decomposition. In *Geometric Constr Solving and Appl*, B. B. and R. D., Eds. 170–195.

Hoffmann, C. M., Lomonosov, A., and Sitharam, M. 1999. Planning geometric constraint decompositions via graph transformations. In *AGTIVE '99 (Graph Transformations with Industrial Relevance), Springer lecture notes, LNCS 1779, eds Nagl, Schurr, Munch*. 309–324.

Hoffmann, C. M., Lomonosov, A., and Sitharam, M. 2001a. Decomposition of geometric constraints systems, part i: performance measures. *Journal of Symbolic Computation 31,* 4.

Hoffmann, C. M., Lomonosov, A., and Sitharam, M. 2001b. Decomposition of geometric constraints systems, part ii: new algorithms. *Journal of Symbolic Computation 31,* 4.

Joan-Arinyo, R., Luzon, M., and Soto, A. 2003. Genetic algorithms for root multiselection in geometric constraint solving. *Computers and Graphics 27,* 1, 51–60.

Kohareswaran, N. 2003. Design of a 3d graphical user interface for frontier, a geometric constraint solver graphs. *Tech.rep., Masters thesis, Univ. of Florida, Gainesville, Dept. of Computer and Information Science, Gainesville, FL, 32611-6120,USA*.

Kramer, G. 1992. *Solving Geometric Constraint Systems*. MIT Press.

Laman, G. 1970. On graphs and rigidity of plane skeletal structures. *J. Engrg. Math. 4*, 331–340.

Lomonosov, A. 2004. Graph and Combinatorial Analysis for Geometric Constraint Graphs. Tech. rep., Ph.D thesis, Univ. of Florida, Gainesville, Dept. of Computer a nd Information Science, Gainesville, FL, 32611-6120, USA.

Lomonosov, A. and Sitharam, M. 2004. Graph algorithms for geometric constraint solving. In *submitted, based on Lomonosov's Univ Florida PhD Thesis, 04*.

Oung, J. J., Sitharam, M., Moro, B., and Arbree, A. 2001. Frontier: fully enabling geometric constraints for feature based design and assembly. In *abstract in Proceedings of the ACM Solid Modeling conference*.

Owen, J. www.d-cubed.co.uk/. In *D-cubed commercial geometric constraint solving software*.

Sitharam, M. 2004a. Frontier, an opensource 3d geometric constraint solver: algorithms and architecture. *monograph, in preparation*.

Sitharam, M. 2004b. Frontier, opensource gnu geometric constraint solver: Version 1 (2001) for general 2d systems; version 2 (2002) for 2d and some 3d systems; version 3 (2003) for general 2d and 3d systems. In *http://www.cise.ufl.edu/∼sitharam, http://www.gnu.org*.

Sitharam, M. 2004c. Graph based geometric constraint solving: problems, progress and directions. In *To appear in AMS-DIMACS volume on Computer Aided Design*, D. Dutta, R. Janardhan, and M. Smid, Eds.

Sitharam, M., Peters, J., and Zhou, Y. 2004. Solving minimal, wellconstrained, 3d geometric constraint systems: combinatorial optimization of algebraic complexity. *Automated deduction in Geometry (ADG) 2004, available upon request*.

Sitharam, M. and Zhou, Y. 2004a. Mixing features and variational constraints in 3d. *accepted to CAD, available upon request*.

Sitharam, M. and Zhou, Y. 2004b. A tractable, approximate, combinatorial 3d rigidity characterization. *Fifth Automated Deduction in Geometry (ADG)*.

Sitharam, M. and Zhou, Y. 2005. Determining an independent set of overlap constraints between rigid bodies. *Manuscript; available upon request*.

Whiteley, W. 1997. Rigidity and scene analysis. In *Handbook of Discrete and Computational Geometry*. CRC Press, 893 –916.