DESIGN AND IMPLEMENTATION OF SKETCHER – USER INTERFACE FOR A GEOMETRIC CONSTRAINT SOLVER

By

NAGANANDHINI KOHARESWARAN

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2002

TO MY PARENTS.

ACKNOWLEDGMENTS

I would like to extend my special thanks to Dr. Meera Sitharam for her continued guidance and support. She has been a great inspiration. I would like to thank Dr. Jorg Peters and Dr. Paul Fishwick for their academic guidance and for serving on my committee. I would like to thank my parents and my sister for their understanding, support and encouragement without which this thesis would have been impossible. I would also like to thank Andrew Lomonosov, Jianjun Oung and Adam Arbree for all their help during the past two years. I would also like to thank my friends Naren, Akhil, Seema and Nikhil for always being there for me. Above all I would like to thank God for never letting me down.

TABLE OF CONTENTS

LIST OF FIGURES

Abstract of Thesis presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

DESIGN AND IMPLEMENTATION OF SKETCHER – USER INTERFACE FOR A GEOMETRIC CONSTRAINT SOLVER

By

Naganandhini Kohareswaran

December 2002

Chair: Dr. Meera Sitharam
Major Department: Computer and Information Science and Engineering

Geometric constraint solvers are used in various applications like CAD/CAM, graphics and visualization, geometric theorem proving, etc. FRONTIER is a geometric constraint solver, which addresses many of the key issues that hamper the utilization of geometric constraints in modeling and visualization systems as well as assembly systems. Sketcher is a versatile user interface for FRONTIER, which enables FRONTIER to express all its capabilities. Sketcher allows the user to tap FRONTIER's potential to the greatest extent.

This thesis presents the object-oriented design of Sketcher and the various advantages such a design offers. The object-oriented nature of the design has made the user interface easily expandable, simple and manageable. This thesis also presents the representation language that is used. This representation language is very general; thus any external system used to call FRONTIER can easily use it. It is common for all modules, thus making communication among the various modules simple and straightforward.

This thesis deals with the various facilities of Sketcher that reflect and capture the facilities of FRONTIER's back end. Sketcher makes the visualization of the constraint systems as easy as possible. Sketcher also incorporates a “Simplesolver” as part of the front end. This solves simple constraint systems for display purposes, without invoking the back end. Sketcher allows the user to intervene and guide the system in the appropriate direction whenever necessary. Sketcher communicates with the other modules of the system just as efficiently.

This thesis is a detailed presentation of both the 2D and the 3D user interfaces. While the 2D user interface deals with constraints in 2 dimensions, the 3D user interface deals with constraints in 3 dimensions. The 3D user interface makes visualization of these 3D constraints easier and more intuitive.

CHAPTER 1
INTRODUCTION

Given a finite set of primitive geometric objects or rigid objects and a finite set of constraints between them, finding valid configurations of the objects that satisfies these constraints is a classical problem – geometric constraint solving. The geometric objects could be points, lines, circles, etc., and the constraints could be incidence, tangency, perpendicularity, parallelism, distance, angle, etc. Such systems are reformulated as algebraic equations whose variables are coordinates of the participating objects [13]. These equations are solved either symbolically or numerically.

Applications of Geometric Constraint Solving

Geometric constraint solvers are most commonly used in CAD type applications [10, 11]. They are used in CAD systems to determine the exact dimensions and the positions of the objects and also the order in which the objects have to be built. Then they can display the various components of the design in proper proportions and realizable configurations depending on the various constraints imposed on them. Thus geometric solvers help the CAD designers to come up with a complete construction program.

Constraint-based drawing applications allow interactive drawing [11]. In 1963 Ivan Sutherland made the first constraint-based interactive computer graphics system – Sketchpad [20]. This system was able to use geometric constraint solving successfully in drawing. Such systems use geometric constraint solvers to continually maintain relationships between the objects drawn, thus making editing easier. Many similar drawing tools have hence been created that use constraint solving for display purposes [2].

Molecular modeling is another field in which geometric constraint solving can be used. The shape and geometry of the molecules are very important. An atom is usually modeled as a spherical ball since the nucleus is at the center of the atom with the electron cloud wrapped around it. There are various forces acting on these atoms that dictate the relative distances of the atoms from each other, and they can be modeled using geometric constraints.

There are many other applications like tolerance analysis, geometric theorem proving, solid modeling [6], architectural design [23], assembly of tensegrity structures [17], etc.

FRONTIER

FRONTIER addresses most of the issues geometric constraints solvers are facing today [8, 9, 15, 16, 18, 19]. FRONTIER is a very efficient constraint solver available for general constraint systems. FRONTIER's strength lies in the degree-of-freedom-graph-based decomposition and recombination method. This algorithm applies equally well to 3D. FRONTIER is a general geometric constraint solver. Flexibility is also one of its important features.

FRONTIER

1. Tells the user whether or not there exists a feasible solution to general geometric constraint systems.
2. Gives a decomposition, and solution pathway for the constraint system.
3. Provides a roadmap of the possible solutions to the user.
4. Allows the users to employ their expertise to guide the system to find their desired solutions.

Sketcher

Sketcher is FRONTIER's front-end. Apart from behaving as an efficient drawing program it has other important duties. Most of the features and facilities of FRONTIER are such that they are relevant only if the user is aware of them and uses them appropriately. It is Sketcher's responsibility to help the user use them. There are a number of potential applications to the system. Hence it is very important for Sketcher to be able to transform quickly to suit any new application that comes along. Sketcher was designed keeping all this is in mind. Sketchers let the user

1. Sketch, visualize (3D) and edit easily.
2. Extend and adapt to new applications easily.
3. Tap the potential and the specific facilities of the underlying solver.
4. Communicate with all the other modules of FRONTIER efficiently.
5. Walk through the solving process and guide the system in the appropriate direction whenever necessary.

Organization of This Thesis

The organization of this thesis is as follows. Chapter 2 of the thesis gives a fairly detailed overview of the FRONTIER system. We discuss all the modules of the system, explaining the function each of them performs. While doing so we also demonstrate by using screen shots the solving of a basic example of a constraint system using FRONTIER. In Chapter 3 we start by exploring all the necessary features that the front-end for a system like FRONTIER should possess. Then we demonstrate using various examples and screenshots how Sketcher meets all the requirements of a highly versatile and efficient front-end for FRONTIER. We describe all the features and facilities that all the three versions of Sketcher – 2D Sketcher, 2D input 3D output Sketcher and 3D

Sketcher have to offer. Then in Chapter 4 we discuss the design of the code and all the other important implementation details of the three versions of Sketcher. We discuss how the design has made Sketcher robust and easily extendible. Though Sketcher can not be compared to other similar systems in all respects, in Chapter 5 we discuss some similar systems and briefly compare them to Sketcher with respect to some aspects. Finally we conclude with Chapter 6 where we discuss the tasks that need to be done as part of the ongoing adaptation of the system. It contains some suggestions that can further improve the usability and versatility of the system.

CHAPTER 2
OVERVIEW OF FRONTIER

The most important issues in geometric constraint solving are generality and efficiency. An efficient geometric constraint solver tries to restrict the usage of the symbolic/numeric solver and tries to keep the subsystems as small as possible. So the geometric constraint solver should create a plan [7, 8, 9, 15] (decomposition-recombination plan) for decomposing the system into small subsystems and then combining the solutions of these subsystems step by step to obtain a final solution. The size of the largest subsystem dictates the overall cost of solving, so an efficient geometric constraint solver tries to minimize this size as much as possible. This is done by repeatedly finding a small solvable subsystem, solving it using the symbolic/numeric solver and replacing the subsystem with an abstraction using the solution obtained, thus simplifying the whole system.

The figure below shows the organization of the various modules in FRONTIER.

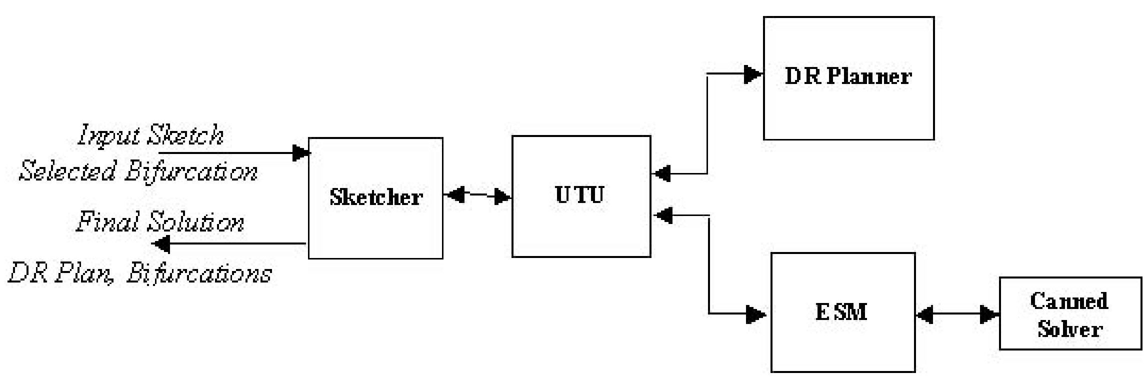


Figure 2-1 Organization of modules in FRONTIER

FRONTIER consists of the following modules: FUI - FRONTIER user interface, UTU - Universal Transfer Unit, DR - Decomposition and Recombination Planner and the ESM - Equation and Solution Manager and a canned solver. The FUI (also called the Sketcher) allows the user to sketch the constraint system on the screen. The UTU receives its data from Sketcher and then it forms the degree of freedom graph representing the system and sends it to the DR Planner. The DR Planner creates the DR Plan, which is passed to the ESM. The ESM interprets the DR tree/DAG/forest and produces equations in form of strings, which are passed on to an off-the-shelf (Canned) solver – MAPLE (presently) to solve. The solutions are returned to the Sketcher, which finally displays the solved output.

Sketcher

While applying a geometric constraint solver in problems like assembly problems it is first modeled using primitive geometric entities like lines, points, circles and arcs. Sketcher is FRONTIER's user interface that lets the user draw a rough sketch of this model on the screen by simply picking the objects from the menu and dropping them on the screen. Then the user can constrain these objects by adding legal constraints between them like incidence, tangency, parallelism etc.

For example the figure below shows a constraint system drawn using Sketcher. It involves four point objects and five distance constraints between them. The distance constraints are fixed to be 1 unit each. In this case the geometric constraint solver needs to solve for the x, y coordinates of all the five points such that all the distance constraints are satisfied. There are many possible solutions for this example. But most of them are translated, rotated and/or scaled versions of a finite number of configurations.

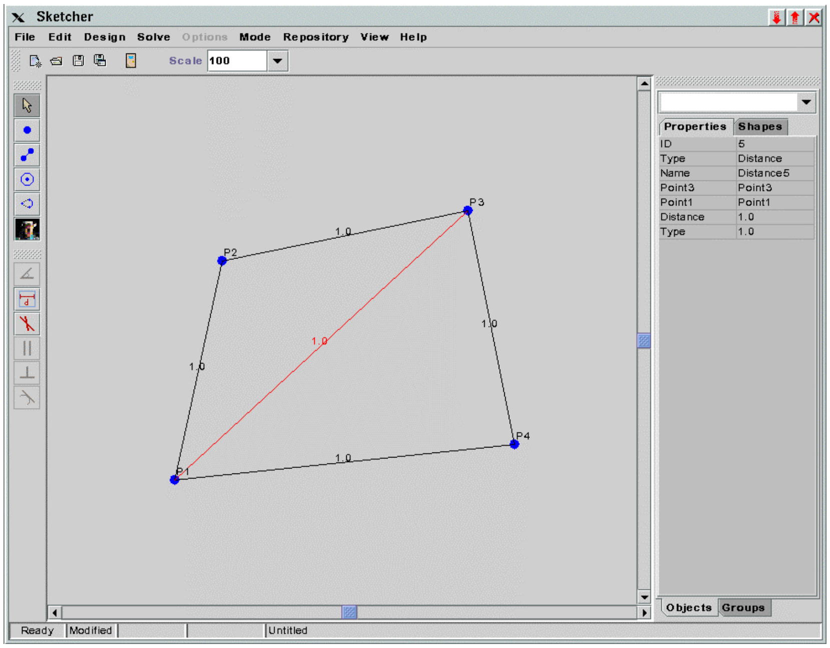


Figure 2-2 Constraint system with four points and five distance constraints.

This sketch is sent to the back-end. While solving this constraint system five equations can be generated one for each constraint for example: $(P1.x – P2.x)^2 + (P1.y – P2.y)^2 = (1.0)^2$, where P.x is the x-coordinate and P.y is the y coordinate of point P. These equations can be solved to find the coordinates of the points such that they satisfy the distance constraints between them. Any well-constrained system in two-dimensions will have three degrees of freedom. Here we have eight variables (two coordinates of each point) or eight degrees of freedom and five equations each removing one degree of freedom. Thus the system is well constrained. The back-end solves the constraint system and returns the final solved output, which is displayed, on the screen.

UTU

The universal translation unit translates the information sent to it by the Sketcher into a DOF- degree of freedom graph. It interprets the options and mode picked by the user and communicates the information to the other modules. Every time the Sketcher and the

ESM or the DR Planner have to have a dialog UTU acts as the interpreter and passes the appropriate data to and fro. UTU is implemented in C++. Java Native Interface is used to facilitate the communication between the Sketcher, written in Java and the other modules written in C++ [15]. Apart from the communication the most important function that the UTU performs is the construction of the DOF. There is no unique solution to this and the UTU tries to build the most generic DOF that is used by the DR-Planner.

DR Planner

The DR Planner is the implementation of the frontier vertex algorithm [15]. The DR Planner receives its input from the UTU. The input is generally a set of DOF graphs representing the input system. Sometimes the user might also decide to provide the solver with initial partial decompositions, which tell the DR Planner about the feature hierarchy in the constraint system. These partial decompositions are incorporated in the DR DAG. During the update mode, the DR Planner is provided with the already created DR DAG, the additional changes to the input constraint system or the partial decompositions, and information about which of the nodes of the DR-DAG have already been solved. The DR Planner creates one or more DR DAG depending on whether the system is well constrained or under constrained. This DAG tells the ESM the partial order in which the input system and its subsystems must be resolved. To let the user see the complete process of solving the Sketcher displays the DR-Plan to the user at each step which tells the user exactly which of the sub-problems (represented as clusters of objects) have already been solved and which one is presently being solved.

The figure below shows the DR Plan that FRONTIER created for the above example. The whole system is represented as Group10.

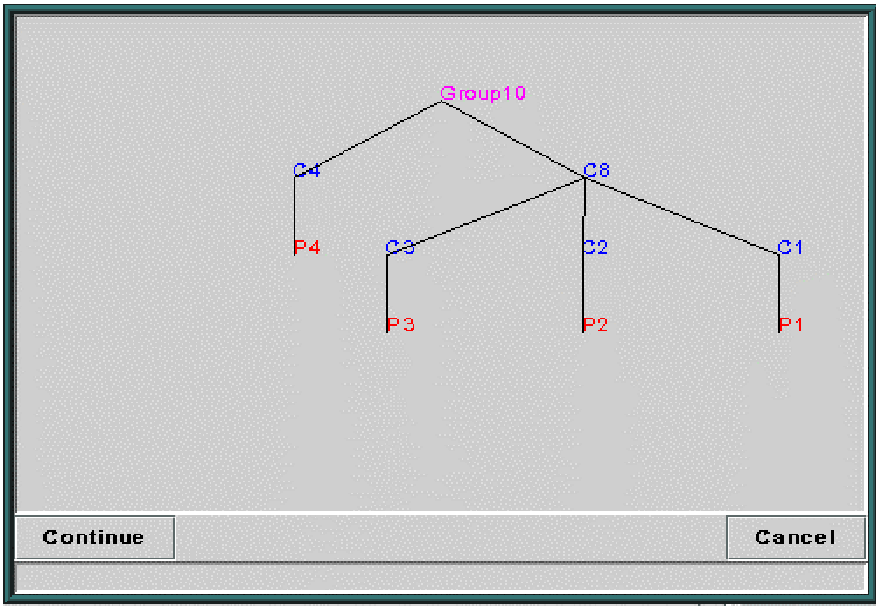


Figure 2-3 DR Plan for the constraint system.

It has decomposed the system with four points to a sub-system with 3 points (P1, P2, P3) represented as C8 (cluster 8) in the figure and a sub-system with the remaining one point (P4) represented as C4 (cluster 4).

ESM

The ESM takes the DR DAG given to it by the DR Planner and generates equations for the tree. It generates equations for each node in the DR DAG. As it walks the tree recombining the sub-systems it keeps assigning values to all the degrees of freedom of the shapes. During the solving process there are situations when there is more than one possible solution for intermediate problems. At this point the ESM either chooses the solution by itself or goes back to the user and lets the user pick the solution, depending on the option that the user specifies. It proceeds by solving for the rotations and translations of all the children clusters (which at the lowest level are usually individual objects) of a cluster thus obtaining the orientation of the children clusters in the parent cluster's coordinates. The same operation is performed at a next level where it has to solve for the rotations and translations of larger clusters instead of individual shapes. Then these rotations and translations are applied to all the children clusters of these larger clusters.

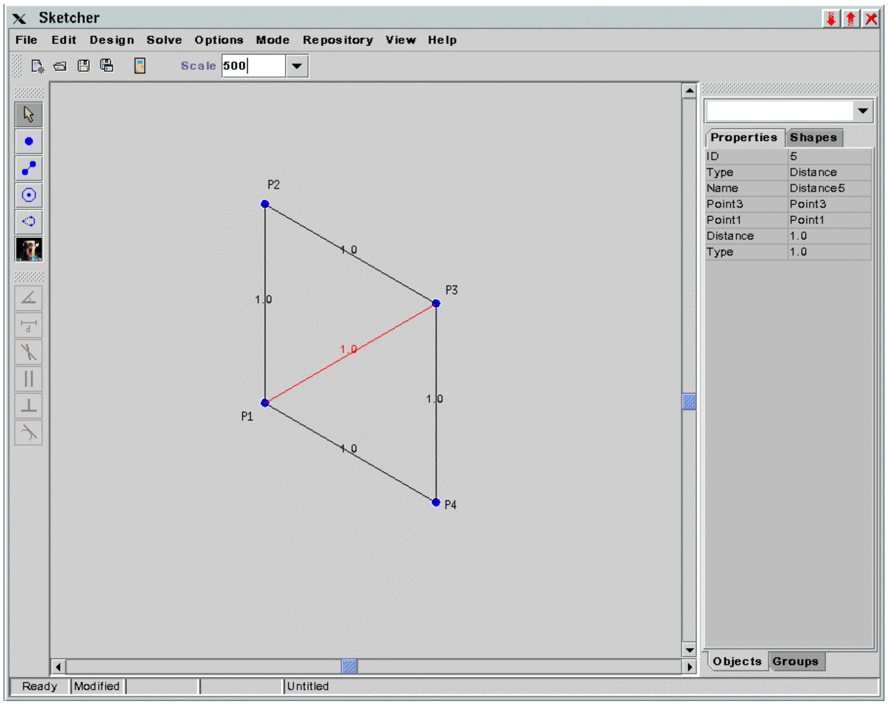


Figure 2-4 Solved constraint system with all the points satisfying all the constraints.

In case of the above example the sub-system containing points P1,P2 and P3 was solved first. Then at the next level the ESM solved for the rotations and translations of the children clusters, which are P4 and the rigid sub-system consisting of P1, P2 and P3. The above figure shows the solution thus obtained. It satisfies all the distance constraints between the points.

CHAPTER 3
ISSUES IN A USER INTERFACE FOR FRONTIER

The user interface of FRONTIER has many important duties in addition to the generic user interface duties. It is solely responsible for demonstrating all the facilities and features of FRONTIER. Only then the user will be able to utilize them. This significantly increases the demands on the user interface. To start with, the sketch should be easily editable. FRONTIER is a very general constraint solving system. Hence there is always scope for new applications and extensions to the existing applications. The user interface is the one that needs to undergo most of the changes. Hence it should be easily extendable and adaptable. As we have discussed already the first thing that is done during the solving process it decomposition of the system into smaller sub-systems. The user should be informed of the hierarchy created. The user should be allowed to direct the process of decomposition if he wishes. The FRONTIER uses the declarative approach of feature modeling. Hence it is faced with the standard problem of classifying and steering through the multiple generic solutions or bifurcations. To address this problem the feedback from the user should be used. Moreover this feedback and the navigation at every stage should be made as flexible as possible. To do this the user should be allowed to update the input sketch, including the input decomposition, at any time. The user interface should be able to detect and support online resolution of systems. The user interface should help the user to recognize and deal with non-generic, special case behavior that is not recognized by the automated procedure. It should allow the user to inspect the process step-by-step. Many a times the user may want to reuse a set of shapes and constraints between them. In that case the user should be allowed to save the frequently used set and retrieve it whenever required.

Sketcher aims to address all this issues. To realize this aim Sketcher performs the following tasks.

DR Plan Display

The DR Plan display helps the user to walk through the solving process. Initially before the solving begins the DR Plan is displayed so the user can choose to stop or continue the solving process. If the user hits cancel, the solving process stops and Sketcher enters the update mode. The figure below shows a 2D constraint system drawn using 2D Sketcher. The figure 3-2 shows the DR Plan that was created for the system in figure 3-1.

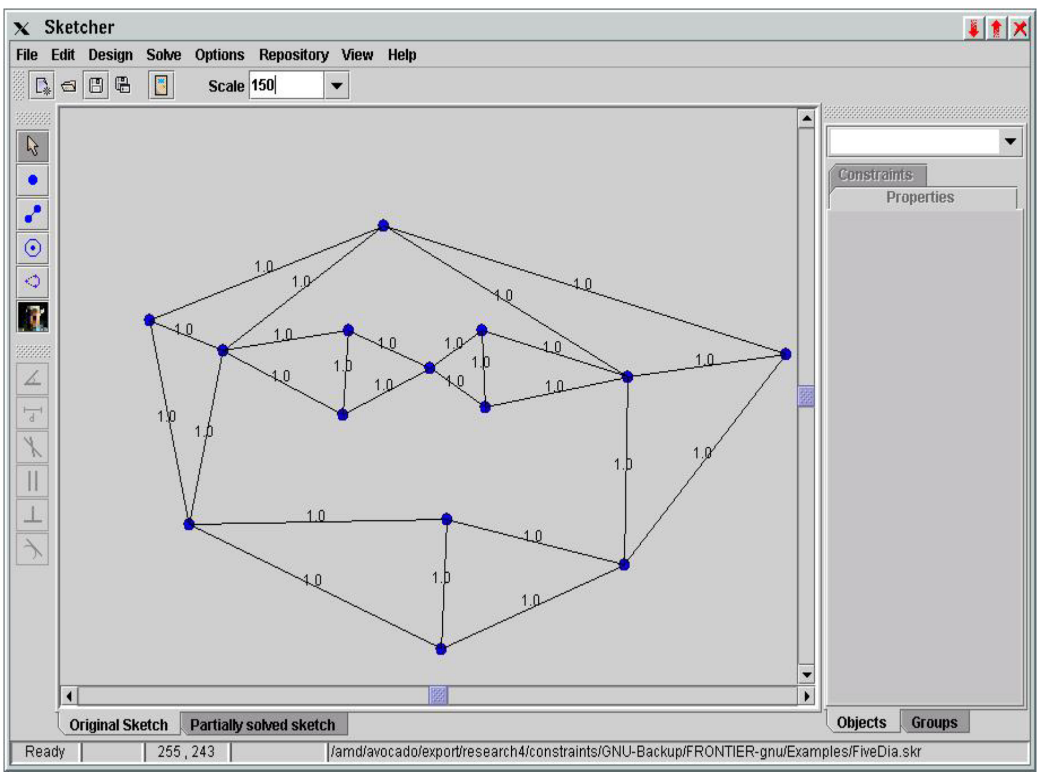


Figure 3-1 Input 2D constraint system containing 14 points and 25 distance constraints.

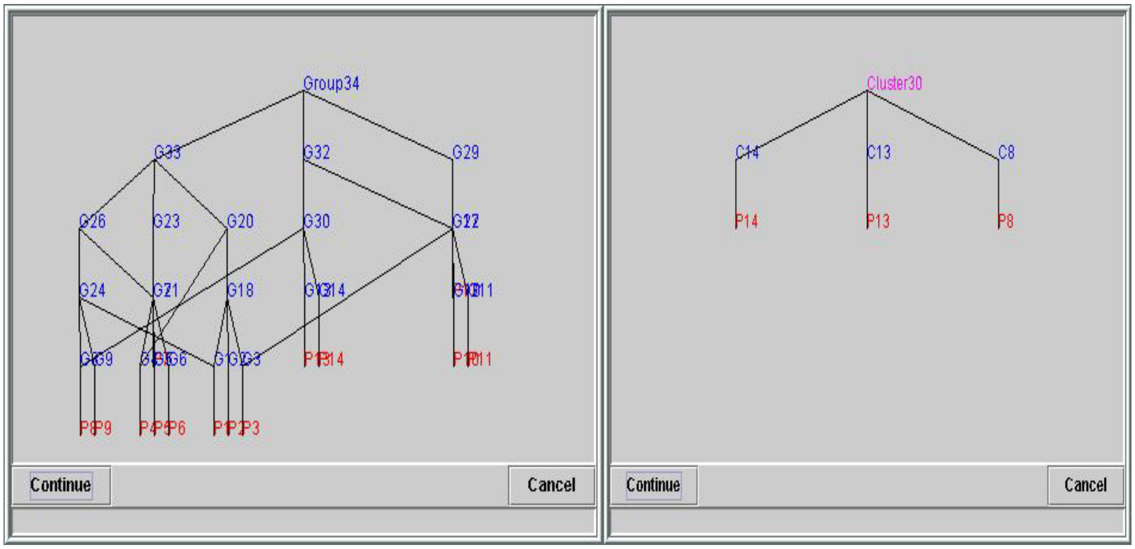


Figure 3-2 The first window shows the DR Plan of the system in figure 3-1 and the second window shows the zoomed view of cluster 30.

The format used to display the DR Plan is most intuitive in that it looks like what the user would draw using a paper and a pencil. But this format can become very difficult to read when we are dealing with many clusters. So Sketcher provides a zoom-in facility to concentrate on any one particular cluster as shown in the second window in figure 3-2. The users can zoom-in on a cluster by clicking on it and then zoom out to click on another cluster. The DR DAG also indicates to the user which of the clusters in the DAG is being solved presently and which of them have already been solved. This is achieved using the "fin" flag. The ESM maintains a fin flag for each cluster, which is set when the cluster is solved. When the ESM sends the bifurcation information to the Sketcher, the fin flags are communicated to the Sketcher along with the cluster ID of the clusters whose bifurcation data has been sent. When the DR DAG is first displayed, Sketcher also notifies the user of the over-constrained clusters' IDs. The display also indicates which clusters in the DR Plan were the results of the partial input-decompositions provided by the user and which partial input-decomposition the cluster corresponds to.

Bifurcation Window

Bifurcations are possible solutions to the sub systems of the system. For example consider a small sub-system having four points and five distances.

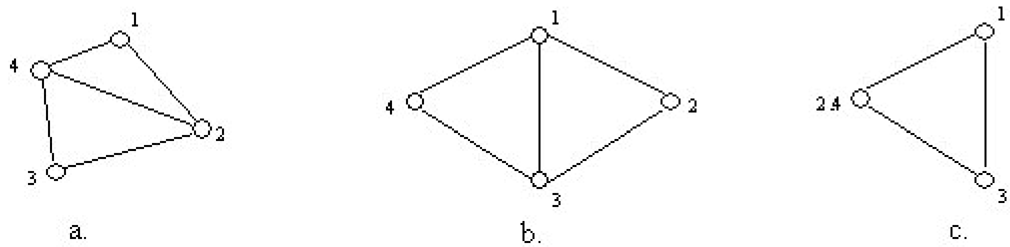


Figure 3-3 Bifurcations of a constraint system. (a) Constraint system, (b) & (c) Two possible solutions to the constraint system in figure (a).

The small circles in the above figure represent the points and the lines between them represent the distance constraints between them. The b and c parts of the figure above show the two possible solutions to this system. The user is allowed to pick one of these configurations or bifurcations. When the ESM obtains the set of solutions for a sub-system it sends them back to the Sketcher. For each bifurcation the ESM sends the Sketcher the coordinates of all the shapes in that cluster, in the coordinate space of that cluster. The shapes are displayed in the bifurcation window. Sketcher can also displays the IDs of all the shapes on the bifurcation window. It also simultaneously highlights the corresponding shapes on the input sketch so that the user can make the right choice of bifurcation depending on the positions of these shapes in the whole system.

In case of the 2D input 3D output Sketcher the input sketch is a 3D constraint system, but it is drawn on a 2D palette. When the bifurcations are displayed the objects involved are displayed on a 3D canvas. Similarly the final sketch is also displayed on a 3D canvas.

For example the figure below shows an input 3D constraint system drawn using the 2D input 3D output Sketcher and the DR plan for the system.

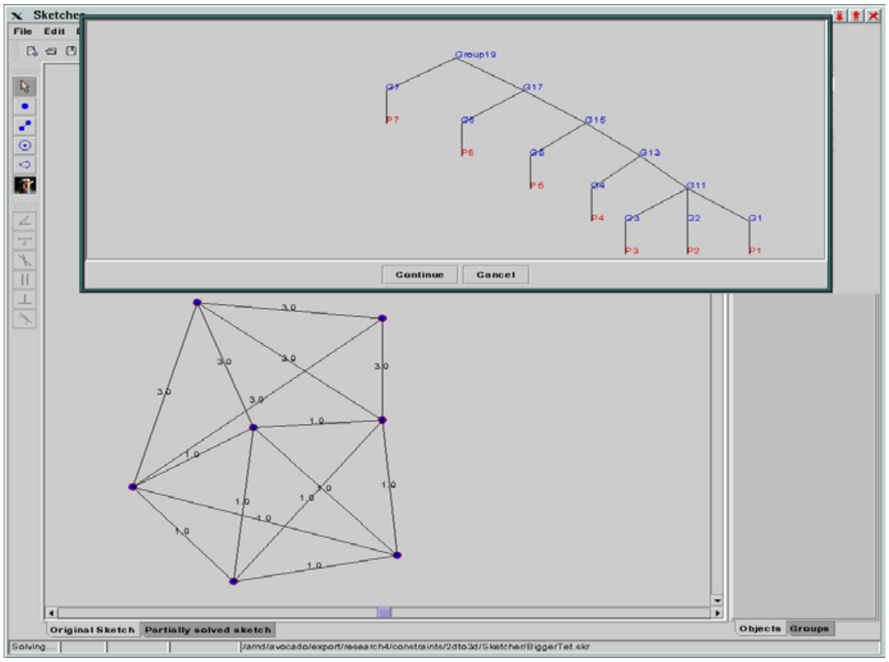


Figure 3-4 A 3D constraint system drawn on the 2D input 3D output Sketcher along with the DR Plan for the system.

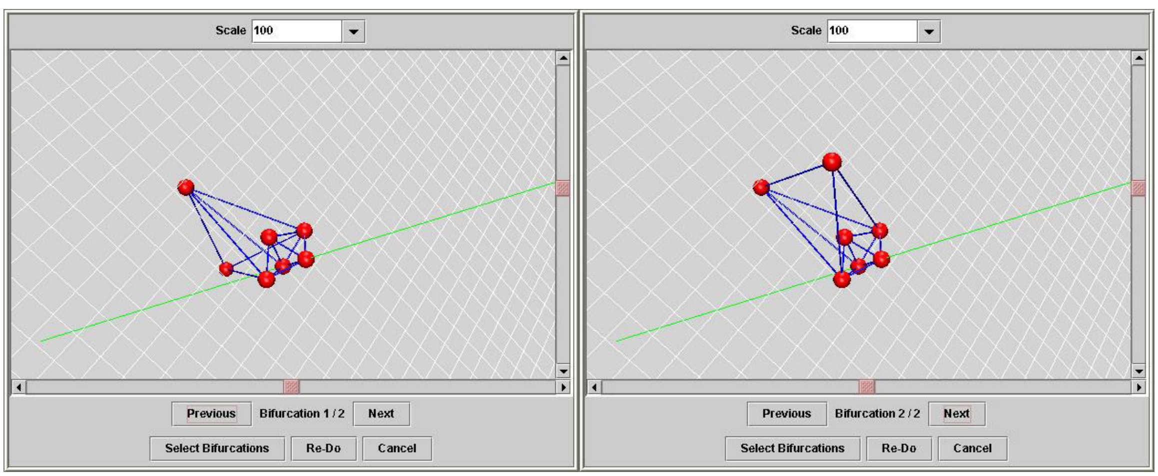


Figure 3-5 The bifurcations of the highest level cluster in the system shown in figure 3-4.

When the system in figure 3-4 is solved then the bifurcations at each cluster are sent to the user for display. The user can choose one of them. Sketcher then sends back the bifurcation number the user picked for each cluster, to the ESM (through the UTU). This information is used by the ESM in the solving process. The figure 3-5 shows both the possible solutions at the highest-level cluster.

Back Tracking

Backtracking is very important. This allows the users to go back and change a bifurcation choice they made at any point of time. The DR Plan is displayed to the user at all times so he can choose the cluster whose bifurcation he wants to change. On the bifurcation window there is a “Re-Do” button which when pressed asks the user which cluster he wants to re-do and the user can simply type the cluster ID from the DR Plan, to re-do it. Then depending on whether Sketcher has already sent the bifurcation choice of the picked cluster to the ESM or not, further operations are performed. If Sketcher has not yet sent the bifurcation choice of the picked cluster to the ESM it means that bifurcation choice has not been used by the ESM in solving yet. All the bifurcation data of such clusters still exist in the data obtained from the ESM. So Sketcher can simply display the bifurcations of the picked cluster for the user. Once the user chooses the new bifurcation Sketcher changes the bifurcation choice for that cluster in its own records that will be sent to the ESM later. But if the picked cluster is a cluster whose former bifurcation choice has been used by the ESM to solve its parent cluster then its parent cluster needs to be re-solved. Also its bifurcation data has been over written in the data structures. In such a case the Sketcher tells the ESM which cluster the user wants to redo and the ESM simply resets the fin flag of that cluster. So the next time it walks up the DAG the first unsolved cluster that it finds would be the picked cluster and it will start solving from there. But the other branches of the DAG that are not affected by the bifurcation of the picked cluster will not be solved because their fin flags do not change.

The figure below shows the input sketch of a 2D constraint system. In this example the constraint system contains 14 points with 25 distance constraints. The huge system is decomposed to obtain the DR Plan shown in the figure 3-7.

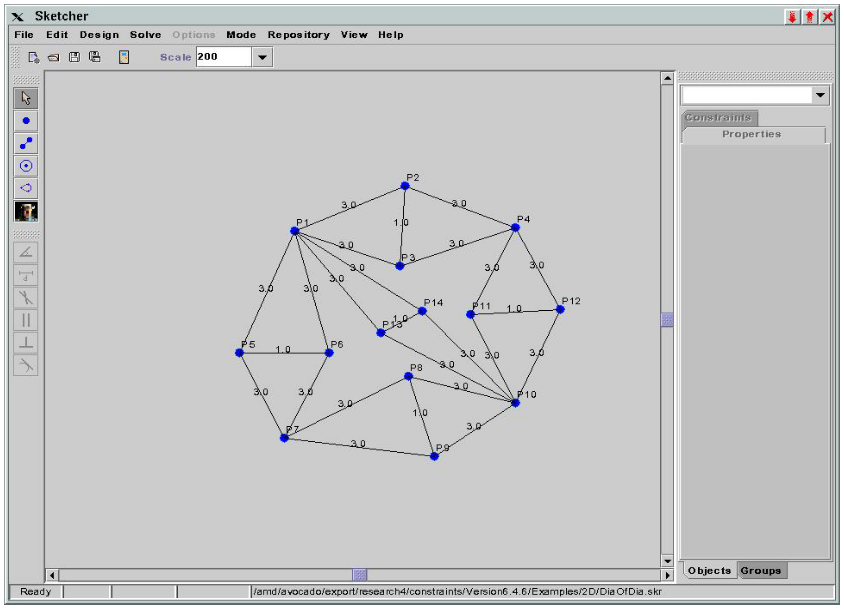


Figure 3-6 Input sketch.

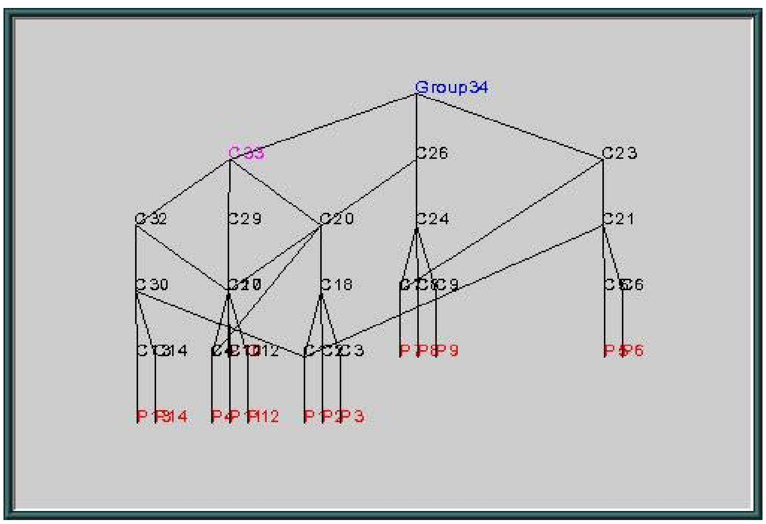


Figure 3-7 DR Plan for constraint system in above figure.

At the intermediate stage shown in the figure 3-7 the cluster number 33 is being solved. At this point the three children clusters (32, 29, 20) of this cluster have to be rotated and translated appropriately. The user is provided with all the possible solutions (bifurcations) of these children clusters using the bifurcation window, so he can pick one of them.

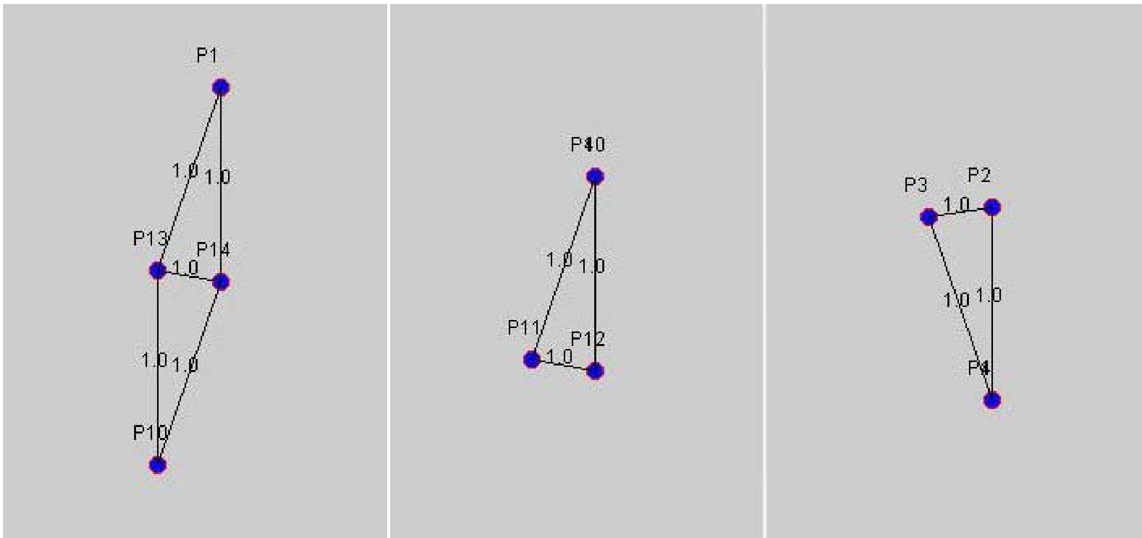


Figure 3-8 Bifurcations picked by the user for the clusters 32, 29 and 20 respectively.

Suppose the three bifurcation choices made by the user initially for the clusters 32, 29 and 20 respectively are as shown in the above figure. These choices are then sent to the solver for solving the parent cluster 33. Even though the solutions for all the three children of the cluster number 33 are correct the cluster 33 cannot be solved because all the constraints at that level cannot be satisfied with this set of bifurcation choices for the children clusters. So the solver informs the Sketcher that it was unable to find any solution with these bifurcation choices.

Figure 3-9 Error generated by the solver.

Sketcher passes this information on to the user so the user now has to pick another bifurcation of any/some of these three clusters. The user may now pick a different bifurcation for any cluster below the one that is being solved. It is not necessary to pick a cluster among the children of the cluster being solved.

Figure 3-10 The cluster picked by the user to re-do.

Suppose that the user picked the cluster number 29 to “re-do”, in other words the user wishes to pick a different bifurcation for the cluster number 29. Then the bifurcation window for the cluster 29 is shown to the user. Depending upon whether the cluster picked to re-do is at the same level as the cluster being solved or not, Sketcher handles the communication with the solver as discussed earlier. Let us say that the user picks the bifurcation shown in the figure below for cluster 29.

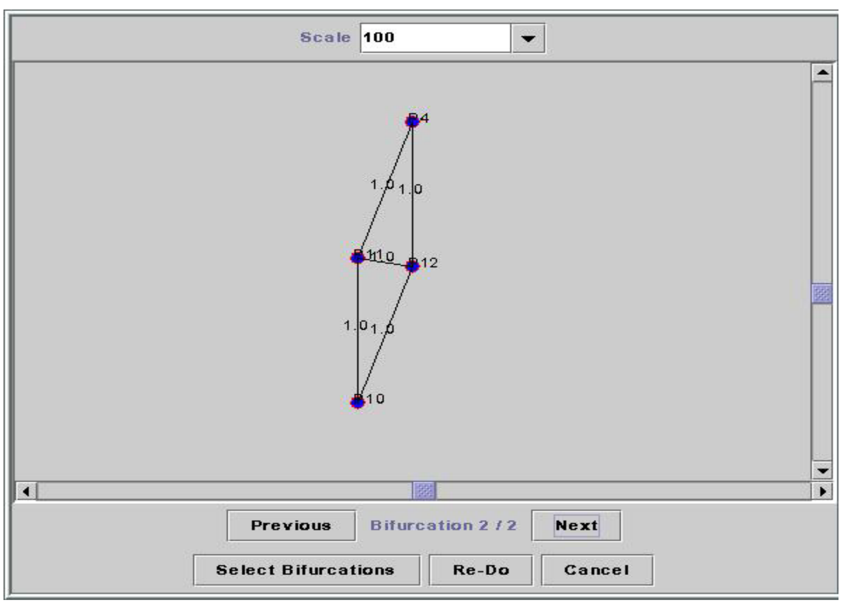


Figure 3-11 The bifurcation picked by the user for cluster 29, for the second time.

Similarly the user may again choose to pick a different bifurcation for cluster 20 also. Let us say the user picks a similar bifurcation for the cluster 20. After this the solving process proceeds as usual and the final solution is obtained. In this case one solution to the system looks like that shown in the figure below.

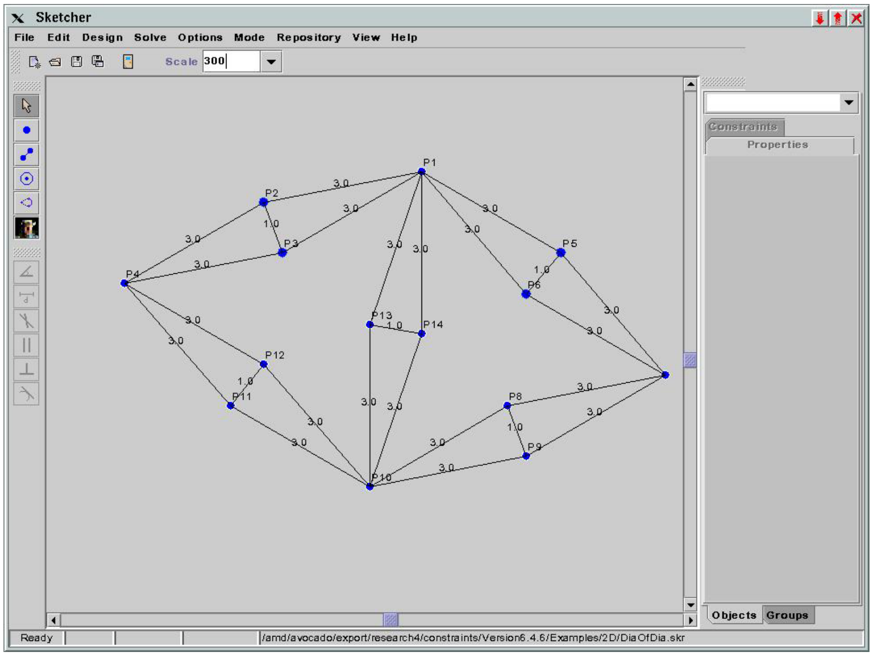


Figure 3-12 Final solution.

Input Partial Decomposition

There are potentially many different decompositions of a system possible but the DR Planner comes up with just one of them. Sketcher allows the user to direct the decomposition process thus allowing the user to pick one of the many decompositions. Sketcher lets the user provide the DR planner with a partial decomposition. The user can group some of the objects in the input sketch to indicate that he wants those objects to form a single cluster in the final DR Plan. In other words the user would like those objects to form one sub-system during the decomposition process. In this way the user can also provide a hierarchy within the sub-systems. These partial decompositions are displayed on the a groups pane on the right hand side of the window. It can be viewed in the tree format also. It is possible that more than one person is working on the same sketch and they might want to have different decompositions. It also allows different users working on the same sketch to input different partial input-decompositions.

The figure 3-13 below shows a constraint system with two different input partial decompositions. The figure 3-14 shows the partial decompositions provided by the two users for the same system.

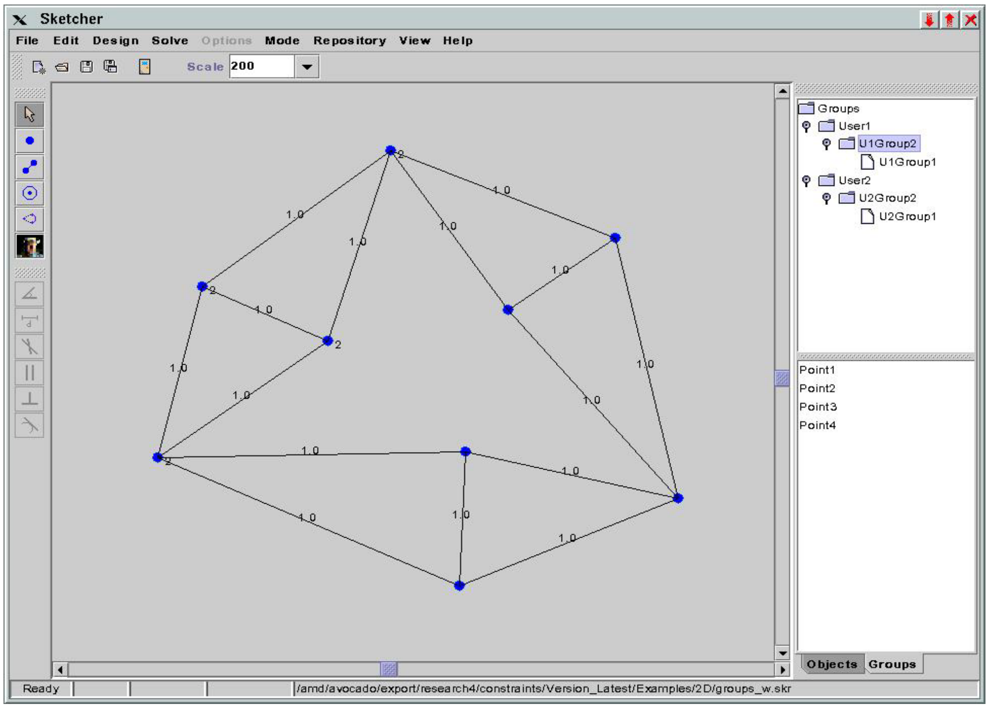


Figure 3-13 Constraint system having two different partial input-decompositions.

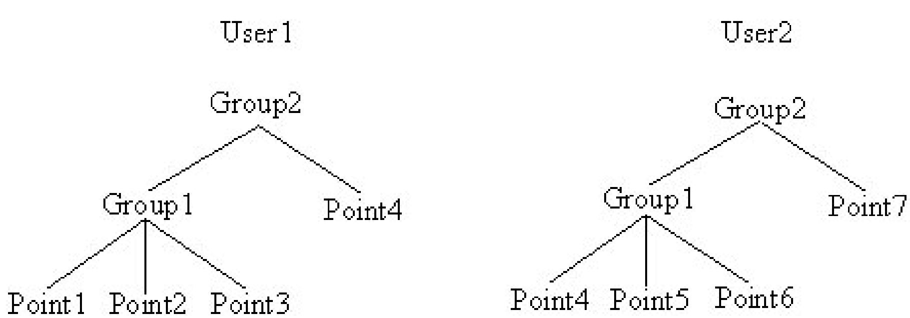


Figure 3-14 Partial input decompositions of the constraint system.

These partial decompositions are part of the decomposition plan that the DR Planner comes up with. In the DR Plan display the clusters that correspond to the input groups are marked by a bold spot.

When the mouse is moved over these clusters the corresponding user and the group names are displayed at the bottom of the window.

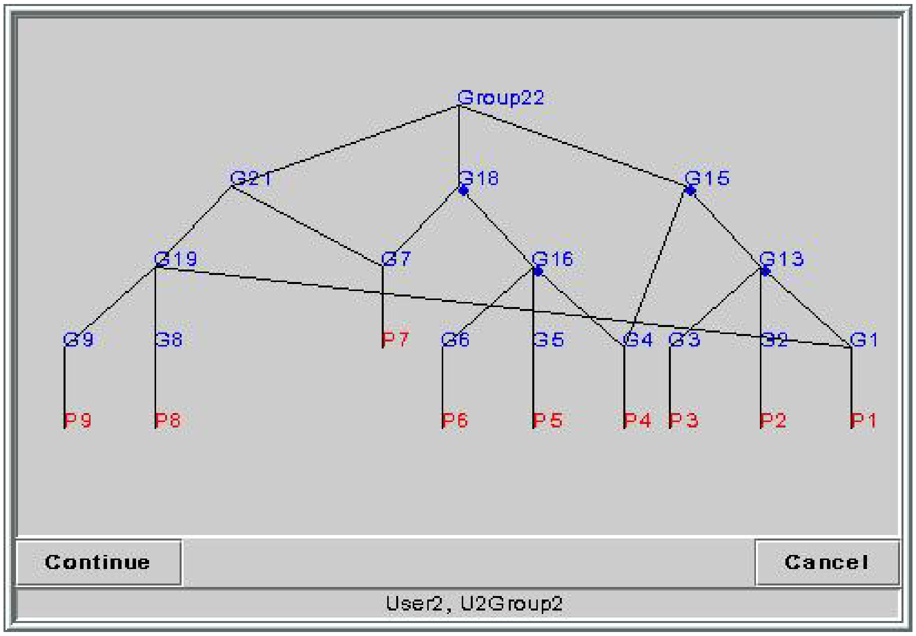


Figure 3-15 The DR Plan with the input groups marked on the corresponding clusters.

The above figure shows the DR Plan, which has incorporated the partial input-decompositions of the two users. This figure is a snap shot of the DR Plan when the mouse was placed on cluster 18. Cluster 18 corresponds to the group U2Group2 which is displayed at the bottom of the window.

Modes of Operation

The Sketcher offers different modes of operation for the convenience of the user.

Generate Mode

In this mode the operation is straightforward. The user draws the sketch of the 2D or the 3D constraint system on the 2D panel and optionally provides input partial decompositions to direct the decomposition process. The user then solves the sketch. In this mode he/she can again choose to use either the the auto-solve or the get-bifurcations mode.

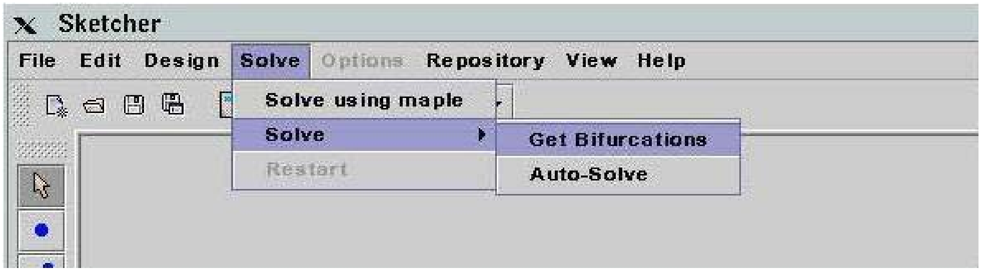


Figure 3-16 Screenshot showing the solving options.

Get-bifurcation

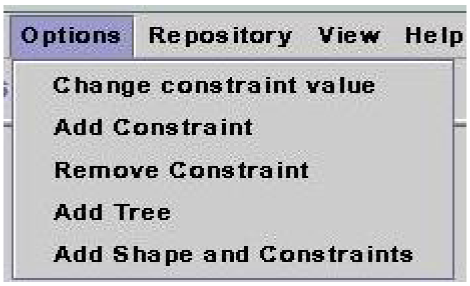
In this mode whenever the ESM finds more than one possible solution to the sub system it stops and sends them back to the Sketcher. Sketcher in turn displays the different options to the user in the bifurcation window and allows the user to pick the bifurcation that he/she thinks correct. This information is sent back to the ESM and the ESM picks the solving processes up from that point.

Auto-solve

In this mode the ESM does the choosing of the bifurcations for the user. The user may choose this mode if he is not sure which intermediate bifurcation he should pick and so wants to just look at the final output and decide or if the bifurcation choice is not an issue at all as long as the final output conforms to all the input constraints.

Update Mode

The user may want to add a constraint, or delete a constraint or make changes in the sketch after looking at the DAG or after walking through some bifurcations. In the update mode the user is allowed to stop the solving process at any point in between and make changes to the sketch and then continue solving. Once the user enters this mode he/she is forced to indicate what change he/she is going to make prior to making those changes. This is done by disabling all the other facilities. This is a check that will make sure that the user does not do something that Sketcher is not expecting him to do. So once the user picks some update option Sketcher allows him to make only those changes to the sketch. Then those changes are sent to the UTU, which are incorporated in the DR Plan. A different code is pre-defined for each of these different updates and the UTU flag is set to that number, telling the UTU how to interpret the further information sent.



3 – Change Constraint
4 – Add Constraint
5 – Remove Constraint
6 – Add Tree
7 – Add Shape and Constraint

Figure 3-17 Screen shot of the various update options and the pre-decided code for each of the options for communication.

The information about what changes have been made in the sketch and in case of change constraint what the constraint value has been changed to, is sent to the UTU so that suitable changes can be made in the DR DAG.

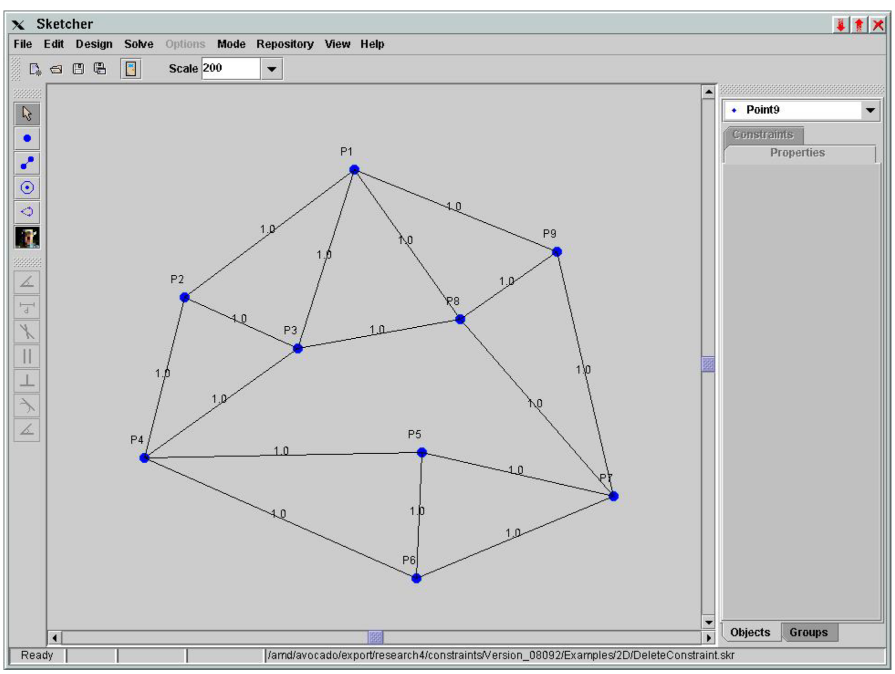


Figure 3-18 2D constraint system drawn using the 2D Sketcher.

For instance let us say that the user tries to solve the 2D constraint system shown in the figure 3-18. The system consists of 9 points and 16 distance constraints. This system is over constrained. This over constraint is detected by the back-end which informs the Sketcher about it when it sends the DR Plan to the Sketcher. Sketcher reports this over constraint to the user. The system not only informs the user of the presence of the over constraint but also tell him/her at which level (cluster) the system is over constrained. The figure 3-19 shows the DR Plan obtained and the information window that informs the user about the over-constraint at group 25.

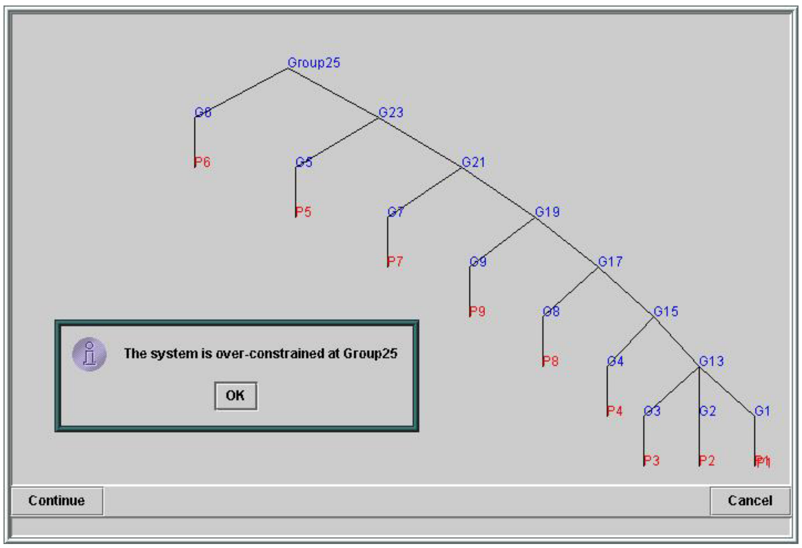


Figure 3-19 DR Plan for the system shown in figure 3-18.

Now the user can hit the cancel button and pick one of the update options. Since the system is over constrained let us say that the user deletes a constraint and then solves the updated sketch.

The figure below shows the updated sketch and the new DR Plan obtained for the updated sketch. We can see that the distance constraint between points P3 and P8 has been deleted and the resultant DR Plan is different. Now on the solving proceeds as usual.

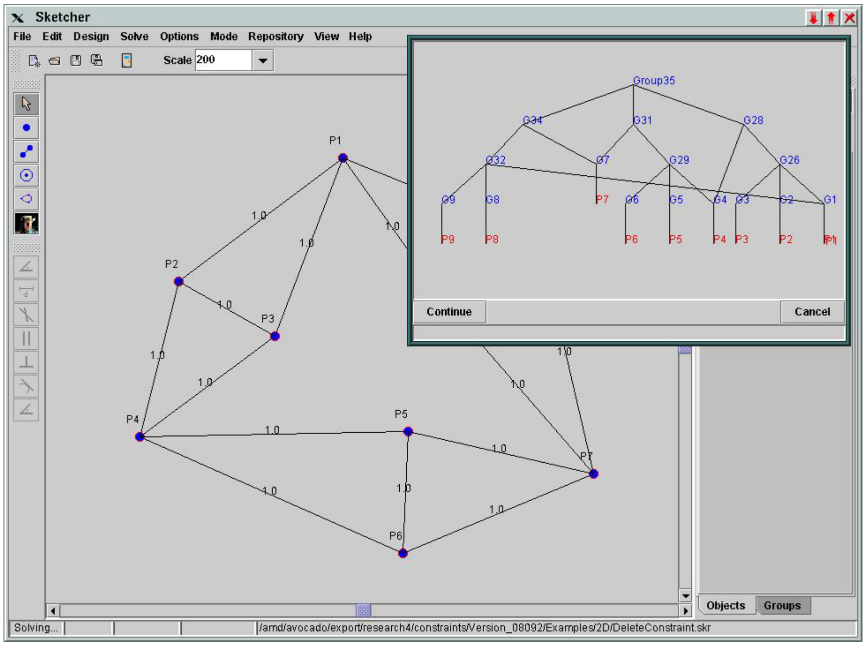


Figure 3-20 Updated sketch.

Online Solving

Online solving allows the system to be solved as and when the user draws the input sketch. First a module called the Simplesolver is used to solve the constraint systems. This module is a hack that is part of Sketcher itself used only for display purposes. It does not invoke the back-end. But the Simplesolver gives up at some point. The user can also choose to invoke the back-end each time a constraint is added. This is the online mode of solving. The idea is to have the simple solver do the online solving and once it gives up invoke the back-end for the same purpose. But as of now the Simplesolver and the online solving using the back-end are totally independent.

Simplesolver

While drawing the sketch when the constraints are input the simple solver is invoked. There are two versions of the Simplesolver in Sketcher. The one we shall discuss tries to solve constraints like incidences, parallelism, perpendicularity etc. It goes through the list of constraints once, solving all of them.

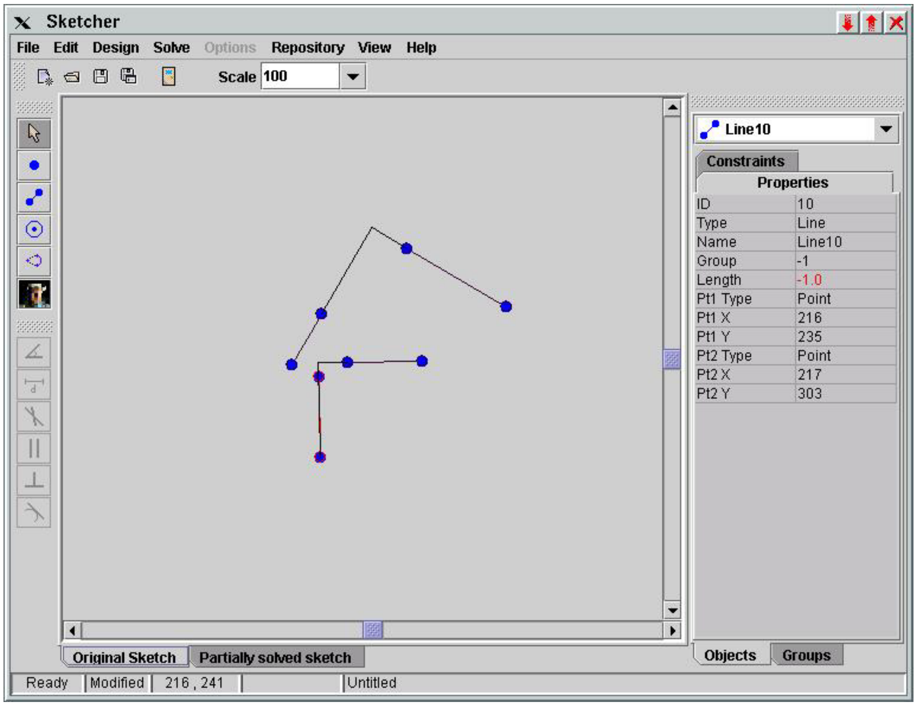


Figure 3-21 Sketch with two pairs of line segments with two perpendicularity constraints (solved by the Simplesolver) between line segments in each pair.

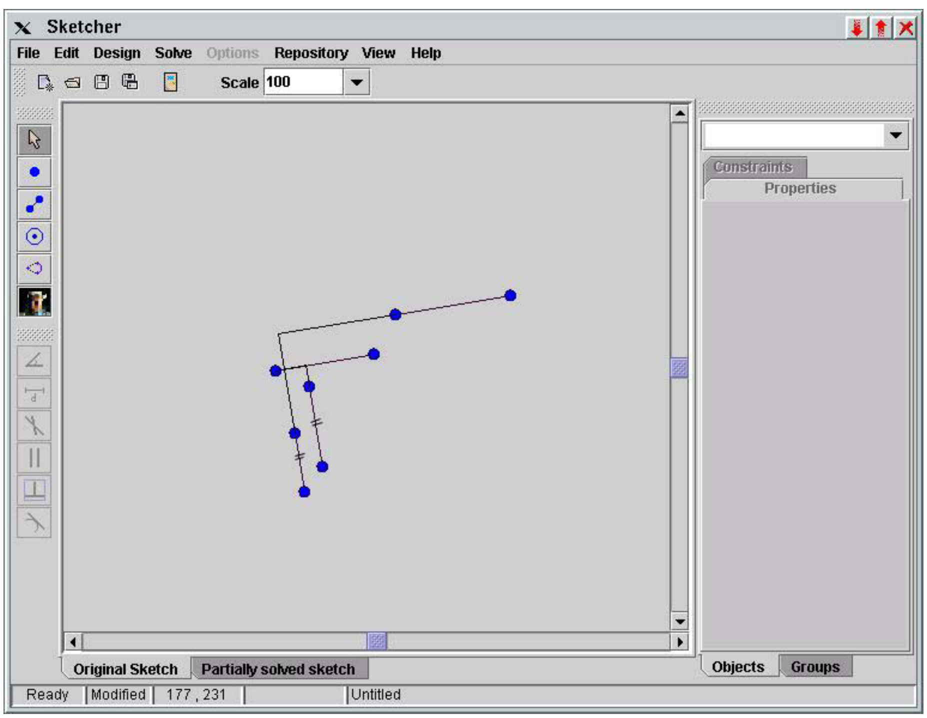


Figure 3-22 Sketch with one parallelism constraint (solved by Simplesolver) between one line segment of each pair.

The above figures show how the Simplesolver maintaines perpendicularity constraints between the two pairs of line segments and a parallelism between two line segments. We can see how the perpendicularity between the other two line segments which do not have an explicit constraint placed between them, is maintained as a side-effect.

Simple solving becomes a non-trivial job once the number of constraints increases, because solving one constraint may move the objects so that some other constraint is not satisfied anymore. And in trying to solve that constraint yet another constraint may not satisfied.

Online Solving Mode

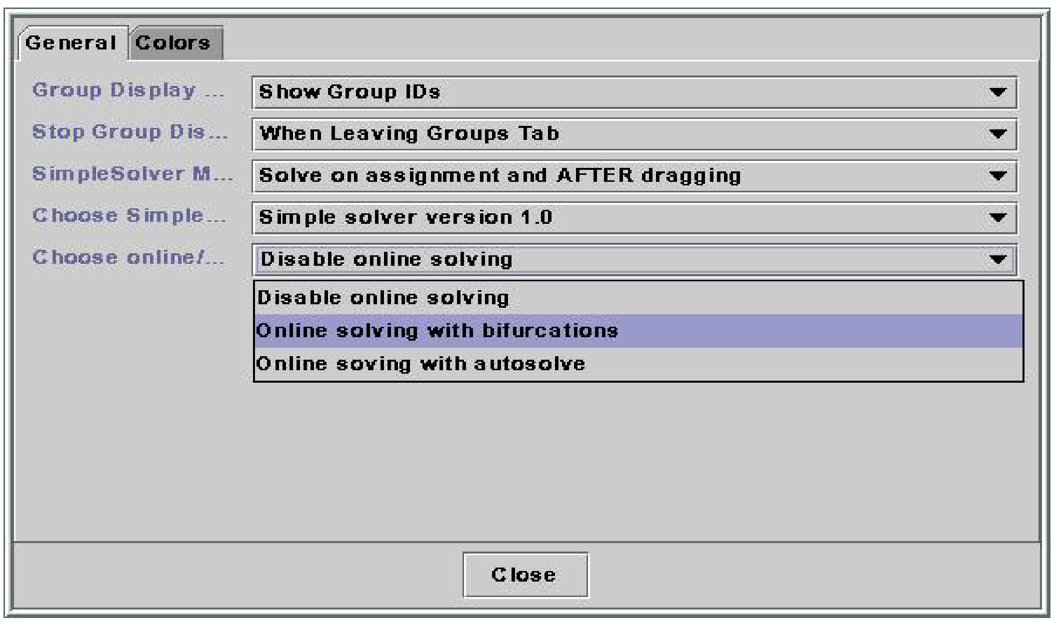


Figure3-23 Screen shot displaying the preferences window.

In this mode again the user can choose to use the Get-Bifurcation mode or the Auto-Solve mode. So whenever a constraint is placed in the sketch the back-end gets invoked and the solution is displayed after solving. Basically the back-end treats each addition as an update to the original sketch. So the back-end behaves just as it does in the "Add constraint" or "Add shape and constraint" mode.

Extensibility and Flexibility

In most applications there are some sub-systems used frequently. Sketcher allows the user to save such sub-systems in a repository for easy re-use. The user may choose to save even a system already solved using FRONTIER. User might want to interface a new constraint system with such a solved, rigid system. User might also want to use a rigid system represented in a totally different representation like Brep. Sketcher allows him to represent these rigid systems with an object that has the same degrees of freedom as the rigid systems.

Repository

Very often when a person does extensive work in the same domain then he/she might have to use the same kind of systems/sub-systems. So the user may find it useful to have some basic pieces of his sketch e.g. triangles or diamonds, saved in a repository so he/she can just pick it up from there. Sketcher provides exactly this facility. The user can save sketches in the repository and retrieve them whenever he requires them.

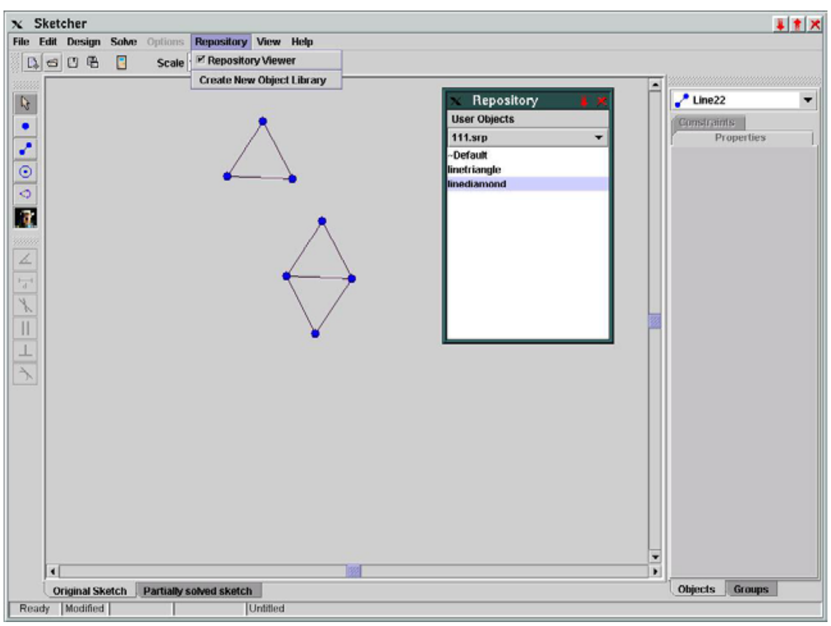


Figure 3-24 Screen shot showing the repository and the sub-systems drawn using it.

The figure 3-24 shows the repository window and the triangle and diamond shapes drawn using the repository. The repository can be used to save solved systems also in a similar fashion.

Bitmap

The bitmap is only a representation of a rigid system. This rigid system could be a solved system out of the repository or a system represented using a different representation language. If the user wishes to interface this rigid system with another constraint system by adding some more objects and constraining the rigid body with the newly added objects then he may represent the rigid part of the whole system with the bitmap. The bitmap is provided with two handles, which allow the use to rotate, scale the image. The bitmap is treated just like a line segment. The back-end does not know the existence of the bitmap object at all. The scaling of the bitmap corresponds to change in length of the line segment and the rotation of the bitmap corresponds to the change in slope of the line segment. And the constraints that can be imposed on it are also same as those that can be imposed on a line segment. The ESM has no way of differentiating between a line segment and an image shape. When the input data is sent to the solver the image shape is represented by as a line segment.

The figure below shows a constraint system consisting of a bitmap object, two line segments and a point. The point has perpendicular distance constraints with the two line segments and the bitmap. There is an angle constraint between one of the line segment and the bitmap and another angle constraint between the two line segments. There are three incidence constraints between the line segments and the bitmap object. This system can be solved in the normal fashion using any of the various modes of solving.

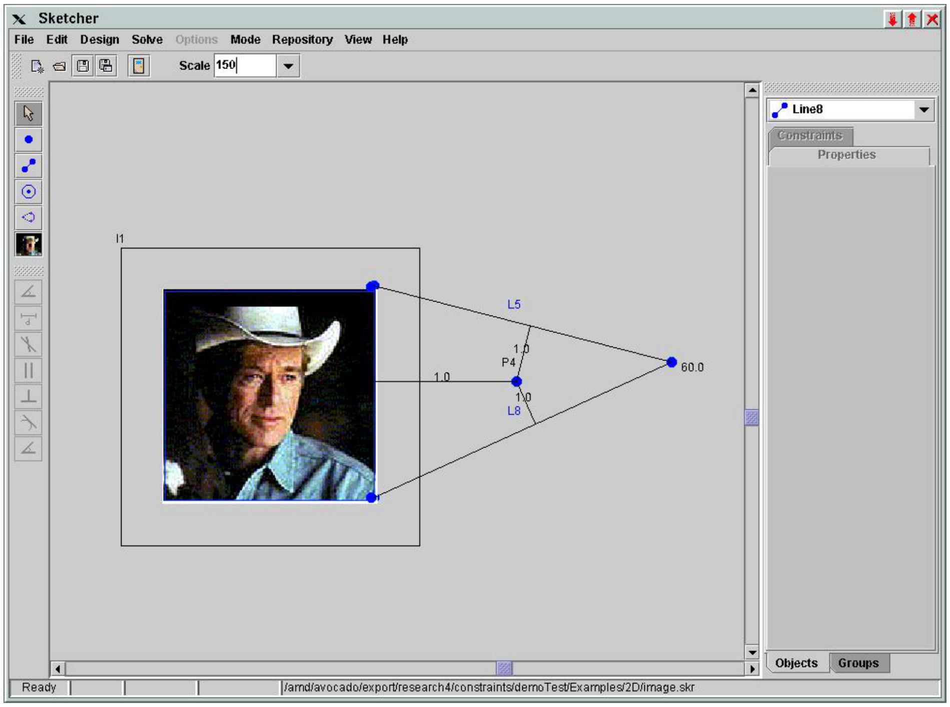


Figure 3-25 A constraint system consisting of an image shape, a point and two line segments. The system had 3 distance constraints and 2 angle constraints.

The following figures show the DR Plan for the system and the final solved sketch. In the final solved sketch we can see that the bitmap image has be rotated to form a triangle with the other two line segments.

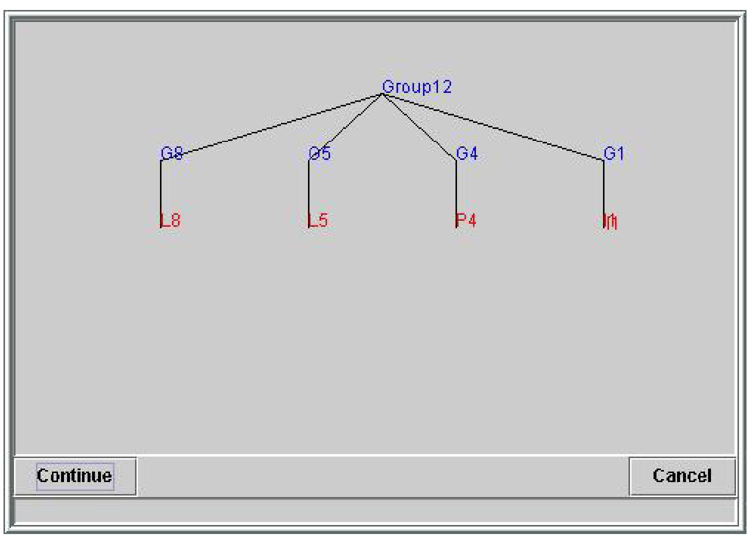


Figure 3-26 The DR Plan for the above constraint system.

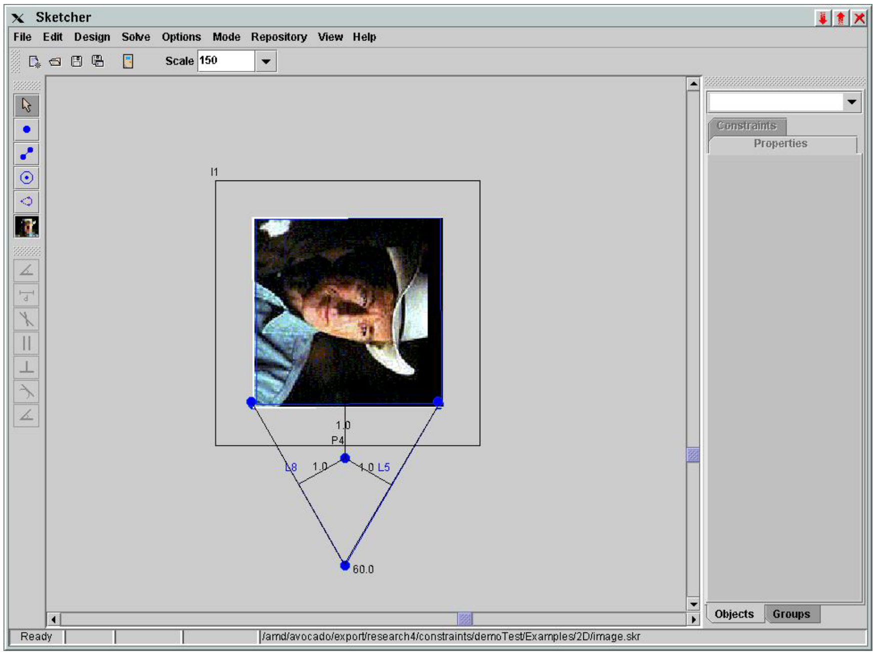


Figure 3-27 The final solution for the constraint system.

Adaptability

For the (Macro)Molecular structure and assembly pathway modeling(static) applications the various atomic forces and bonds can be modeled as geometric constraints. For this application in particular the torsion angle constraint has been added to the 2D input 3D output Sketcher. The torsion angle constraints can be used to model some of the forces in molecules.

The torsion constraint is defined between four points. The first two points picked define the axis for the torsion angle. The figure 3-28 shows a 3D constraint system. In this system there is a torsion angle constraint between points P1, P2, P5, and P6. P1 and P5 define the axis. To interpret the constraint, imagine P1, P2 to be a line segment and P5, P6 to be another line segment. Then the torsion angle is the relative angle of the projections of these imaginary line segments on planes perpendicular to the axis, about the axis.

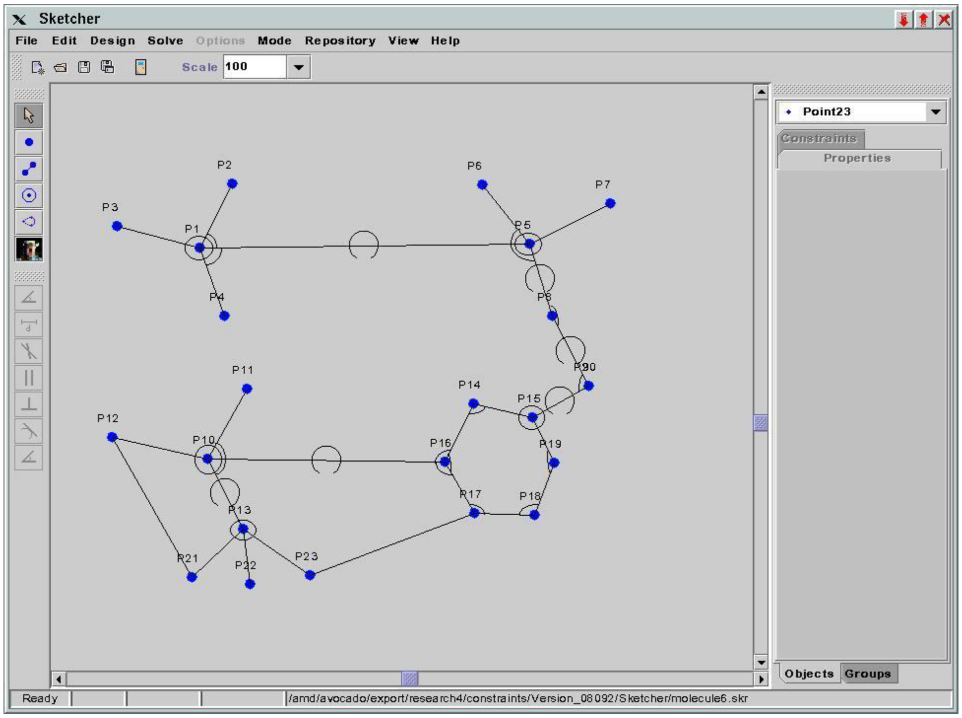


Figure 3-28 3D input sketch containing point shapes, angles, distances and torsion angle constraints, drawn using the 2D input 3D output Sketcher.

In such applications the objects involved in the system are atoms of different elements like carbon, oxygen, hydrogen etc. Keeping this in mind two additional properties have been added to the point objects—color and radius. These properties help the user in differentiating the different elements and also visualizing the molecule. Thus Sketcher offers both stick and ball and electron cloud models of molecules for display. There exist different types of bonds within these molecules like single bonds and double bonds. To simulate this, the distance constraints have been given an additional color property for display. These additional properties are not sent to the back-end right. They are used only for display purposes. In future they can be sent to the solver if some specific operations need to be performed on some particular atoms.

The figure 3-29 shows a 3D constraint system drawn on a 2D panel.

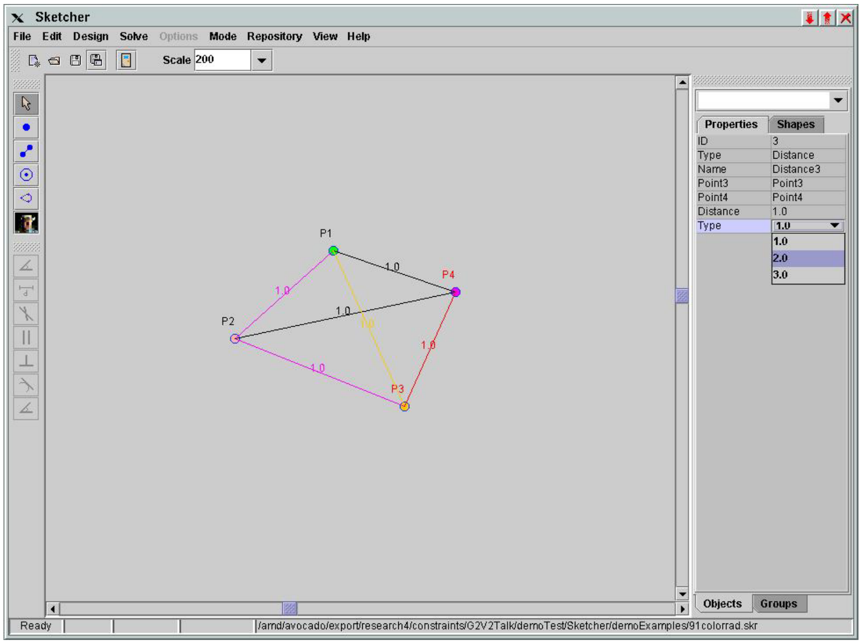


Figure 3-29 Input sketch of a 3D constraint system drawn using the 2D input 3D output Sketcher showing points with different colors and distance constraints of different types.

The color and the radius properties of the four points in the system have be changed. Similarly also the type property of the six distance constraints have been changed. The final solution to the 3D constraint system in figure 3-29 retains of the color and radii properties of points as well as the type property of the distance constraints.

Partially Solved Sketch

FRONTIER solves complex problems step by step walking the DAG from bottom towards the root. As each cluster in the DAG is solved a rigid portion of the final output it obtained. So it is possible to display the intermediate pieces of the final sketch to the user. This would be very useful to the user so he can look at the final sketch as it is being built. For example in the example shown below, there are eight points and fifteen distance constraints, forming three diamonds. For instance points P1, P2, P3 and P4 form one diamond. The figure 3-31 shows the DR Plan obtained for this system. The three diamonds each form a cluster in the DR DAG. So each of these diamonds are solved separately. At the next level the solutions of these three sub-systems are assembled to obtain the final solution.

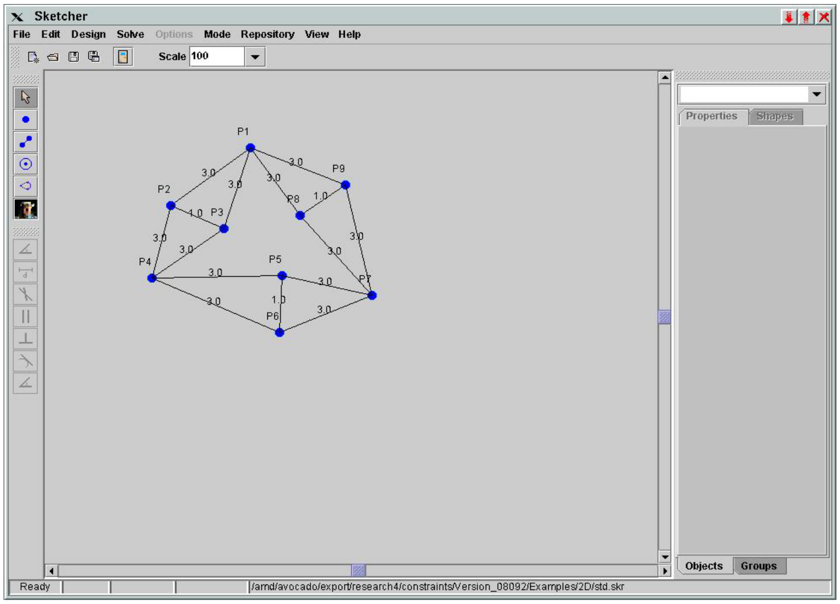


Figure 3-30 Input constraint system.

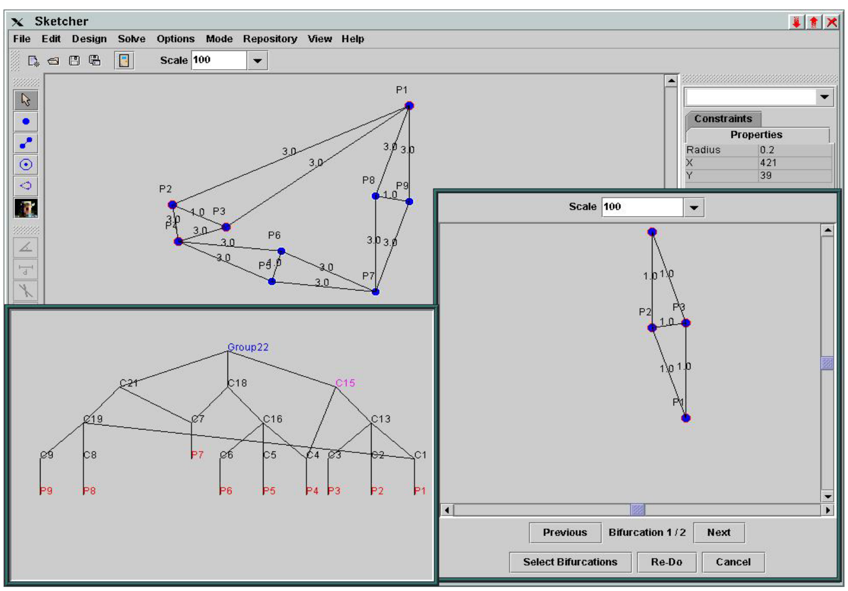


Figure 3-31 Partially solved sketch at an intermediate point during solving.

Sketcher tries to display the partially solved sketch. This becomes very tricky because whenever a sub-system is solved, the coordinates of all the objects in the sub-system are defined in the sub-system's own coordinate space. So if these objects are displayed directly using the coordinates sent by the ESM then most of the objects will overlap and it will become very hard to decipher the sketch. Hence Sketcher fixes the coordinates of one of the objects in the cluster, to wherever the user placed it and then moves rest of the objects in the cluster maintaining the relative positions of all the objects. It also tries to rotate the clusters in such a way that the orientation of at least one object with respect to one other object is maintained as it was in the user input sketch. This manipulation of the partially solved sketch fails to work when a cluster that is being solved presently has two objects in common with clusters that have already been solved. Because at this point any one of those two objects will have to be moved and this may disrupt the solution of some formerly solved sub-system. So once this happens Sketcher gives up.

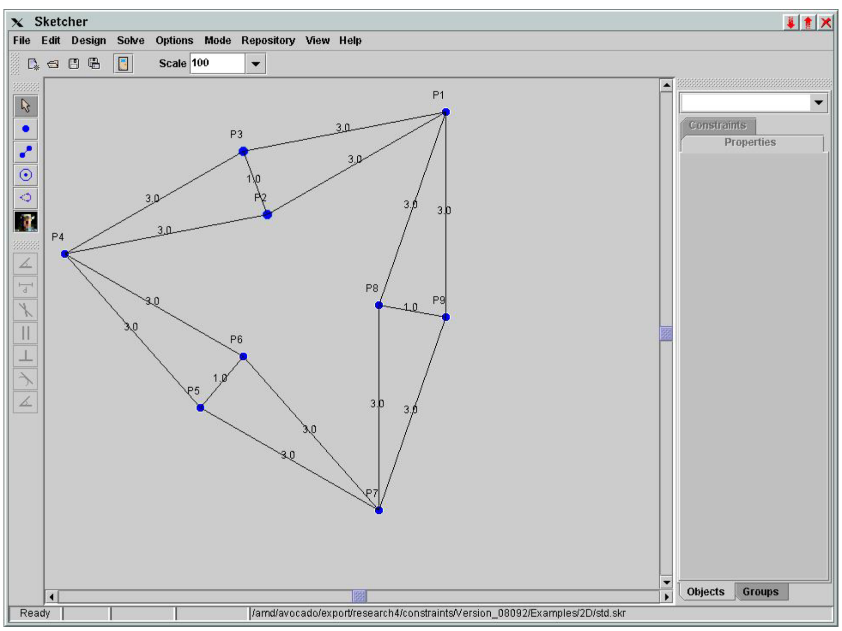


Figure 3-32 Final solved sketch.

The fact that we are only interested in the relative positions of the objects in the system allows Sketcher to move and rotate the final solution also, in such way that one of the shapes in the system is placed in the same place where the user placed it. This way it becomes easier for the user to interpret the solution.

Error Handling

Sometimes the input sketch given to the solver may be under-constrained. In this case the DR Plan is a forest, so the user can detect that easily. But in case the sketch is over-constrained then the user is informed about the over constraint when the DR Plan is first displayed. Sketcher also tells the user at which cluster (level) the system is over constrained. If the input system has no solution at all then it has to be reported to the user accordingly. In such case the solver sends back a flag, which reports the appropriate message to the user. Then the user can modify the sketch so as to fix the problem.

The Sketcher eliminates the need of error checking in the input constraints while solving. This is done by allowing the user to input only legal constraints in between any given set of objects for example when two points are picked only legal constraints are incidence and distance. All the other constraints are disabled. Similarly while working in the update mode Sketcher allows the user to only make the updates that he chose to do earlier.

Easy Editability

Keeping in mind easy edit ability of the sketch, Sketcher tries to keep the drawing tools as intuitive as possible. For example: The circles are made up only one sub shape the center of the circle. It allows the circles to expand or contract when the circle (graphics) itself is dragged. The new radius of the circle is calculated using the distance by which the mouse was dragged and then the circle is redrawn. Dragging the center of the circle can change the position of the circle. Similarly the arc is constructed using three sub-shapes. These can be viewed as control points for the arc. The arc has four degrees of freedom. The x, y co-ordinates of the center of the arc, the radius of the arc and the angle of the arc. Hence if the "control points" of the arc are dragged then we cannot be sure whether the user is trying to increase the angle of the arc or its radius. So differentiate between these two actions Sketcher allows the user to fix the angle while increasing the radius and similarly fix the radius while increasing the angle. Sketcher also allows the user to do both simultaneously. Dragging the center of the arc can change the position of the arc.

CHAPTER 4
DESIGN AND IMPLEMENTATION OF THE SKETCHER

The most important qualities that any user interface should have are robustness and scalability. Also in particular for FRONTIER's user interface robustness and extensibility are essential characteristics. Also the representation language used for the features and constraints should be general. It is even more important that the representation be easily convertible to other representation languages so that an external system can call FRONTIER. The code of Sketcher has been designed keeping all this in mind.

The main features of the design are modularity and the hierarchy within the objects. These properties are a direct effect of the object-oriented nature of the code. This simplifies any future expansion. There are three different versions of Sketcher, the 2D Sketcher, the 2D-input 3D-output Sketcher, and the 3D Sketcher. We shall first discuss the implementation details of the 2D Sketcher. The 2D-input 3D-output Sketcher and the 3D Sketcher have a similar design, thus maintaining consistency in addition to the other features.

2D Sketcher

The 2D Sketcher is written in JAVA using JAVA AWT and swing packages. The application can be divided (implementation-wise) into two major components— the part that handles the display with all the associated components and the interface that communicates with the UTU. The 2D Sketcher has the following primitive geometric objects: points, lines, rays, line segments, circles and arcs and the image object. The constraints that can be applied to these objects are: distance, incidence, perpendicularity, parallel and tangency constraints. Each of the shapes has a unique ID, and similarly each of the constraints have a unique ID.

Shapes

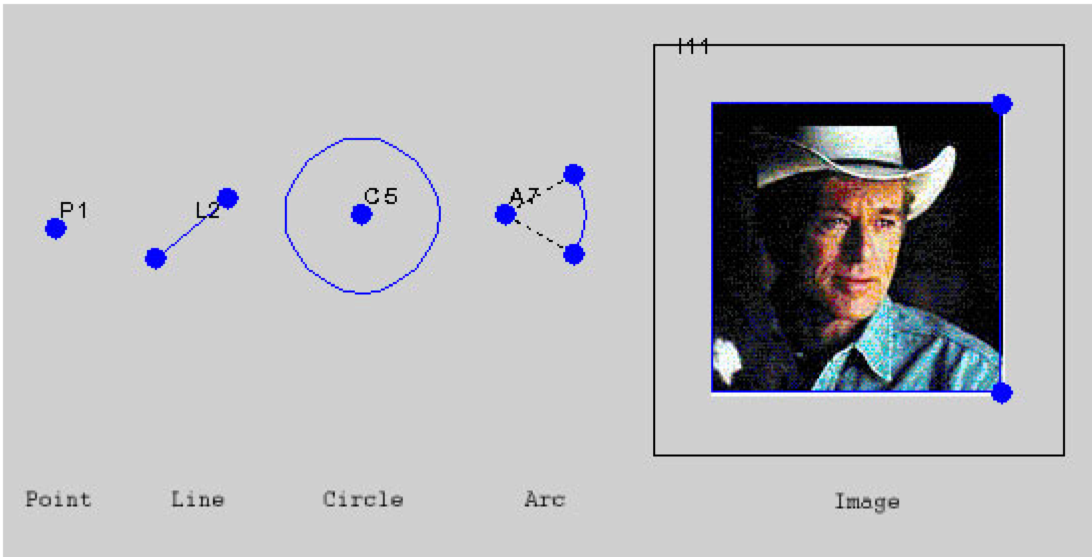


Figure 4-1 Shapes

All the shapes descend from a baseShape class. The properties that are common to all the shapes like: Name – Name of the shape, ID – Unique ID of that particular instance, ShapeTypeID – Unique ID for that class of shape, Selected – Flag that indicated whether the shape is selected or not, Color etc., are declared in this class. The Line, circle and the arc shapes are descendents of this class. The normalShape class is a descendent of the baseShape class. It also includes another property. The objects of this class have x, y coordinates defined by their position on the screen. These objects can be dragged on the screen using the mouse. The point and the image shapes are both descendents of this class.

The following figures show the object hierarchy within the shapes and the various attributes of each shape.

Object Hierarchy in Shapes

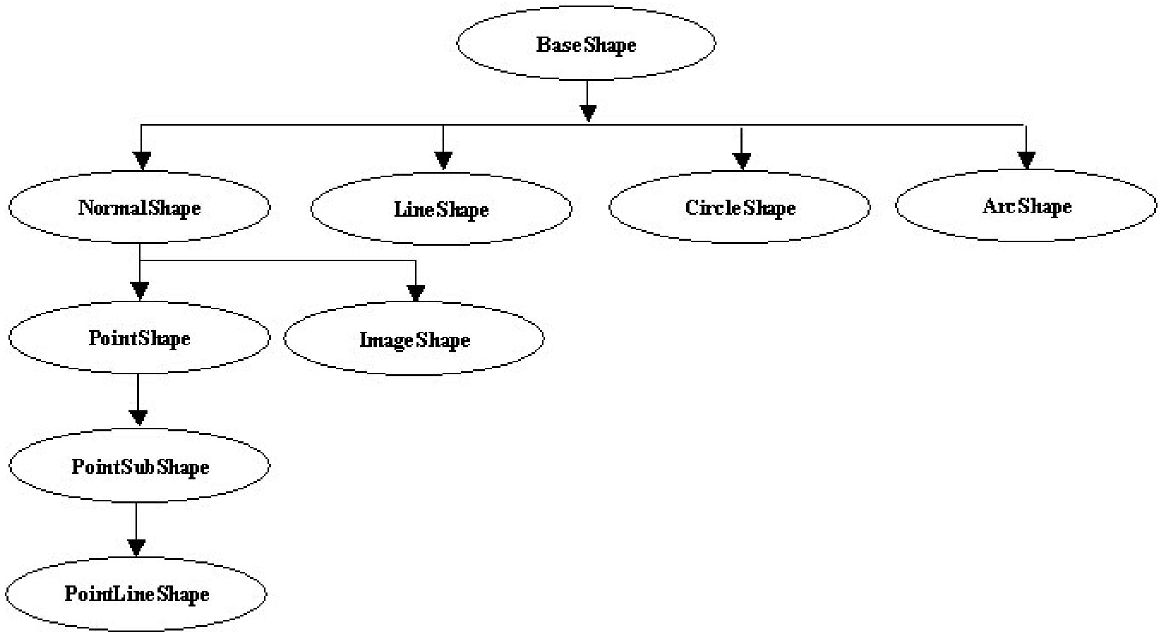


Figure 4-2 Hierarchy within shapes

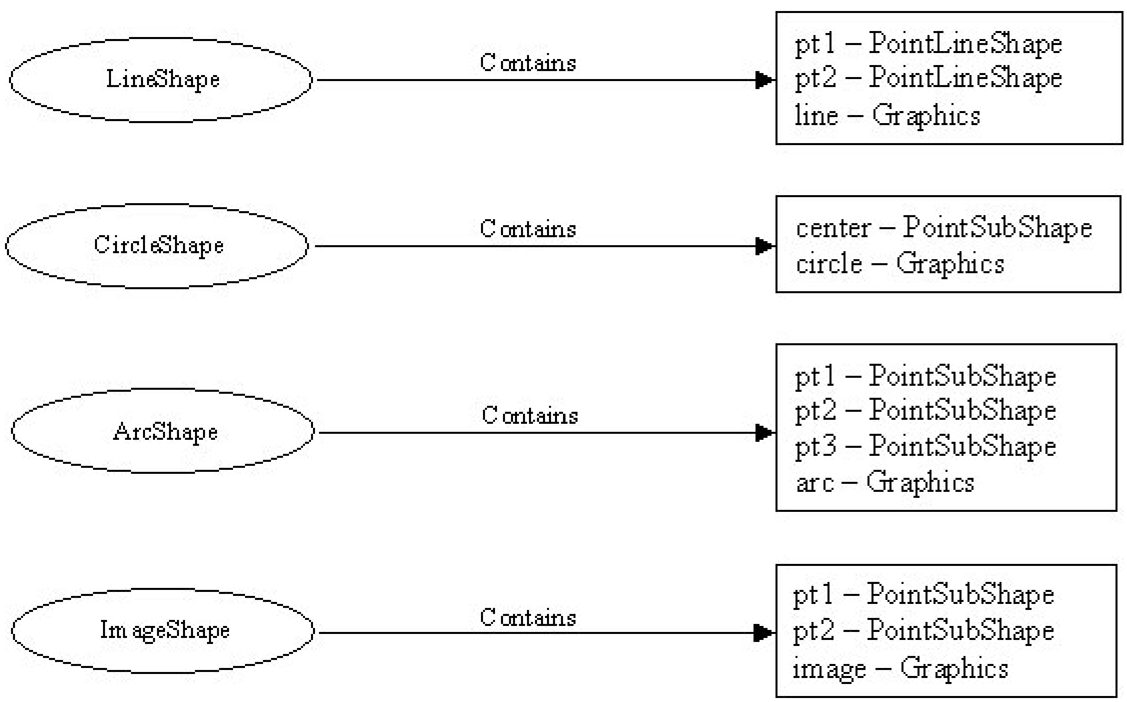


Figure 4-3 Sub-Shapes within shapes

Another key feature of the design is the pointSubShape class, which is a descendent of the pointShape class. All the other objects like the line, circle, arc and the image shapes have sub-shapes that are of type pointSubShape. For example the line segment has two sub-shapes that are its end points. In case of the line segments the sub-shapes are of type pointLineShape. The pointLineShape is a descendent of the pointSubShape, which can be set to infinity. The line and the ray are considered to be special cases of the line segment where both or one of the end points is set to infinity, respectively. The positions of the shapes that are not of type normalShape are defined by the positions of their sub-shapes that are of type normalShape.

This object-oriented design proves to be very advantageous for most of the functionalities of Sketcher. Good examples are the writeToStream/readFromStream methods that are used to save to file and read from one respectively. These methods in the baseShape class write/read the common properties and then call the writeAdditionalProps/readAdditionalProps method in the derived classes to write/read additional properties of the shapes. In each derived class (of shapes) these methods are overridden to write/read the specific properties of that particular shape and then calls the writeAdditionalProps/readAdditionalProps methods of the its sub-shapes, if any. For example in lineShape this method writes/reads the length of the line segment and then calls the writeAdditionalProps/readAdditionalProps methods of its end points. In the writeAdditionalProps/readAdditionalProps methods of the end points the coordinates of the points are written/read. Similarly in case of circle shape the radius is written/read and then the writeAdditionalProps/readAdditionalProps method of its center is called, so on and so forth.

Constraints

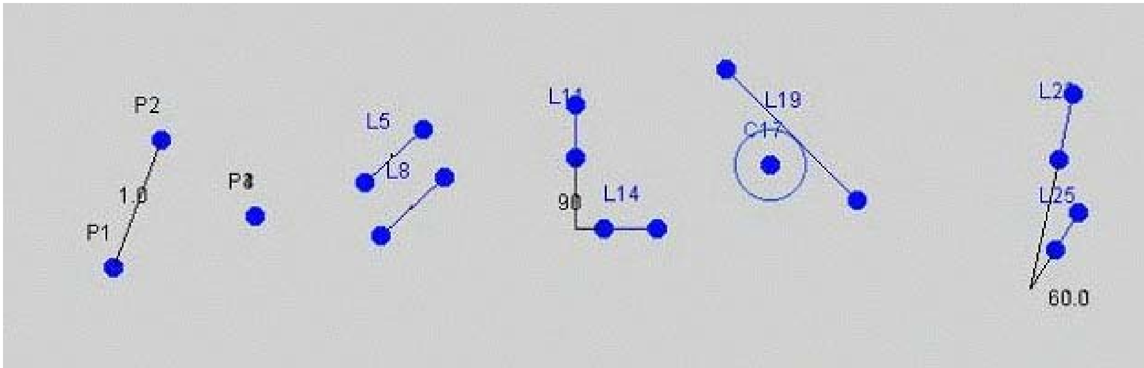


Figure 4-4 Constraints

The constraints are also implemented in a similar manner. All constraints descend from a baseConstraint class. The metric constraints like distance and angle have a property associated with them that define their value while the logical constraints like tangency, parallelism, and perpendicularity do not. Each constraint maintains an array of shapes that are associated with that constraint.

The figure below shows the object hierarchy within constraints.

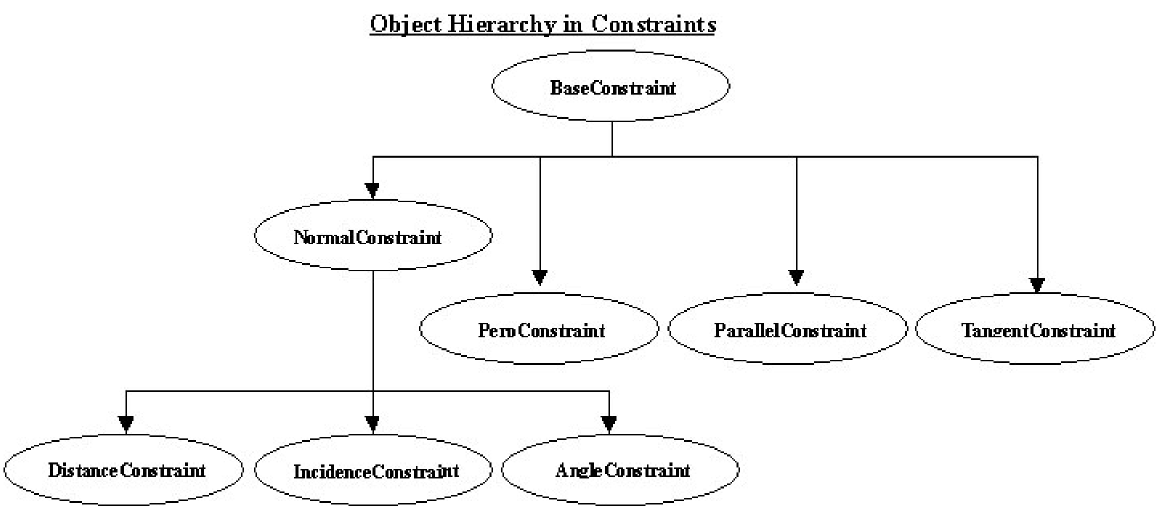


Figure 4-5 Hierarchy within constraints.

For some of the constraints just specifying the shapes involved, is not enough. We also need to specify which part (“sub-shape”) of that shape is involved. For example if we specify an incidence constraint between a point and a line segment then we have to specify which end point of the line segment is incident with the point. Similar to the hierarchy in shapes there is normalConstraint class that descends from the baseConstraint class. All the descendents of the normalConstraint class have an additional property that specifies which sub-shape of each of the two shapes are involved in the constraint.

Displaying the constraints using graphics, by itself is an interesting issue. The representation of the constraint needs to be intuitive enough for the user to be able to interpret it easily. The simple solver module usually solves the incidence and tangency constraints for display, and is not represented explicitly by any graphics. A distance constraint is represented by a line segment joining the two sub-shapes/shapes involved in the constraint. If the distance is a perpendicular distance between a point and a line then a perpendicular line is drawn from that point to the line. For displaying an angle constraint the two lines involved in the constraint are extended till they meet and at the point where they meet a small arc is drawn. The angle is set to 60 degrees by default when the constraint is created. The parallelism constraint is displayed by an equal number of small marks on the two lines involved in the constraint. The perpendicularity constraint is treated like an angle constraint with a 90-degree angle.

Again the object-oriented design proves to be very useful with the constraints, for the same reasons discussed earlier. For example the method drawConstraint is called every time the screen is repainted. And this method is overridden in all the constraints to display the specific graphics of that constraint.

Representation of Objects for Communication

The communication of the data is one of the most important functions that Sketcher performs. All the modules of FRONTIER other than Sketcher are written in C/C++ hence

JNI is used to communicate with those modules. Sketcher invokes the other modules in the same way it calls any of its functions. The input sketches are translated into two arrays, one integer array and one double array having a pre-decided format. These arrays are passed on as arguments to the UTU. The UTU reads off some of the data from these arrays and instructs the other modules accordingly. The UTU then passes on the arrays to the DR Planner or the ESM. The communication is two-way. The DR Planner and the ESM also send information to the Sketcher through the UTU. The same arrays are used for this purpose. When the user hits solve the sketch is converted into the arrays using a standard representation. The following figures describe this representation.

```
Flag for the UTU
Flag for the Equation generator
Shapes data
....
-1
Constraints data
....
-1
Groups data
```

Figure 4-6 General representation of the constraint system.

```
Shapes

Integer Array:
ShapeTypeID
ID
Double Array:
for each subshape
{
        X co-ordinate of subshape
        Y co-ordinate of subshape
}
Length/radius (if line segment or circle)
```

```
Constraints

Integer Array:
ConstraintTypeID
ID
No. of shapes involved
for each shape
{
        ID of the shape
        Subshape of the shape
}
Double Array:
Distance/Angle (if distance or angle)
```

Figure 4-7 General representation of Shapes and Constraints data.

Groups

<u>Integer Array</u>:
ID
No. of shapes only belonging to this group
for each shape
{
    ID
}
No. of children
for each child
{
    Groups data of the child
}

Figure 4-8 General representation of groups (Input partial decomposition) data.

The flag for the UTU instructs the UTU to direct the other modules appropriately. Here is the generic sequence of operations that take place while solving a constraint system. The first time the UTU is invoked it sends the arrays to the DR Planner. The DR Planner uses the data to generate the DR Plan. The DR Plan is sent back to the Sketcher that reads it and displays it to the user. All the intermediate data and the data structures are also saved in the arrays along with the DR Plan. All this data is placed in the beginning of the arrays when they are sent back. The Sketcher skips this data and reads only the part that is relevant to it. Along with the data Sketcher receives a flag that indicates whether the data sent to it is the DR Plan, the bifurcation data, the final sketch data or some error message.

-2 – DR Plan
0 – Bifurcation data
1 – Final sketch data

Figure 4-9 Flags for Sketcher.

If the user disapproves the DR Plan and makes some changes to the input constraint system then the Sketcher informs the UTU about the changes. Again the UTU flag is used instruct the UTU appropriately so that the DR Planner can take the changes into consideration and modify the DR Plan. If the user approves the DR Plan the UTU is instructed to continue. The UTU then sends the DR Plan to the ESM that begins the solving process starting with the smallest sub-systems. Any time when the ESM gets more than one possible solution to a sub-system all possible solutions are placed in the arrays and sent back to the Sketcher. Sketcher displays these possible solutions to the user and lets him/her pick the solution he/she wants. The ESM is informed of this choice. Since all the intermediate data is saved in the arrays the ESM can pick up the solving from where it left. Thus it uses the users choices and solves the whole system. Finally when the final solution is obtained it is sent back to the Sketcher in the same arrays. Sketcher reads the data off the arrays and displays the final sketch.

2D-Input 3D-Output Sketcher

This version of Sketcher is capable of handling both 2D and 3D constraint systems. The user can choose to work in either the 2D mode or the 3D mode. In the 2D mode this version, superficially functions exactly like the 2D version. In the 3D mode the user is allowed to draw the 3D sketch on a 2D canvas and solve it in the 3D mode. However the bifurcations and the final sketch are displayed on the 3D canvas. The solver does not use the input coordinates of the objects to calculate the solution. So the Sketcher simply sends a 0 as the z coordinate for all the objects. The other modules treat the sketch as a 3D problem and solve for all the 3 dimensions. Now when the bifurcations have to be displayed all the objects have 3 meaningful dimensions so they are displayed on a 3D canvas. And the final out put is also displayed on a 3D canvas.

2D Implementation

The implementation of the 2D parts of this Sketcher is similar to that of the 2D Sketcher. The only difference is that every time a 2D shape or constraint is created a corresponding 3D shape or constraint is created and attached to the scenegraph, which may or may not be used in future. This is done to allow the user to switch between the two modes easily at any point.

3D Implementation

As far as the implementation of the 3D objects (Shapes and Constraints) is concerned, it is same in both the 2D-input 3D-output version and the 3D version. These objects are implemented using JAVA3D [1].

2D Sketcher is written using JAVA AWT and JAVA Swing. The canvas3D that is a J3D component has to be placed on top of a JPanel that is a Swing component to display the 3D scene. But JAVA3D and JAVA Swing are not fully compatible with each other [12]. Swing is lightweight and J3D is heavy weight. This means that a canvas3D will draw on top of Swing objects no matter what order Swing thinks it should draw in. A heavyweight component is one that is associated with its own native screen resource (commonly known as a peer). A lightweight component is one that "borrows" the screen resource of an ancestor (which means it has no native resource of its own -- so it's "lighter"). This means that the Swing menus will not show up on top of a canvas3D. To walk around this the Swing team has introduced a flag, which forces the Swing menus to be heavyweight. This flag can be set by doing the following:

JPopupMenu.setDefaultLightWeightPopupEnabled(false);

Similarly there is another flag to allow tool tips to be visible on top of canvas3D. This flag can be set by doing the following:

ToolTipManager.sharedInstance().setLightWeightPopupEnabled(false);

Scenegraph

Java3D uses a scene graph for rendering purposes. The scene graph is a graph structure that contains Java3D *nodes*. Each node connection represents a parent-child relationship.

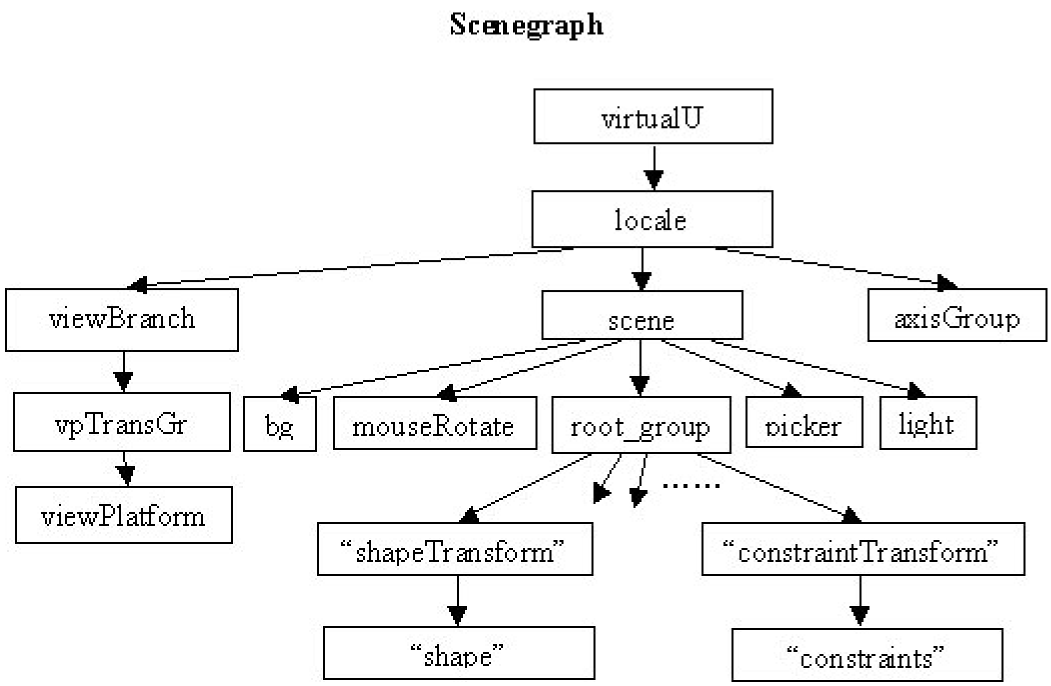


Figure 4-10 Partial scenegraph used to display 3D scenes in Sketcher.

A scene graph is constructed in such a way that state information cannot be shared among sub-graphs. This enables Java3D to render scenes concurrently.

The viewBranch node governs the camera position. The axisGroup node contains the nodes corresponding to X, Y and Z axes and the X-Z plane that are displayed at all times in the 3D scene to help the user visualize the 3D space. The scene node has a mouseRotate node that allows the user to rotate the whole scene. Along with the scene the axes and the plane are also rotated using an other node (not shown in the figure). The picker node under the scene node allows the user to pick any of the objects that are present in the scene. The light node is responsible for the lighting in the scene. The root_group node has the all the nodes corresponding to the different shapes and constraints in the scene.

Shapes

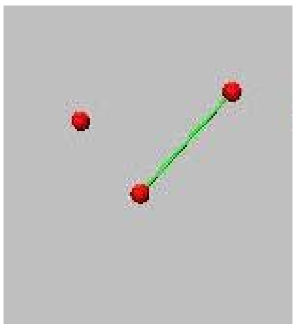


Figure 4-11 Point and Line shapes on a 3D canvas in the 2D-input 3D-output Sketcher.

The point is represented by a small sphere in 3D space. The basic hierarchy among the shapes is maintained exactly as it is in 2D Sketcher. The point is the basic shape and all the other shapes have sub-shapes that are points and act as handle to manipulate the position or the dimensions of the object.

The line segment has two sub-shapes that are its end points. The line segment consists of a thin cylinder connecting the two end points. The orientation and the length are calculated using the positions of the two endpoints of the line segments. The length is changed by appropriately scaling the cylinder along its axis. Then the rotation is applied to the scaled cylinder to position it accurately.

Constraints

Presently the 2d-input 3d-output Sketcher has got only two constraints the distance constraint and angle constraint. The Angle constraint in this version is slightly different from that in 2D Sketcher. For 3D examples it allows an angle constraint to be specified between three points/sub-shapes. The three shapes involved in an angle constraint can be sub-shapes of line segments or a point objects. It considers the point that was picked second among the three points to be the vertex of the angle.

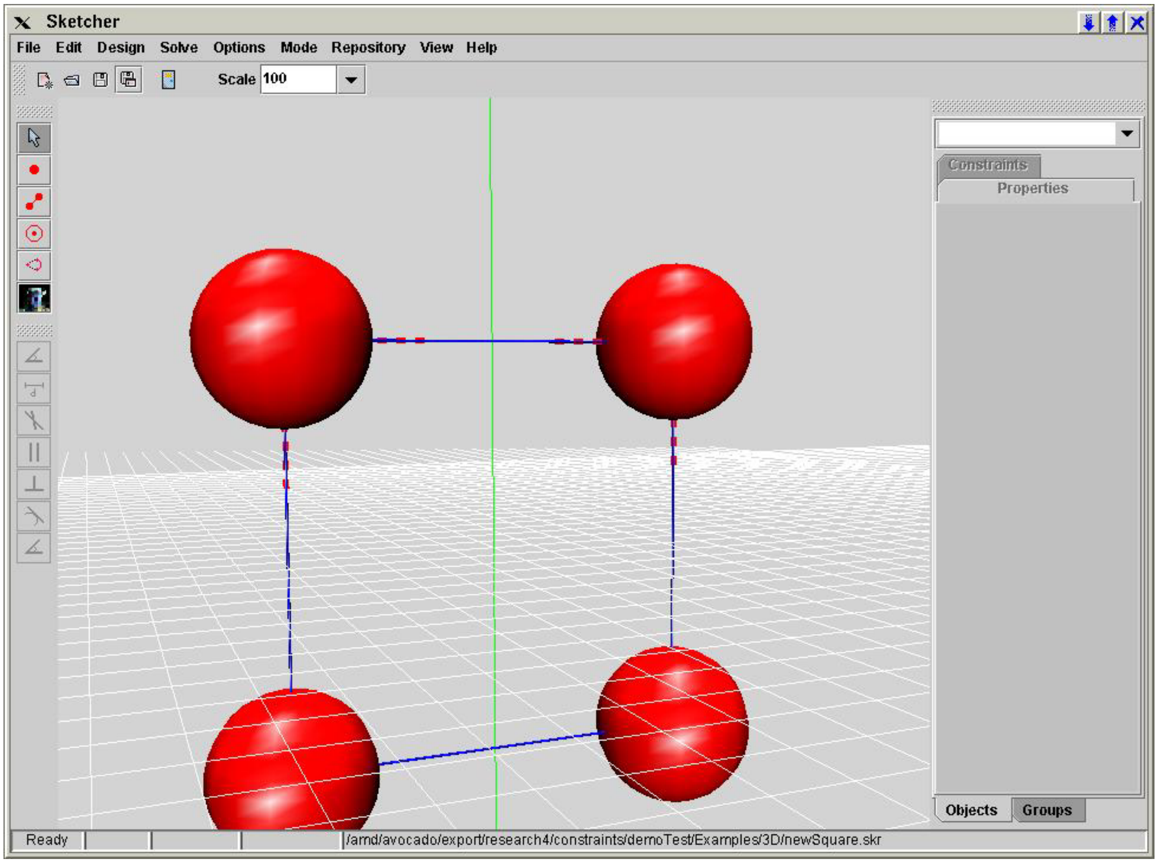


Figure 4-12 Screen shot of the final sketch of a constraint system consisting of four points, four distance constraints and two angle constraints.

The above figure shows a 3D constraint system displayed using the 2D-input 3D-output Sketcher. The above system is the final solution to a 3D constraint system displayed on a 3D canvas. It consists of four points, four distance constraints, and two angle constraints having the upper two vertices of the square as their vertices. A distance constraint is represented by a thin blue line between the shapes involved in the constraint. An angle constraint is represented by two thin, red, dotted lines starting from the vertex of the angle and extending in the directions of the other two shapes involved in the constraint.

3D Sketcher

The 3D Sketcher is the version of Sketcher that allows the user to interactively sketch in 3D space. There are many issues that need to be addressed in case of a 3D GUI [14]. Part of them spring from the field of HCI (human computer interaction). People have realized that, to make the user interface more intuitive it is very important to pay attention the user's perception of the 3D world and to make it more user-friendly the user's interactions with the UI should be made more natural [3]. The 3D Sketcher also has to deal with these issues.

The 3D Sketcher uses the same object hierarchy as the 2D Sketcher and the 2D-input 3D-output Sketcher, but the 3D Sketcher has some more 3D Shapes. This version is not yet integrated with the other modules.

Shapes

In addition to the points and the line segments that are used for the 3D examples of the 2D-input 3D-output Sketcher there are some more shapes in the 3D Sketcher.

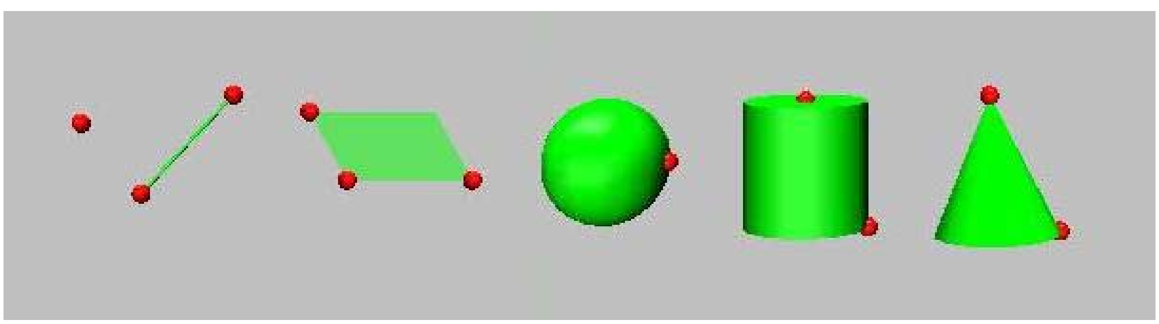


Figure 4-13 Shapes in 3D Sketcher.

The point shape and the line segment shape in the 3D Sketcher are same as that in 2D-input 3D-output Sketcher. While drawing the sketch the user is allowed manipulate the position, length and orientation of the line segment by dragging any of its end points. The ray and line object are not yet implemented in this Sketcher.

The plane shape is a translucent quadrilateral having three handles, each positioned at one of the four corners of the rectangle. The plane of the object can be changed by dragging the handles. The handle can also be used to extrude and or rotate the plane shape as required.

The sphere object has only one handle, which is used to change the radius of the sphere. The position can be changed by dragging the sphere itself.

The cylinder object is similar to a line segment except that it has one more handle that is used to change the radius of the cylinder. This handle is placed at on the circumference of the cylinder. The handles along the axis of the cylinder can be used to rotate the cylinder or change the height of the cylinder.

The cone object is similar to the cylinder object having three handles.

Constraints

The only two types of constraints in the 3D Sketcher presently are distance constraint and angle constraint. These are similar to those in the 2D-input 3D-output Sketcher.

Issues in Interactive 3D Scene Editing

Today there are many 3D user interfaces that are commercially available. Most of them use 3D input devices like the space mouse and spaceball that provide six degrees of freedom. The three major issues that make the design of a 3D user interface complicated are:

1. Picking.
2. Movement/Navigation.
3. Visualization.

3D Sketcher is also faced with these three issues. There are many ways to tackle these issues. The commercial software like Autocad2000, Houdini, 3dstudioviz and others all handle them differently. In 3D Sketcher we have tried to use the most intuitive way to tackle these issues.

Picking

Picking in 3D Sketcher is done using the JAVA 3D class PickMouseBehavior.

Movement/Navigation

The 2D mouse coordinates have to be mapped to 3D coordinates to be able to place or drag objects in a 3D scene. This is done using some mathematical manipulations of the 2D mouse coordinates. Different commercial software have their own way of doing the mapping from 2D coordinates to 3D coordinates. For instance in 3DstudioViz whenever an object is picked tiny coordinate axes appear on top of the object and by moving the mouse to the appropriate plane the user informs the UI that he wants to move the object in that plane. And thus the 2D mouse coordinates are interpreted in that plane. In AutoCAD2000 when an object is picked and dragged a vector is drawn in 3D by doing certain mathematical calculations and then when the mouse is released the object is moved to the end of the vector.

In Sketcher the user has three options. Either he can switch to top view and then move the object parallel to the X-Z plane or he can switch to the front view and then move the object parallel to the X-Y plane or he can work with any random rotation, R of the world and move the object along the plane of the screen. The first two methods are straightforward. In case of motion parallel to the X-Z plane the y coordinate of the object is not altered and the displacement in the x-y coordinates of the mouse are assume to be the displacement in the x-z coordinates of the object. Similarly in case of the motion parallel to the X-Y plane the z coordinate of the object is kept unaltered and the displacement in the x-y coordinates of the mouse is assumed to be the displacement in the x-y coordinates of the object. In case of the third method—movement along the plane of the screen the plane of the screen could be any random plane so the following calculations are used to manipulate the coordinates.

Assume camera is directly pointing at origin of world and the translation matrix of the world is R. Let us assume that a point p is picked with the mouse at location P(x, y). Now let us say that the mouse is dragged from point P(x, y) to P'(x', y') on the screen. Consider the displacement vector V = (x'-x, y'-y, 0) which is the displacement vector of the mouse assuming that it is moved on the X-Y plane. Now to get the actual displacement of the mouse in the translated and or rotated world we can apply the formula: V_actual = R^(-1) (V)+p. Now this actual displacement vector can be applied to the actual 3D coordinates of the object that is being dragged to get the new position of the object. In the same logic we can assume that the starting position of the point p is origin at every point.

For navigation the Sketcher allows the rotation of the whole world along with the coordinate axes. This can be done by clicking anywhere in the scene other than any of the objects and dragging the mouse.

Visualization

Perception of depth is very important for visualization. The X, Y and Z axes are displayed in the scene at all times. Along with the axes the X-Z plane is also displayed at all time. This helps the user to visualize all the objects in the scene with the proper relative spacing.

CHAPTER 5
SURVEY OF SIMILAR SYSTEMS

Strictly speaking, Sketcher cannot be compared in many aspects, with the other systems because it is a specific front-end geared to FRONTIER. This chapter is simply a survey of some other systems similar to FRONTIER, with limited comparisons on relevant functionalities.

Most of the existing systems lack the back-end functionalities of FRONTIER and hence try to abstract the solving process completely from the user. The 3 variational constraint solvers mentioned below (Erep, D-cubed, I-DEAS) are largely restricted to limited classes of constraint systems (even their 3d versions) while FRONTIER deals with fully general constraint systems. FRONTIER also offers various functionalities that Sketcher aims at delivering to the user. Sketcher attempts to make the system totally transparent so that the user can see the step-by-step solving of the input problem, steer the system if necessary and interact with the solver conveniently.

In the 2 dimensional constraint solver developed by Bouma et. al. [4] the communication between its graphical user interface and its constraint solver is very similar to the communication between the front-end and back-end of FRONTIER. FRONTIER's Sketcher communicates with the UTU using the Java Native Interface. The sketch is translated into textual data using an open representation based on clusters, called the frep (FRONTIER's representation) that is used by all the modules of the system. This representation language is useable by any other system that calls FRONTIER. In the system developed by Bouma et. al. [4], the GUI communicates with the constraints solver using Erep (Editable Representation). It then applies placement heuristics to the textual translation that the GUI produces. These rules are based on the assumption that the sketch is topologically accurate. So it makes observations such as which side of a line a specific point lies in the sketch. These observations are then used to decide where that point should lie in the final sketch. Once these rules fail they fall back on interaction with the user to amend it. FRONTIER's Sketcher on the other hand uses the topological information from the initial sketch only while displaying the partially solved sketch which has no bearing on the final solution. The solver fully relies on the user interaction for choosing one of the many possible solutions (in the Get-bifurcations mode) or picks the most reasonable bifurcation (in the Auto-solve mode) by itself. FRONTIER should ideally make use of the topological information, especially in the Auto-solve mode.

The D-cubed's 2D Dimensional Constraint Manager [5] is a 2D variational constraint solving engine. The user is allowed to specify and control the geometric models using rules that include the dimensions like distances, angles etc. and constraints like parallelism, tangency and concentricity. To modify a model, the user simply specifies a change to the rules, such as a modified value for a dimension. The 2D DCM then automatically re-calculates the locations of all the geometries affected by the rules still maintaining consistency with the previously applied dimensions and constraints. This is also used in profile sketching for 3D solid modeling applications. DCM's online solving works similar to FRONTIER's Simplesolver. But the performance of its online solving is comparatively very good. Although, since it relies on a collection of adhoc heuristics and solving methods it does not always guarantee a solution. But our emphasis is to approach the Simplesolver algorithms systematically, since they pose interesting theoretical problems. D-cubed has also recently released the 3D DCM which is a 3D variational constraint solving engine for assembly part positioning and kinematic simulations that works for a specific class of 3D constraint systems. Many other commercial systems also use variational constraint solvers like I-Deas. Pro/Engineer on the other hand uses parametric constraint solvers. Parametric constraint solving is a much easier constraint solving problem.

Some user interfaces also interpret gestures made by the user. Beat Bruderlin's Quicksketch [21] is a GUI that interprets the user's strokes to construct the object rather than picking objects from the menu like most other 3D computer modeling/sketching systems. The sketches thus drawn can then be refined by defining 2D/3D constraints on them. Quicksketch is a tool for pen-based computers. But Brown University's Sketch [24] uses a similar method for drawing and moving 3D objects on the screen using the mouse. Many of the commercial solid modeling and CAD systems infer geometric constraints and try to capture design intent by interpreting the user's mouse movements. Sketcher also draws such inferences while interpreting which sub-shape (of a shape) is involved in a constraint.

There is also a lot of research being done on constraint solvers that deal with algebraic equations. These solvers are used in graphical layout management, user interface designing, simulations etc. The series of algorithms built at University of Washington [22] – Deltablue, Skyblue, Indigo, Ultraviolet, and Cassowary all fall into this category. Though these constraint solvers can handle simple geometric constraint problems – e.g. in sketching programs – they are too general to perform comparably to geometric constraint solvers that are designed specifically to handle geometric constraint systems.

CHAPTER 6
FUTURE WORK

As asserted earlier also FRONTIER is a generic solver, so it can be easily extended to be used in different applications. There is always scope for new applications and additions to the existing system. Sketcher is flexible enough to evolve quickly to adapt to these changes. The key is to use this flexibility optimally to make the system more and more user-friendly for any application. In this section we discuss some of the tasks that should be done and some suggestions that might prove to be very useful for the current and any future application.

In the 2D input 3D output Sketcher as well as the 2D Sketcher the online solving should be made more natural. The simple solver should solve the simple constraints as they are added and once the simple solver fails the back-end should be invoked automatically.

FRONTIER is already capable of handling reasonably big 3D problems. Huge system is broken down into smaller sub-systems and then solved. Hence handling really huge systems will not be problem. Sketcher has already adapted itself greatly for the molecular structure and assembly modeling application. For instance the addition of the color and radius attributes to the point objects was done keeping the application in mind. But the adaptation process is ongoing and continuous. Right now the color and radius properties of the point objects and the type property of the distance constraint are not used by the back-end at all. This information may prove important for solving also, if the different elements (represented by points of different color or radii) need to be handled differently as far as the solving is concerned. Also the different chemical bonds (represented by different types of distance constraints) may have to be handled differently.

Sketcher is capable of rendering complex 3D structures. For instance the figure below shows a molecular structure rendered using 3D Sketcher.

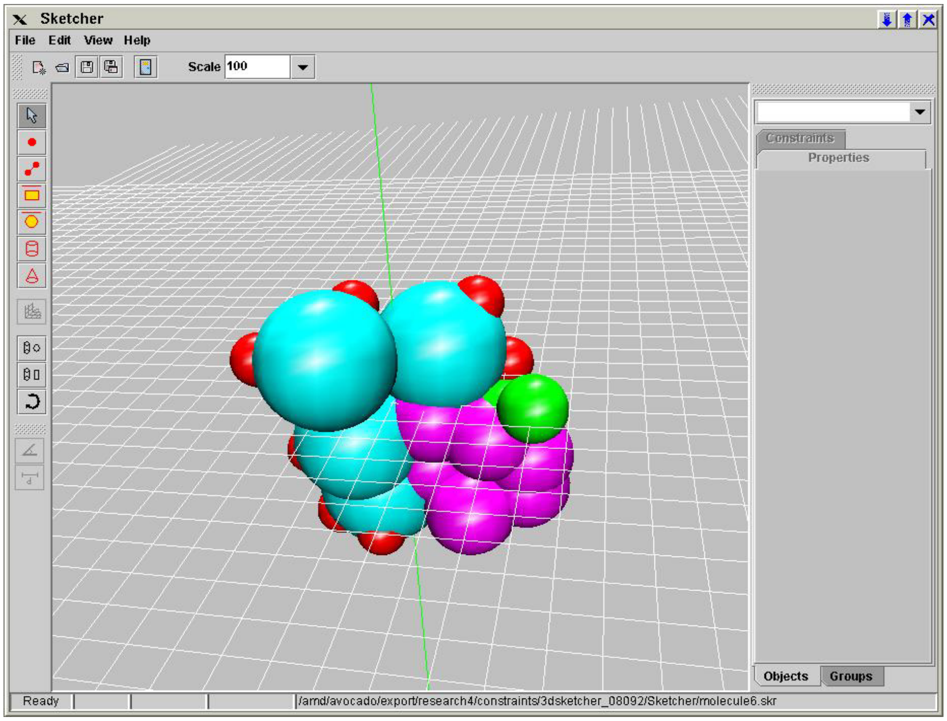


Figure 6-1 Molecular structure created using 3D Sketcher.

The 3D Sketcher can also be used to draw 3D sketches on a 3D palate. But the two immediate upgrades that need to be done are optimization of the 3D user interface and integration of the 3D user interface with the solver. Optimization of the 3D user interface involves certain implementation tasks and some conceptual modifications. As far as the implementation is concerned the most important task would be increasing the speed of rendering which is very important for a user who draws a sketch on a 3D palate. Then a repository for 3D objects needs to be added. There are a few other facilities that need to be added to make the user interface most intuitive. The user should be allowed to make a choice as to whether he wants to move the object on the plane of the screen or along the X, Y and Z axes. Navigation in the 3D world should be made easier.

Both in case of 2D examples and 3D examples it would be good to have a movie showing the system as it is being built (solved). For instance in case of 2D constraint systems if the user solves using the auto-solve mode then it would be good to show the user all the intermediate solutions continuously as a movie by recording all the bifurcations chosen by the solver. This may be very useful to the user. In case of 3D constraint systems if there is a facility to assemble all the pre-recorded bifurcation choices of the user into a single movie then we can simulate the whole virus assembly process.

LIST OF REFERENCES

[1] Alice: Free, Easy, Interactive 3D Graphics for the WWW, "Java3D Documentation," 2000, Accessible at http://www.alice.org/jalice/docs/Java3D/api/index.html, accessed 5 September 2001.

[2] Greg J. Badros, "Constraints in Interactive Graphical Applications," Ph.D. Dissertation, Department of Computer Science and Engineering, University of Washington, December 1998.

[3] Farid BenHajji, Erik Dybner, "3D Graphical User Interfaces," Master's Thesis, Department of Computer and Systems Sciences, Stockholm University and the Royal Institute of Technology, October 1999.

[4] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, R. Paige, "Geometric Constraint Solver," *Computer Aided Design,* Vol. 27, No. 6, 1995, pp. 487-501.

[5] D-Cubed, "Products 2D & 3D DCM," 2000, Accessible at http://www.d-cubed.co.uk/prod_dcm_intro.htm, accessed 1 March 2002.

[6] C. Hoffmann, J. Rossignac, "A Road Map to Solid Modeling," *IEEE Transactions on Visualization and Computer Graphics,* Vol. 2, No. 1, 1996, pp. 3-10.

[7] C. Hoffman, A. Lomonosov, M. Sitharam, "Planning Geometric Constraint Decompositions via Graph Transformations," *Proceedings of AGTIVE '99 Graph Transformations with Industrial Relevance* (Kerkrade, Netherlands, September 1-3, 1999), Nagl, Schurr and Munch (eds), Springer LNCS 1779, 1999, pp. 309-324.

[8] C. Hoffman, A. Lomonosov, M. Sitharam, "Decomposition of Geometric Constraints Part I: Performance Measures," *Journal of Symbolic Computation,* Vol. 31, No. 4, April 2001, pp. 367-408.

[9] C. Hoffman, A. Lomonosov, M. Sitharam, "Decomposition of Geometric Constraints Part II: New Algorithms," *Journal of Symbolic Computation,* Vol. 31, No. 4, April 2001, pp. 409-428.

[10] C. Hoffmann, Jorg Peters, "Geometric Constraints for CAGD," *Mathematical Methods in Computer-Aided Geometric Design III,* M. Dahlen, T. Lyche, and L. L. Schumaker (eds), Vanderbilt Press, Nashville, 1995, pp. 237-254.

[11] W. Hower, W. H. Graf, “Research in Constraint-Based Layout, Visualization, CAD, and Related Topics: A Bibliographical Survey,” *Proceedings of the International Workshop on Constraints for Graphics and V*isualization *(CGV '95) in conjunction with the International Conference on Principles and Practice of Constraint Programming (CP95)* (Cassis, France, September 1995), I. F. Cruz, K. Mariott, and P. Van Hentenryck (eds), 1995, pp. 3-23.

[12] The Java 3D Community Site (J3D), “Java 3D FAQ,” 1999, Accessible at http://www.j3d.org/faq/, accessed 3 September 2001.

[13] G.A.Kramer, “Solving Geometric Constraint Systems,” *Proceedings of the 8th National Conference on Artificial Intelligence* (Boston, Massachusetts, July 29 - August 3, 1990), MIT Press, 1990, pp. 708-714.

[14] G. Leach, G. Al-Qaimari, M. Grieve, N. Jinks, C. McKay, “Elements of a Three-Dimensional Graphical User Interface,” March 1997, Accessible at http://goanna.cs.rmit.edu.au/~gl/research/HCC/interact97.html, accessed 1 September 2001.

[15] J-J Oung, “Design and Implementation of an Object-Oriented Geometric Constraint Solver,” Master’s Thesis, University of Florida, 2001.

[16] J-J. Oung, M. Sitharam, B. Moro, A. Arbree, “FRONTIER: Fully Enabling Geometric Constraints for Feature Based Modeling and Assembly,” *Proceedings of ACM Solid Modeling Symposium* (Ann Arbor, Michigan, 2001), ACM Press, 2001, pp. 307-308.

[17] B. Roth, W. Whiteley, “Tensegrity Frameworks,” *Transactions of the American Mathematical Society*, Vol. 265, No. 2, 1981, pp. 419-446.

[18] Meera Sitharam, Jian-Jun Oung, Adam Arbree, Nandhini Kohareswaran, “FRONTIER a General Geometric Constraint Solver Part I: Functionality,” in preparation, Department of Computer and Information Science and Engineering, University of Florida, Gainesville.

[19] Meera Sitharam, Jian-Jun Oung, Adam Arbree, Nandhini Kohareswaran, “FRONTIER a General Geometric Constraint Solver Part II: Architecture,” in preparation, Department of Computer and Information Science and Engineering, University of Florida, Gainesville.

[20] I. E. Sutherland, “Sketchpad: A Man-Machine Graphical Communication System,” Ph.D. Dissertation, MIT, 1963.

[21] Technical University of Ilmenau, Computer Graphics Group, Department of Computer Science and Automation, “Research: Sketch-Based Design,” 2001, Accessible at http://rabbit.prakinf.tu-ilmenau.de/qsketch.html, accessed 28 August 2002.

[22] University of Washington, Department of Computer Science and Engineering, “Constraint-Based Systems,” 1999, Accessible at http://www.cs.washington.edu/research/constraints/, accessed 24 March 2002.

[23] B. de Vries, A. J. Jessurun, R. H. M. C. Kelleners, “Using 3D Geometric Constraints in Architectural Design Support Systems,” *Proceedings of the 8-th International Conference in Central Europa on Computer Graphics, Visualization and Interactive Digital Media 2000* (University of West Bohemia , Czech Republic, February 7-10, 2000), WSCG, University of West Bohemia, 2000.

[24] R. C. Zeleznik, K. P. Herndon, J. F. Hughes, “SKETCH: An Interface for Sketching 3D Scenes,” *Proceedings of SIGGRAPH '96* (New Orleans, August 1996), ACM Press, 1996, pp. 163-170.

BIOGRAPHICAL SKETCH

Naganandhini Kohareswaran is originally from Chennai, India. She obtained her Bachelor of Engineering degree from the University of Mumbai (V.E.S.I.T) in 2000. She came to the US in August of 2000 to pursue higher studies in computer science. She received her Master of Science degree from the Department of Computer and Information Sciences and Engineering at the University of Florida, in December of 2002.