Geometric Sketch Constraint Solving with User Feedback

Bridget M. Dixon* and John F. Dannenhoffer III†

Syracuse University, Syracuse, New York, 13244

Modern Computer Aided Design systems rely on two-dimensional sketches as the basis for the creation of most three-dimensional geometry. These sketches consist of qualitative shape information as well as a series of constraints to quantify the exact size and shape of the various segments, arcs, and splines that comprise the sketch. Example constraints include overall lengths, symmetries, parallel and/or perpendicular segments, and arc radii. For a given sketch, there are many sets of constraints that unambiguously define the shape. Unfortunately, sometimes a user prescribes a set of constraints that do not allow the sketch to be solved. Providing feedback to the user as to how to change the constraint definitions to remedy the problem is still a largely-unsolved problem. Herein is described a series of techniques for helping a user diagnose sketch problems and provide to the user a series of options that will cause the sketch to be properly solved. The techniques are incorporated in the OpenCSM solid modeling system. Several sample configurations that use these techniques are shown, with a description of the user interaction that the system provides.

I. Introduction

In most Computer-Aided Design (CAD) systems, the 2D sketch is the fundamental element that allows generation of three dimensional models. Sketches are commonly created by starting with general shapes and then adding constraints that uniquely define an intended sketch. Examples of sketch constraints are lengths of segments, radii of arcs, intersection angles, parallel segments, etc. In general, the user needs to prescribe as many constraints as there are free variables associated with the sketch's line segments, arcs, and splines. The constraint solver then finds the dimensions that satisfy the constraints.

The idea of sketching has its roots in the first graphic system, which was developed in the mid 1950s at Massachusetts Institute of Technology in the Lincoln Labs.[1] In 1960, the first step was taken for what became known as the CAD industry by a project called SKETCHPAD that was developed in the same MIT laboratory. Over time, the concept of sketching became strongly linked with the idea of "parametrics".[2] Since the introduction of parametrics nearly 25 years ago, most sketcher advances have focused on integrating parametrics while little attention has been given to helping the user understand how to properly constrain sketches.

Meanwhile, OpenCSM is a software system under development that, among many other features, incorporates geometry and sketch solving for multi-disciplinary analysis and optimization platforms.[3]

This paper describes advances in user-directed constraint solving in OpenCSM. But even though the sketcher advances described here were initially targeted at OpenCSM, they can be adapted for use with other CAD systems as well.

II. Method of Solving Sketches

A sketch is a two-dimension closed figure composed of line segments, arcs, and splines. Fig. 1 shows a very simple sketch composed of three line segments and one arc. Superimposed on the figure are several letters and numbers that describe the constraints that a user may have specified in order to "solve" the sketch.

*PhD Student, Mechanical & Aerospace Engineering, AIAA Student Member.

†Associate Professor, Mechanical & Aerospace Engineering, AIAA Associate Fellow.

Table 1. Legend for plot labels in OpenCSM

Labels for Constraints			
X	TempFix x	l	length
Y	TempFix y	lr	length relation
C	TempFix C	h	horizontal
x	fixed x	v	vertical
y	fixed y	i	inclination angle
p	perpendicular	r	radius arc
t	tangent	T	turning angle of arc
c	co-linear points	c	co-linear center
d	distance relation		

Table 1 defines the meaning of the letters in the sketch. The "TempFix" constraints in Table 1 are temporary constraints that are added by the solver when diagnosing a sketch; they are discussed further in the next section. One may notice that co-linear points and co-linear centers are both designated with a "c" on the plot; the difference is that co-linear points puts the "c" at a vertex and co-linear centers puts the "c" at the center of an arc. The co-linear, distance relation, and length relation constraints are explained and demonstrated in later examples.

Since almost all of the constraints reference a vertex number, every other vertex number is labeled on the plot as shown in Fig. 1. Note that all of the line segments and arcs are labeled using its starting vertex, which is determined by going counterclockwise around the figure.

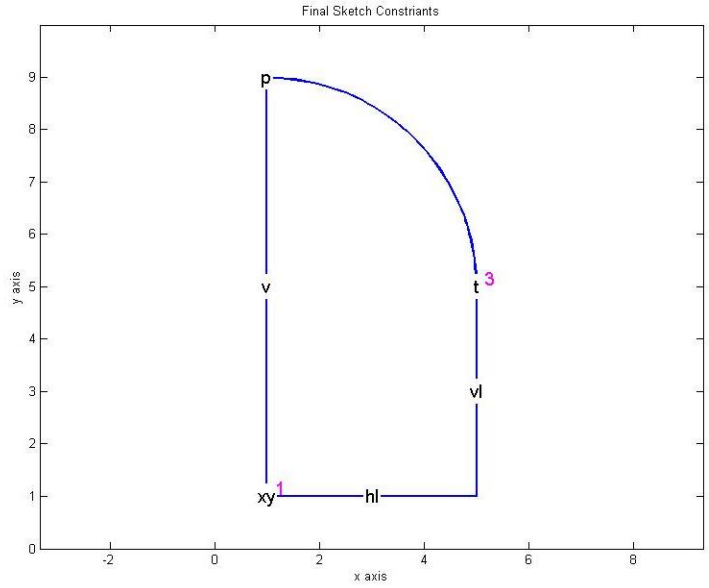

Figure 1. Simple sketch to demonstrate problem setup

According to Hoffman and Vermeer, there are four basic methods for solving a system of constraints: numerically, commonly known as the Newton-Raphson method; symbolically; with a graph-based approach; or using logical inference and term rewriting.[4] The numerical method, while iterative, is the the most common method used due to it's versatility and is the basis for the technique described herein.

The Newton-Raphson method is set up to drive a set of residuals, R, (each of which is derived from a user-specified constraint) to zero by adjusting the sketch's free variables. For the sketch in Fig. 1, the variables are the x and y vertex locations as well as the arc curvature C.

$$x^T = \begin{bmatrix} x_1 & y_1 & x_2 & y_2 & x_3 & y_3 & x_4 & y_4 & C3 \end{bmatrix} \tag{1}$$

The number of variables in the system dictates the number of defining constraint equations that are necessary to uniquely solve the sketch and locate the vertex coordinates. This example has nine variables and therefore nine residual equations are necessary to define the sketch.

Table 2. Residual Equations to Define Example

Residual Equations	
$R_1 = y_2 - y_1$	(horizontal)
$R_2 = x_4 - x_1$	(vertical)
$R_3 = x_3 - x_2$	(vertical)
$R_4 = x_1 - 1$	(fixed at 1)
$R_5 = y_1 - 1$	(fixed at 1)
$R_6 = \texttt{tangent}(x_2, y_2, 0, x_3, y_3, C3, x_4, y_4)$	(tangent)
$R_7 = \sqrt{(y_2 - y_3)^2 + (x_2 - x_3)^2} - 4$	(length is 4)
$R_8 = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2} - 4$	(length is 4)
$R_9 = \texttt{tangent}(x_3, y_3, C3, x_4, y_4, 0, x_1, y_1) - \frac{\pi}{2}$	(Perpendicular)

The residual equations in Table 2 are all arranged so that when the sketch is solved and the constraints are satisfied the residuals will vanish. Note that the `tangent` function is a function defined in OpenCSM, where given the vertices and the curvatures of adjacent segments, it returns the dihedral angle at the intermediate vertex. If `tangent` returns zero, the segments are tangent at this vertex; if $\pm\frac{\pi}{2}$ is returned, the segments are perpendicular.

The Newton method uses a Jacobian matrix that is defined by $\frac{\partial R_n}{\partial x_n}$. The problem is set up using a matrix system as shown in Eq. (2).

$$\begin{bmatrix} \frac{\partial R_1}{\partial x_1} & \frac{\partial R_1}{\partial x_2} & \cdots & \frac{\partial R_1}{\partial x_n} \\ \frac{\partial R_2}{\partial x_1} & \frac{\partial R_2}{\partial x_2} & \cdots & \frac{\partial R_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial R_n}{\partial x_1} & \frac{\partial R_n}{\partial x_2} & \cdots & \frac{\partial R_n}{\partial x_n} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_n \end{bmatrix} = - \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_n \end{bmatrix} \tag{2}$$

The individual entries in the Jacobian matrix are solved by linearizing about a point (the initial or current guess) and applying finite difference expressions. Direct evaluation of the derivatives by symbolic differentiation of the constraint formulae is possible, but has been found to be of little benefit in the current work. To solve the system of equations using an iterative method, the Δx vector is updated with each iteration, driving the R vector to zero. The iterations stop and the matrix is considered solved when the maximum value in the R vector is less than the given tolerance, which is 10^{-12} in the OpenCSM sketcher.

The advantage of the matrix setup in Eq. (2) is that it allows for any problem to be solved if defined properly. For the current example problem, the matrix is well defined and solves to generate the sketch shown in Fig. 1 with the constraints appropriately labeled.

The numerical method is used to find sketch solutions because, given a well defined system and quality initial guess, the method will always find a solution. The chief disadvantages to this system are its dependence on initial conditions and the fact that an iterative method can be time consuming. Despite these disadvantages, the numerical approach is used due to its ability to solve any well defined system, no matter how large.

III. Diagnosing Sketch Problems

For further demonstrations of commercial CAD software and the OpenCSM sketcher, the sketch shown in Fig. 2 will be used. This sketch was specifically designed to test different sketchers and demonstrate concepts.

This sketch is defined by eight vertices and the curvature of three arcs making a total of 19 variables as shown in Eq. (3).

$$x^T = [x_1\ y_1\ x_2\ y_2\ x_3\ y_3\ x_4\ y_4\ x_5\ y_5\ x_6\ y_6\ x_7\ y_7\ x_8\ y_8\ C1\ C2\ C7] \tag{3}$$

This sketch uses length relations which are constraints used to relate various lengths. In this test sketch a "length relation" is used to define that segment 4-5 is three times the length of segment 6-7. The label appears only on the first segment in the relation as shown (later) in Fig. 5.

The fully defined system results in the sketch shown in Fig. 2.

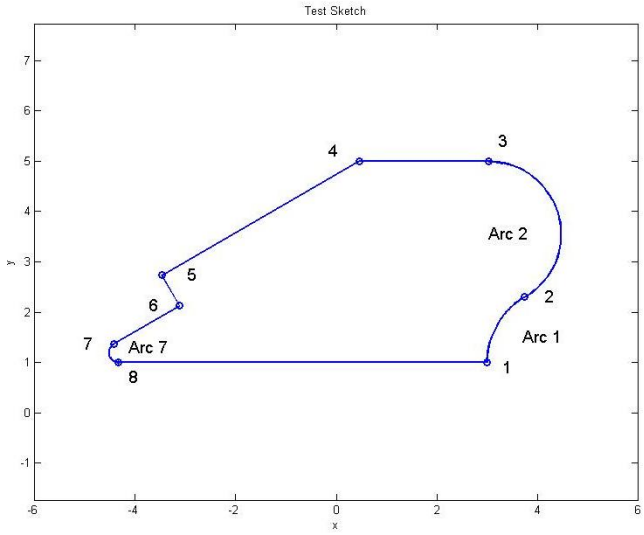

Figure 2. Custom sketch designed to test system features with segments labeled for reference.

The constraints used to define this sketch are given in Table 3. While Fig. 2 may not seem complicated, there are many different ways to define the constraints for the sketch to get a fully constrained system, which is the ultimate goal. The constraints shown in the table are just one of various ways to fully define the desired sketch, which will be explored in detail later.

Table 3. Constraints to define the sketch in Fig. 2 with Jacobian matrix row locations

Constraints	
Row 1: Fixed Point x at 1	Row 11: Horizontal 3-4
Row 2: Fixed Point y at 1	Row 12: Overall Length
Row 3: Horizontal 8-1	Row 13: Overall Height
Row 4: Perpendicular at 1	Row 14: Length of segment 5-6
Row 5: Tangent at 2	Row 15: Length Relation of segments 4-5 & 6-7
Row 6: Tangent at 3	Row 16: Turning Angle of Arc 1
Row 7: Perpendicular at 5	Row 17: Inclination Angle of 6-7
Row 8: Perpendicular at 6	Row 18: Length of Radius Arc 7
Row 9: Tangent at 7	Row 19: Length of Radius Arc 1
Row 10: Tangent at 8	

A sketch can be classified into one of four different states; fully defined, over-constrained, under-constrained or improperly constrained. A fully defined system has the proper number of constraints and is solvable, as in no singularities exist, resulting in one unambiguous solution for the sketch. An over-constrained system has too many constraints and those constraints may or may not properly define a sketch. An under-constrained system does not have enough constraints to uniquely define the sketch and constraints must be added. An

improperly constrained system has the correct number of constraints but there may be conflicting constraints that cause a singularity, and certain coordinates to be unsolvable. Each of these cases is described below.

III.A. Case 1: Under-Constrained

Most CAD systems will identify the state of the constraints on a given sketch. The commercial systems also have limitations for the types of constraints that can be added; therefore there is a slight difference in how the sketch is defined in ProEngineer[5] and how it was previously defined for OpenCSM. Overall length and height constraints are not available in this commercial package. Also, to use a length relation one of the lengths must be defined, unlike what was used in the OpenCSM sketch solver.

When identifying the state of the constraint system, ProEngineer takes a unique approach. ProEngineer actually prevents a sketch from ever begin under-constrained due to its intent manager. As objects (lines, arcs, etc.) are added to a sketch, "weak" constraints are added, which are purely dimensional constraints, creating and maintaining a fully defined sketch. The user adds constraints to the system to create the desired object and as constraints are added the weak auto-added constraints are removed. While this keeps the system in a solvable state for a complex sketch, the dimensions added are typically not what the user would actually want to define and there is no feedback as to what other constraints could be added. Using our test shape in ProEngineer, the fully defined sketch looks like that in Fig. 3. When we remove the length of segment 8-1 (the bottom horizontal segment) a new dimension is added to the shape shown in Fig. 4.

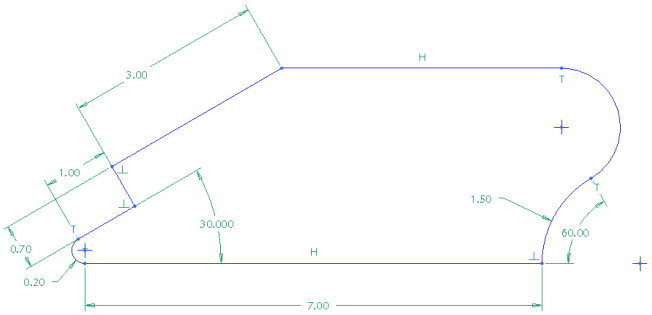

Figure 3. Fully defined sketch in ProEngineer with all user defined constraints.

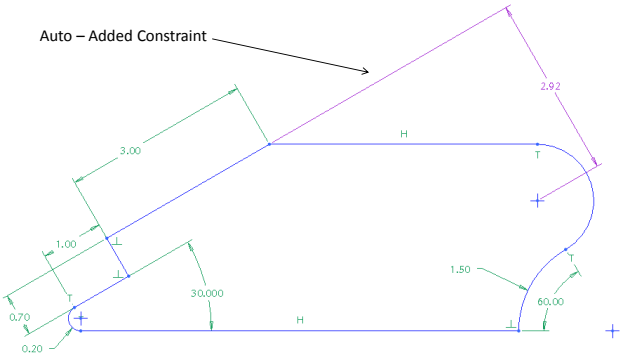

Figure 4. Fully defined sketch in ProEngineer when length constraint is removed, "weak" constraint added by the system.

In OpenCSM's sketcher, the sketch is diagnosed and fixed based upon user feedback rather than automatically patched as in ProEngineer. The system first identifies where the singularity is located by row reducing the Jacobian matrix. For a well defined system the row reduction will yield the identity matrix, but for under-constrained systems there will be at least one zero column. The location of the singularity, identified by a zero column, shows which parameter is undefined.

The OpenCSM sketcher temporarily adds a fixed point for the missing parameter, and checks the matrix again. This iterative approach is used since it is possible, and common, to have multiple problems in a

system of constraints. If there were only one problem with the matrix, the temporary constraints would not be necessary. Once all of the singularities have been replaced with temporarily fixed points, the solver checks all of the constraints in Table 4, one-by-one for replacements that would be meaningful to the user.

Table 4. Constraints Options to Add to a sketch

Vertex Constraints	Segment Constraints
Fixed x	Length
Fixed y	Horizontal
Perpendicular	Vertical
Tangent	Radius
	Turning Angle

Some of these constraints require values, such as length, radius and turning angle; for these the initial guess from the user-drawn sketch is used as the value for the constraint. For perpendicularity, the constraint may be for either convex or concave intersections. In all of these instances, the value automatically generated that used to validate the constraint is presented to the user and then the user is asked to provide a value. The internal value used by the sketcher is only offering guidance to the user to help choose an appropriate value. If the user enters a value that is too different from what the sketcher tested with, it is possible the matrix will develop a singularity and cause the Newton solver to diverge; see the Implementation Issues section for more details.

To demonstrate the feedback from the sketcher, two constraints were removed from those in Table 3. The constraints removed were perpendicularly at vertex 6 (the inside corner near the left of the figure) and horizontal segment 3-4 (at the top). Removing these constraints results in an under-defined sketch that cannot solve for the location of y_6 and the curvature $C2$ as shown in Fig. 5.

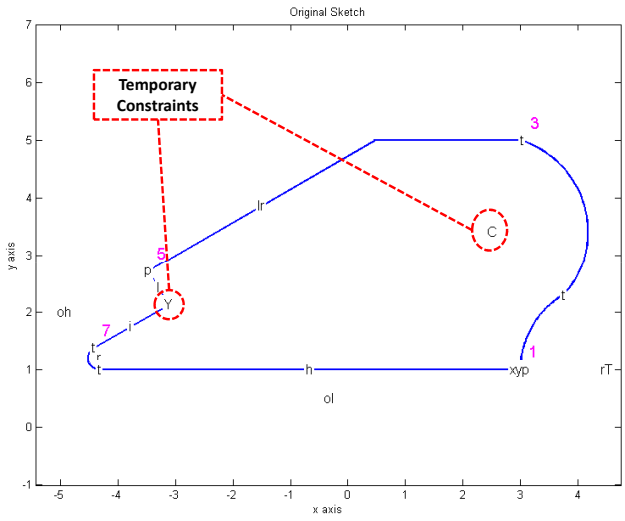

Figure 5. Temporary Constraints added the define the Sketch

After these temporary constraints are added, all of the possible options to replace the temporarily fixed y_6 are presented to the user to choose which one to add to the system. There are nine possible options for this case which are shown in Fig. 6.

For this example, the user selects to add concave perpendicularity at vertex 6, which is the constraint that was removed from the original fully defined system; any of the nine options presented to the user would allow the sketch to be solved. Since two constraints were removed from the system, there is still another temporary constraint that needs to be replaced with one that is more in line with the user's design intent.

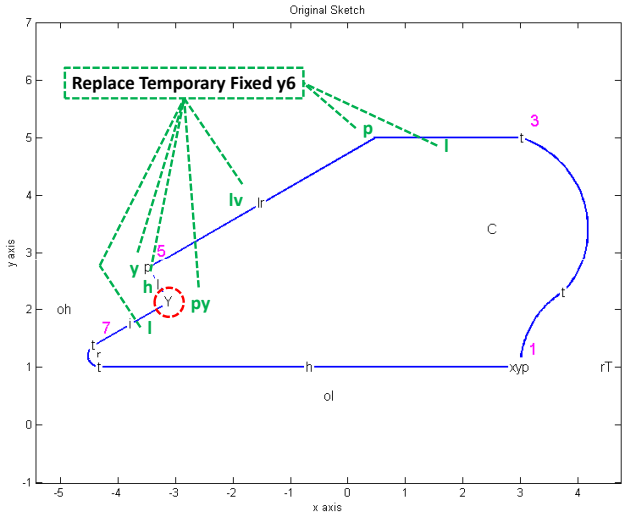

Figure 6. Constraint replacement options for the temporary fixed y_6.

There are 15 options to replace the temporarily fixed $C2$ point; they are shown in Fig. 7.

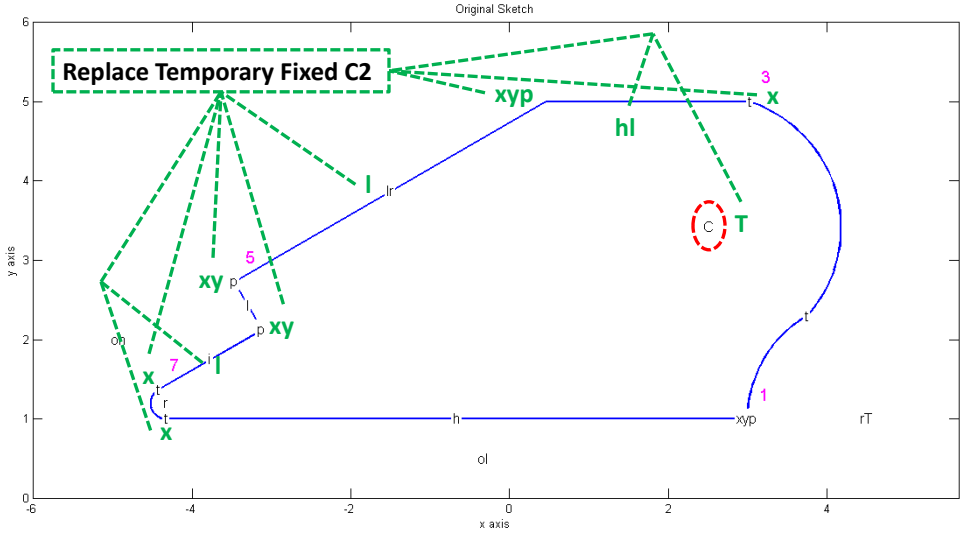

Figure 7. Constraint replacement options for the temporary fixed $C2$.

Choosing the option that segment 3-4 is horizontal will result in the same set of constraints that were originally used and the same figure. As before, any of the 15 options would result in a fully defined sketch.

The advantage of this system is that constraints added are temporary and merely a step to identify all the problems with the current sketch system and are replaced with constraints of the user's choice. The processes of replacing the temporary constraints for an under-defined sketch is the same process that is used to fix an improperly constrained system (to be described later).

III.B. Case 2: Over-Constrained

Over-constrained sketches are the only state that all commercial CAD systems tested acknowledge and offer some advice to repair. In AutoCAD when adding a new constraint that will over-constrain the system, a

window appears and informs the user that adding that constraint will over-constrain the system and therefore it cannot be added as shown in Fig. 8. AutoCAD offers no assistance in how to fix the problem, but only informs the user of that a problem exists.

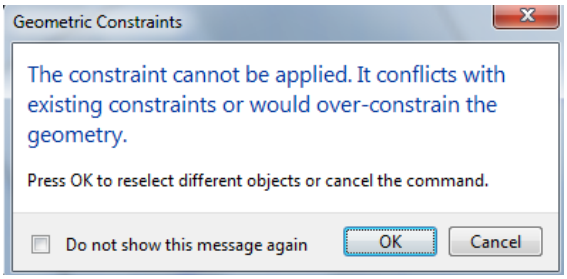

Figure 8. Feedback provided to the user by AutoCAD for an over constrained system.

From the message shown, AutoCAD does not distinguish between improper and over-constrained systems. This could make a fully defined sketch difficult to obtain if it is not apparent what all of the constraints should be. The other challenge is that AutoCAD does not allow certain constraints (such as a perpendicularity constraint at vertex 1 in Fig. 2). To bypass this, the next logical constraint would be setting the center of arc 1 to be co-linear with segment 8-1. The co-linearity constraint, in conjunction with the other constraints, would maintain a perpendicular intersection at that vertex. Regretfully, that is not an option in AutoCAD either, demonstrating a situation where offering suggestions for how to fully constrain a sketch would be helpful to the user.

ProEngineer provides a little more information, posting a window with choices that identifies the constraints that would conflict with the one being added and allows the user to choose which one to remove as shown in Fig. 9.

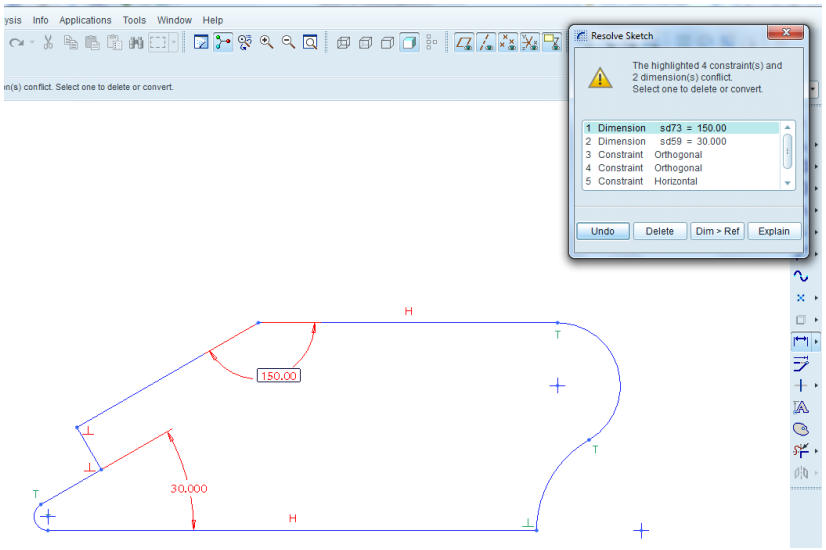

Figure 9. Feedback provided to the user by ProEngineer for an over constrained system.

Fixing a sketch to be better constrained in ProEngineer is certainly more user friendly; however this still does not help for all situations. Occasionally every constraint is presented as an option and that may not actually be helpful for the user. SolidWorks[6] looks at an over-constrained sketch and offers either a manual repair or a diagnostic tool. The manual repair is similar to ProEngineer in that it simply offers a list of the conflicting constraints. The diagnostic tool is more complex and takes time to use. The diagnostic tool removes conflicting constraints and offers solutions to fix the sketch to be properly constrained. This is also

one of the functions of the OpenCSM sketcher developed and presented here. Fig. 10 shows an example of the feedback provided to the user.

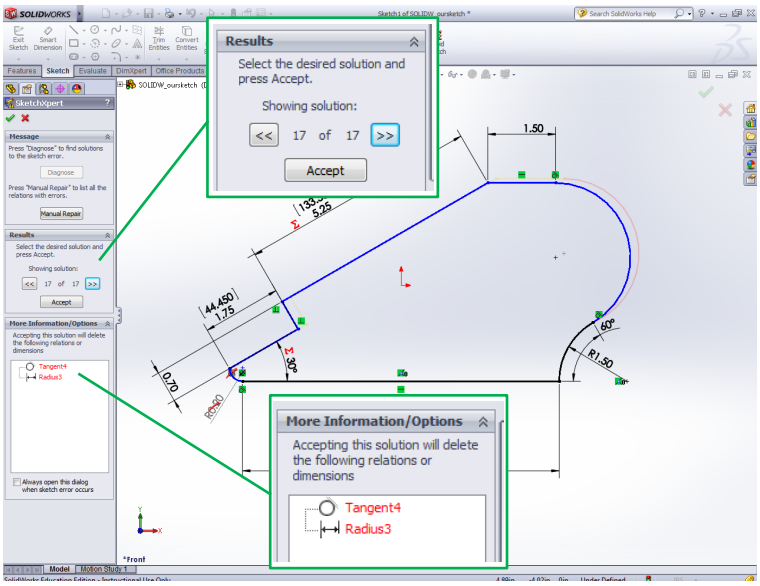

Figure 10. Feedback provided to the user by SolidWorks for an over constrained system.

The diagnostic tool in SolidWorks presents all of the options to fix the sketch, and after making a selection, the sketch is fully defined. Note, that while in the diagnostic tool the sketch is under-defined until the user makes a selection. The OpenCSM sketcher developed takes this feedback one step further by ranking the solutions based on the sensitivity of the sketch and presents the options that would generate the least difference in the final sketch from the original guess, as discussed further in the sensitivity section.

The sketch solver identifies redundant constraint options and presents them to the user by highlighting the constraints in red on the figure. The user can then choose which constraints to remove from the system to produce a fully defined system. The next section uses an example that demonstrates the over-constrained feedback.

III.C. Case 3: Improperly Constrained

Diagnosing an improperly constrained system uses a combination of the under- and over-constrained methods. First the coordinates that are undetermined need to be identified, which is done in the same manner as for an under-constrained problem. To add a temporary constraint to the problem the sketch becomes over-constrained and the temporary constraint must replace one of the current constraints in the system. Identifying where the temporary constraint can be added is executed in the same way as for an over-constrained system. Once the system of constraints has the proper number of constraints and all coordinates are able to be determined, the sketch solver can start the process of replacing the temporary constraints as before.

Neither ProEngineer, AutoCAD nor SolidWorks provided any feedback for an improperly constrained system other than a dialog box that would not allow a conflicting constraint to be added. The problem with this is that occasionally some constraints are more important than others and constraints are not necessarily added by the user in order of importance. In commercial CAD systems, without information about conflicting constraints, a user can either try to figure out where problem is or start over and hope for better results.

Using the sketch shown in Fig. 2 and modifying the constraints to get an improperly constrained problem the sketch solver can be demonstrated. Replacing the perpendicularity constraint at vertex 6 (the inside corner near the left of the figure) with a set value for the radius of arc 2 (the larger top arc on the right) creates redundant information. The y_6 location cannot be determine with this system of equations, so a temporary constraint has to be added. However, an improperly constrained system has the proper number

of constraints. In order to add a temporary constraint to the system of equations, an equation has to be removed. Fig. 11 shows the options for constraints that could be replaced with a temporarily fixed point.

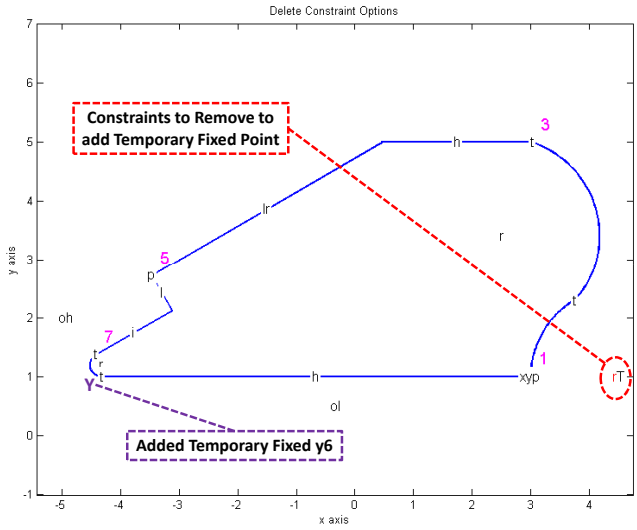

Figure 11. Replaced constraint highlighted in red with a temporary constraint to fix y_6.

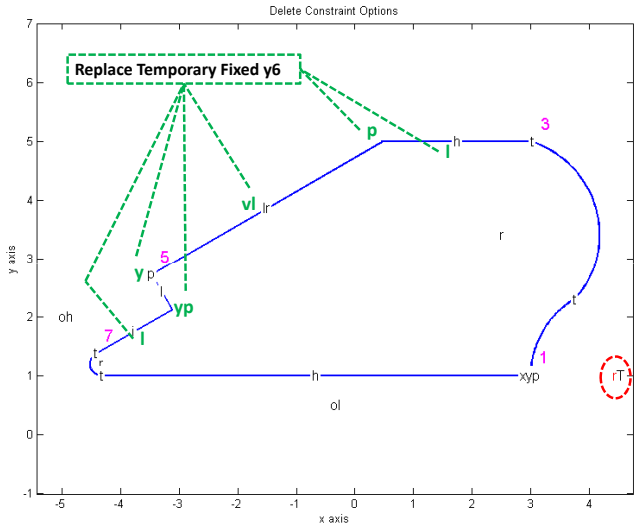

Figure 12. Options to replace the temporary fixed y_6 constraint.

For this particular situation, only one option was available to remove constraints. Once all of the necessary temporary fixed point constraints are added to the system, the sketch solver looks for options to replace the temporary constraint as before, shown in Fig. 12. The current set up results in eight options to replace the temporary constraint.

Any of these can be picked to get a fully-constrained system. The options are presented to the user as an ordered list, with the options that cause the least deviation from the original guess listed first.

IV. Examples and Implementation issues

The sketch defined in Fig. 2 has been used to demonstrate the various types of feedback that the sketch solver will provide in different situations. The sketch solver has also been tested using more complicated, real-world sketches. The sketch in Fig. 13 is an adaptation of a part from Engineering Graphics[7] and reproduced using the OpenCSM sketcher.

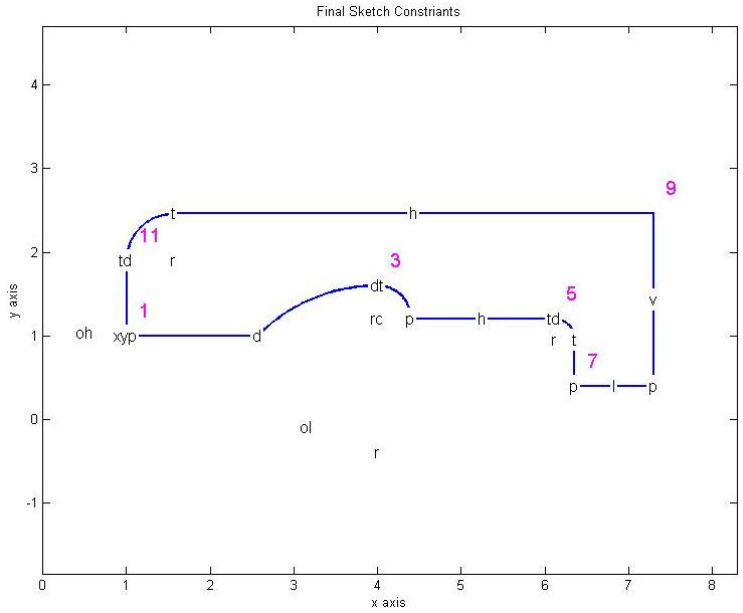

Figure 13. Engineering part example.

Not all sketch constraints were clearly identified in the engineering drawing, and the solver was used to help determine a few of the final constraints that were necessary to fully define the system.

This sketch also provides an opportunity to discuss what a "distance relation" is, especially since it is commonly used when describing mechanical parts. A distance relation is a constraint from a vertex to a line. For example, in Fig. 13 vertex 3 is 3.30 inches from the right hand side of the figure. This is labeled with a "d" at the vertex. Similar to a length relation, only the first vertex in the relation is labeled.

The biggest challenge with this new system was with the singularity issue. Ultimately the state of the sketch is dependent on the Jacobian matrix that results from the constraints and initial or current guess. A singular Jacobian matrix is an indication of one of two problems: either a bad guess or improper constraint equations. Initial guesses are obtained when the user draws the figure on the screen by placing points in space. Therefore a reasonable initial guess is obtained for the vertex locations. When adding constraints to the system, the initial guess may create a problem that is worked around by choosing intelligent constraints. The main function of the sketcher is that it identifies and diagnoses improper constraint equations that cause singularities as discussed in the previous sections.

At first, one might predict that if a matrix has a singularity it will be apparent when the matrix is initially set up. However, due to the iterative process inherent in Newton's method, a singularity may "appear" after a few iterations. For example, the initial guess may have a difference in the y values at each end of a horizontal segment, and as the sketch is solved, this difference gets smaller as the horizontal constraint is satisfied. If there is also a length constraint on that same segment it is possible that the y coordinate at either end of the segment could be become undefined as the horizontal constraint is satisfied.

This same problem is considered when replacing temporary constraints. If the sketch solver is trying to determine a y value and the segment is nearly horizontal, the length of the segment will result in a singularity after a few iterations. This is due to the fact that the length of a horizontal segment is only dependent on the x values at each end of the segment. That would be a poor constraint choice, so the sketch solver does not test length under these circumstances.

The problem of singularities "appearing" in a matrix as some of the constraints are satisfied makes simply looking at the Jacobian not enough to determine if adding that constraint would be a reasonable option.

To prevent options that would present a problem in the final sketch, the matrix is fully solved with each constraint test in order to determine the validity of a constraint being tested.

The capabilities of the sketcher in OpenCSM are vast and provide valuable feedback to the user to help create and maintain a fully defined sketch. As mentioned before, checking all of the constraint options can be a time consuming process. In addition to only checking reasonable options that would not produce a singularity, intuitively bad constraints are also skipped. For example, length of a segment is checked for linear segments, but if the segment is an arc, length is skipped. Similarly, radius and turning angle constraints are only checked for arcs since those wouldn't apply to a linear segment.

V. Sensitivities

In addition to finding and correcting singularity issues, there is also the matter of sensitivities. Occasionally during the design process a user may want to know more than just what the solution is. The sensitivity of each point to a perturbation in one of the constraints can be valuable information when trying to optimize a shape to meet some other engineering goal. For example, looking again at the test sketch in Fig. 2 the user may want to slightly change the overall length to meet an aerodynamic goal. When one constraint is changed there is an effect on the entire system and some points are affected more than others.

Understanding how each vertex is impacted by the different constraints can give the designer valuable information about where changes can be made with minimal impact in other areas. A sensitivity matrix can be simply determined using finite differencing as shown in Eq. (4).

$$\begin{bmatrix} \frac{\partial x_1}{\partial R_1} & \frac{\partial x_1}{\partial R_2} & \cdots & \frac{\partial x_1}{\partial R_n} \\ \frac{\partial x_2}{\partial R_1} & \frac{\partial x_2}{\partial R_2} & \cdots & \frac{\partial x_2}{\partial R_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_n}{\partial R_1} & \frac{\partial x_n}{\partial R_2} & \cdots & \frac{\partial x_n}{\partial R_n} \end{bmatrix} \begin{bmatrix} \Delta R_1 \\ \Delta R_2 \\ \vdots \\ \Delta R_n \end{bmatrix} = \begin{bmatrix} \Delta x_x \\ \Delta x_2 \\ \vdots \\ \Delta x_n \end{bmatrix} \tag{4}$$

$$\frac{\partial x_2}{\partial L} = \frac{x_2(9.02) - x_2(9.00)}{9.02 - 9.00} \tag{5}$$

Inspection of individual elements of this matrix can clarify how the sensitivity matrix is formed. For example, in Eq. (5) L is the overall length which is 9 units and then perturbed to be 9.02 units to find the sensitivity of the x coordinate at the second vertex to the overall length constraint. Each element of the sensitivity matrix is found using the same process. The transpose of the full sensitivity matrix is shown in Table 5 where each constraint equation was perturbed by $\frac{2\pi}{180}$. The rows of the transposed matrix represent the constraint equations as described in Table 3 and the columns are the x and y locations of each point and then the three arc curvatures.

From the sensitivity matrix the impact that each constraint has on each defining coordinate of the system is clear. For example, looking at the first row of the transposed sensitivity matrix we can see that only the x coordinates are affected by a perturbation in the first constraint (fixed x_1). This observation is both logical and trivial. However the last constraint (fixed radius of arc 1) has a little or no impact on the x coordinates on the other side of the sketch, which is not intuitively obvious but is easily apparent via the sensitivity matrix.

The sensitivity information is used within the sketcher to rank order the solutions by putting the solutions that keep all vertices closest to the initial guess first. For a complicated sketch it may not be apparent which constraint replacement would be the best for the system and which will be the closest to the user's intent, so rank ordering the solutions can help a user decide which constraint to add.

VI. Summary

Sketches are the building blocks of modern geometry generation techniques and understanding how to properly constrain them has proven challenging. Providing users with information about where a sketch has constraint problems will help users to build sketches in a more efficient manner. Understanding sensitivities can help engineers make more informed optimization decisions regarding geometry and constraint options.

Table 5. Transpose of sensitivity matrix

	x1	y1	x2	y2	x3	y3	x4	y4	x5	y5	x6	y6	x7	y7	x8	y8	C1	C2	C7
Con1:	-1.0	0	-1.0	0	-1.0	0	-1.0	0	-1.0	0	-1.0	0	-1.0	0	-1.0	0	0	0	0
Con2:	0	-1.0	0	-1.0	0	-1.0	0	-1.0	0	-1.0	0	-1.0	0	-1.0	0	-1.0	0	0	0
Con3:	-13.3	7.8	0	0	-7.1	3.3	-13.7	0	-26.6	0	-16.4	5.8	-16.4	5.8	-13.1	7.8	0	-5.2	-1.0
Con4:	-33.6	0	0	0	-20.0	5.7	-39.0	0	-33.6	0	-33.6	0	-33.6	0	-33.6	0	0	-14.0	0
Con5:	1.1	0	0	0	0	0	1.4	0	1.1	0	1.1	0	1.1	0	1.1	0	0	1.0	0
Con6:	25.8	0	0	0	0	0	67.2	0	25.8	0	25.8	0	25.8	0	25.8	0	0	-46.0	0
Con7:	0	0	0	0	0	0	0	0	-8.7	0	-1.6	-0.9	-1.6	-0.9	0	0	0	0	0
Con8:	0	0	0	0	0	0	0	0	9.5	0	2.3	1.3	1.6	0.9	0	0	0	0	0
Con9:	0	0	0	0	0	0	0	0	0.0	0	0.1	0.1	0.1	0.1	0.2	0.1	0	0	1.0
Con10:	5.7	0	0	0	0	0	0	0	9.6	0	2.4	-4.2	2.4	-4.2	0	-5.5	0	0	44.0
Con11:	0.0	0	0	0	0	0	1.4	0	4.3	2.5	1.1	0.6	1.1	0.6	0.0	0	0	-1.0	0
Con12:	1.0	0	0	0	0	0	0	0	1.0	0	1.0	0	1.0	0	1.0	0	0	0	0
Con13:	-0.3	0	0	0	0	0	0.3	-1.0	-2.0	-1.0	-0.7	-0.3	-0.7	-0.3	-0.3	0	0	0	0
Con14:	0	0	0	0	0	0	0	0	2.0	0	0.9	-0.7	0.4	0.2	0	0	0	0	0
Con15:	0	0	0	0	0	0	0	0	0	0	-0.3	-0.2	-0.3	-0.2	0	0	0	0	0
Con16:	-2.1	0	0	0	-1.3	-0.8	-3.0	0	-2.1	0	-2.1	0	-2.1	0	-2.1	0	1.0	-1.0	0
Con17:	0	0	0	0	0	0	0	0	-11.4	0	-3.4	-0.4	-2.8	0.0	-0.2	-0.1	0	0	-1.0
Con18:	-1.0	0	0	0	0	0	0	0	2.7	0	0.3	-1.4	0.3	-1.4	-0.5	-1.9	0	0	0
Con19:	-0.3	0	0	0	-0.5	-0.9	-0.7	0	-0.3	0	-0.3	0	-0.3	0	-0.3	0	0	0	0

This sketch solver is being implemented in OpenCSM as part of the Engineering Sketch Pad (ESP) Sketcher. While the technique was designed with OpenCSM implementation in mind, there is no reason the sketch solver could not be adapted for use with other CAD systems.

VII. Acknowledgments

This research was conducted thanks to the NASA Cooperative Agreement NNX11AI66A and Christopher Heath, the Technical Monitor for the project. Special thanks to Bob Haimes (MIT) for his support throughout the development of this sketcher software. Also, we would like to thank Robert Latham (Onondaga Community College) for his assistance with the commercial software packages.

References

[1]Bozdoc, M., "The History of CAD," June 2003, mbinfo.mbdesign.net.

[2] "Computer-aided design," Dec 2012, en.wikipedia.org/wiki/Computer-aided_design.

[3]Dannenhoffer, J. F., "OpenCSM: An Open-Source Constructive Solid Modeler for MDAO," *AIAA Aerospace Sciences Meeting*, 2013.

[4]Hoffman, C. M. and Vermeer, P. J., "Geometric Constraint Solving in R^2 and R^3," *Computing in Euclidean Geometry*, World Scientific Publishing, Singapore, 2nd ed., 1995, pp. 266–298.

[5] "ProEngineer," 2009, Release: Wildfire 5.0.

[6] "SolidWorks Education Edition," 2012, version SP4.0.

[7]Giesecke, F. E. et al., *Engineering Graphics*, Pearson Prentice Hall, eighth ed., 2000.