



A Workbench for Geometric Constraint Solving

Rogier de Regt¹, Hilderick A. van der Meiden² and Willem F. Bronsvort³

¹Delft University of Technology, Rogier.de.Regt@gmail.com

²Delft University of Technology, H.A.vanderMeiden@tudelft.nl

³Delft University of Technology, W.F.Bronsvort@tudelft.nl

ABSTRACT

Geometric constraints are used in geometric and feature modeling systems to impose conditions on the shape of a product. To determine whether these conditions are correctly applied to a model, and to compute the final solution in which all conditions are satisfied, a geometric constraint solver is used.

Current constraint solvers usually handle only 2D constraints. With the introduction of solvers that support 3D constraints, the interaction and visualization techniques for 2D constraints are no longer sufficient. Also, the feedback provided by the 2D constraint solvers, especially if problems are not well-constrained, is limited and not applicable to 3D modeling environments.

In this paper, a workbench is presented that can assist a user in defining geometric constraints in 3D and in creating a valid 3D model. Feedback provided by the solver is presented to the user in several ways that enable him to easily interpret this feedback.

Keywords: geometric constraint solving, graph constructive approach, sketching, visual feedback.

DOI: 10.3722/cadaps.2008.471-482

1. INTRODUCTION

Geometric constraint solvers are an important part of, in particular, parametric and feature-based CAD systems. Parametric and feature models include constraints that impose conditions on the model of a product. A geometric constraint solver is used to find a configuration for a set of geometric objects, such that these satisfy a given set of constraints between them.

In current CAD systems, the graphical user interface (GUI) usually gives the user the ability to create a 2D sketch, after which constraints (distances, radii, angles) can be added to the sketch. Operations like extrusion can be applied to a sketch, to create a 3D component. Additionally, between 3D components, simple constraints can be defined to connect them to each other.

However, a model may be incomplete or the specified constraints can be conflicting, in particular if there are not enough constraints or too many constraints defined between the objects. These situations are called under-constrained, respectively over-constrained. The geometric constraint solver is used to detect such conflicts, and gives feedback to the user when a conflict has been detected. If there are no conflicts, then the constraint solver will compute the resulting configuration. In practice, it turns out that it is quite hard to define a system of constraints that is well constrained, and to get insight into why and where a system is under- or over-constrained.

Problems become even more severe when 3D systems, in which constraints can be defined on 3D geometric objects, are considered. Although various 3D constraint solving methods are available [4], e.g. graph constructive methods [5], there are few facilities to easily define 3D constraint systems and to get insight into the results of the solver.

In this paper, a workbench is presented that supports the user in these respects. Constraints can be defined in an interactive way by a 3D sketcher, and feedback, obtained from the constraint solver, is presented to the user in several helpful ways. A 3D graph constructive constraint solver is used here, but many of the facilities of the workbench are

more generally applicable with other geometric constraint solvers too. The main goal of the paper is to show the feasibility and usefulness of a workbench for geometric constraint solving in general.

Section 2 gives some background on geometric constraint solving, and in particular on the graph constructive solver used here. Section 3 introduces the functionality of the workbench as a whole, and of the 3D sketcher in particular, and also gives some implementation aspects. Section 4 elaborates the feedback offered by the workbench to the user. Section 5 presents a case study. Section 6 gives some conclusions.

2. GEOMETRIC CONSTRAINT SOLVING

A *system of geometric constraints* consists of a finite set of geometric elements (the variables) and a finite set of geometric constraints (the relations) defined between these elements. In a CAD system, the user often draws a 2D/3D sketch that consists of points, lines and circles in 2D, and planes, spheres and cylinders in 3D. This is followed by the assignment of geometric constraints, such as distance, angle, parallelism, concentricity, tangency and perpendicularity, between geometric elements. When the geometric elements and constraints have been defined, the user can initiate the solving process to solve the system of constraints (this can also be done automatically when the system is changed, if incremental solving techniques are used).

During the solving process, it may turn out that the system of constraints is not well specified by the user. To handle this, the solver can return three general distinct outcomes: under-, over- or well-constrainedness. When solving a system of geometric constraints, the system may have some degrees of freedom (DOF) left. In that case, an infinite number of solutions is possible. This situation is called *under-constrained*. A constraint problem is *over-constrained* if there is no solution to the problem. A *well-constrained* problem is a problem that has a finite number of solutions.

Constraints that are not represented by actual values, but in general terms, have to be solved by a *generic solver*. Such a solver determines whether the geometric elements can be placed using the constraints, without taking the values that can be assigned to the constraints and geometric elements into account. If explicit values were assigned to each of the constraints, an instance of the general problem would have been created; such problems are solved by an *instance solver*. Several techniques can be used to solve constraint problems [4], including the following.

With *local propagation* [2], the system of constraints is represented in an undirected graph. The nodes of the graph represent the variables, constants and operations in the constraints. The individual constraints can usually be solved by several methods, and the constraint solver selects one of these methods. A problem cannot be solved if there are cycles among the constraints, and this makes local propagation only useful for simple systems of constraints.

Numerical constraint solvers translate the constraints into a system of algebraic equations that are solved by applying iterative techniques, e.g. the Newton-Raphson method [10] or the relaxation method [13]. Numerical constraint solving is a general method and is, in comparison to other constraint solving techniques, able to handle large nonlinear systems. However, a problem that often occurs is that geometric constraint systems have several solutions, whereas an iterative method can provide only one. Another problem is that if the initial configuration is not well chosen, the configuration will not converge to a solution or to an unwanted solution. Alternatives have been brought up, e.g. the use of homotopy [9] and bisection [1]. These methods provide better solutions in some of the cases where normally convergence problems arise, but they are slow compared to the Newton-Raphson method.

Just like numerical constraint solvers, *symbolic constraint solvers* try to solve a system of algebraic equations. But instead of using iterative techniques, symbolic algebraic methods are used to find the generic solution. Used methods are Wu-Ritt's decomposition algorithm [15] and the Gröbner basis method [8]. They change the set of equations to new forms, which are easier to solve. A problem may occur when certain equations in the basis function algebraically depend on one another when evaluated with specific constraint values [6]. At a generic level (where explicit values are avoided in the system of equations), the solver may come to the conclusion that a solution exists, whereas no specific configuration can be found to satisfy the system of constraints. Other drawbacks are that the algorithm may require exponential running times and memory (w.r.t. the number of constraints).

Rule constructive solvers use rewrite rules to discover and execute the constraint steps [3, 7]. For example, Brüderlin [3] makes use of Prolog to define constraints in terms of rules (based on compass and ruler constructions) and facts (defined by the user in a sketch), which has the advantage that it can be easily extended to make the system more powerful. However, this method of constraint solving is not efficient when large systems of constraints have to be solved. This is due to the searching and matching of the facts in the rule base.

Graph constructive solvers consist of two phases; the first phase is the analysis phase and the second the construction phase. In the *analysis phase*, the constraint problem, represented by a graph, is decomposed into several sub-problems that can be solved, and be merged into the solution of the complete problem. For example, Hoffmann et al. [5, 6] make use of clusters to solve geometric constraint problems. In the second phase, the *construction phase*, the sub-problems are actually solved and merged. This is often done by symbolic or numerical methods. Indeed, with graph

constructive solving, different combinations of approaches can be utilized to create a specific constraint solver for a specific task. The approach is efficient by decomposing a large system of constraints into smaller sub-problems, which can usually be easily solved. Therefore, this is a popular approach to geometric constraint solving. In this paper, it is assumed that the workbench is using a 3D graph constructive solver. The most important aspects of the specific solver used here will be discussed now; see [14] for more details.

The class of geometric constraint problems that can be solved is as follows. The variables are limited to 3D points and the constraints are limited to two types: distance (between two points) and angle (between the two lines from a point to two other points) constraints. The parameters of the constraints, i.e. the distance and angle values, are also part of the problem definition. A problem is represented as a constraint graph, a bipartite graph of which a vertex (node) is either a variable or a constraint, and an edge between a variable and a constraint indicates that the constraint is imposed on that variable.

The first phase of the graph constructive method is the *analysis phase*. In this phase, a constraint problem is decomposed into sub-problems of which the solution can be determined relatively easy. Subproblem solutions are represented by *clusters* of points of which the relative position and orientation are known. A cluster can be considered as a rigid body that can be translated and rotated, while remaining a solution.

When a cluster is formed, the relative position of the points in the cluster must be determined, where all constraints on these points must be considered simultaneously. These constraints can form cycles in the constraint graph, which must be solved simultaneously. Here, only cycles consisting of three constraints on three points are solved. The results are three-point clusters, i.e. triangles. For example, when the three distances are known between three points, then their relative position can be computed. Rules for this are referred to as *triangle solving rules*. The decomposition thus begins with the selection of clusters consisting of three points, of which the relative orientation and position can be determined by triangle solving rules.

In this phase, the *merging of clusters* is also planned, to create a complete decomposition of the problem. Clusters can be merged if they share a number of point variables, i.e. if they contain point variables with the same name, such that no degrees of freedom are left between these clusters. Three three-point clusters can be merged if they share four points, i.e. the three triangles can be merged into a tetrahedron, which is a 3D cluster. Two 3D clusters, which can be tetrahedrons or more complex configurations that share three points can be merged by transforming one of the clusters such that the shared points coincide. These rules are referred to as *cluster merging rules*.

By merging clusters, larger clusters are obtained from which previously unknown distances and angles can be determined. For example, if three three-point clusters are merged into a tetrahedron, all distances and angles in the tetrahedron are known. The newly known distances and angles may, in turn, be used to satisfy other (cycles of) constraints.

In some cases, the decomposition into sub-problems fails, in particular if there are too many or too few constraints defined, such that the relative position and/or orientation cannot be determined. These structural problems can be divided into two groups: structurally under-constrained problems and structurally over-constrained problems. A well-constrained tetrahedron consists of four points and six distances (see Fig. 1a). If there are not enough constraints defined, then the problem is *structurally under-constrained* (see Fig. 1b). If there are too many constraints defined, then the problem is *structurally over-constrained* (see Fig. 1c). However, even if there are exactly enough constraints defined, and thus the decomposition is successful, it is not guaranteed that the model is well-constrained, as will be shown later.

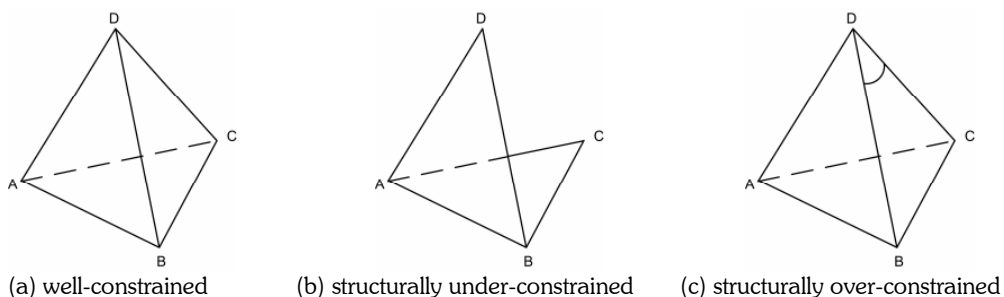


Fig. 1: Constrainedness of a tetrahedron.

If the well-constrained tetrahedron from Fig. 1a is decomposed, then the result can be as shown in Fig. 2. This diagram shows the clusters created and the relations between these clusters. Distance constraints are shown as two-point clusters. The diagram shows the *root cluster* at the top, which is formed by merging the clusters below it that are connected to it by lines. The decomposition algorithm works *bottom-up*, first creating a three-point cluster ABC by merging the distance constraints AB, AC and BC. This forms one side of the tetrahedron. Once a distance constraint has been merged into a cluster, it cannot be used to form other clusters, unless it is derived from the existing cluster. For example, to create cluster ACD, the distance constraints AC, AD and CD are needed, but AC has already been merged into cluster ABC. Therefore, AC is first derived from ABC, as can be seen in the decomposition, and then merged with AD and CD to form cluster ACD. To form cluster BCD, first BC and CD are derived from previously formed clusters, and then merged with BD. The clusters ABC, ACD and BCD are merged into cluster ABCD, which contains all variables and constraints, and therefore the constraint problem is structurally well-constrained.

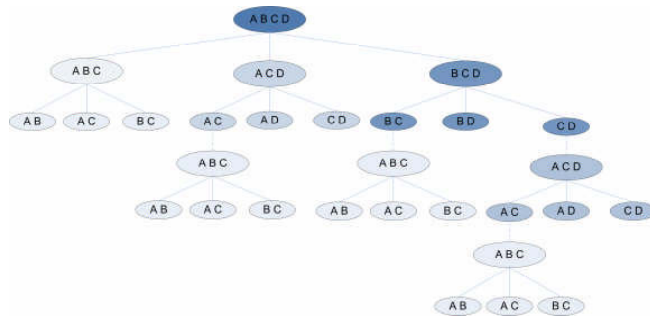


Fig. 2: Decomposition of a well-constrained tetrahedron.

The decomposition of the under-constrained tetrahedron from Fig. 1b is visualized in Fig. 3. In contrast to the well-constrained decomposition, there are now only two three-point clusters, ABC and ADB. Because there is no longer a distance constraint defined between the points D and C, a third three-point cluster cannot be formed. Now the three-point clusters have only two points in common, which is not enough to combine the clusters (for that four points are necessary). If clusters cannot be combined to one root cluster, because of too few constraints, then the constraint problem is structurally under-constrained.

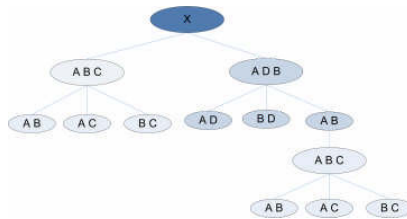


Fig. 3: Decomposition of a structurally under-constrained tetrahedron.

The decomposition of the over-constrained problem from Fig. 1c is very similar to the decomposition of the well-constrained problem in Fig. 2. The difference is that now the three-point cluster BCD cannot be determined, because there are too many constraints defined, which makes the cluster over-constrained and the constraint problem as a whole as well.

After the decomposition and the merge plan have been determined in the analysis phase, in the *construction phase* the triangular sub-problems are solved and the clusters are merged to actually obtain the solution of the constraint problem.

To solve a triangular sub-problem, which contains three points, three distances must be known, or two distances and one angle, or two angles and one distance. Five triangle solving rules are available for this. For example, if the three distances between the three points are known, two points can be arbitrarily fixed at the corresponding distance, and the third computed from the position of these two points and the other two distances. The other triangle solving rules all involve one or two angles, but are also straightforward. In solving a particular triangular problem, two solutions may

be possible. One of these solutions is selected, based on a *prototype*, i.e. a sketch of the intended solution provided by the user, so that, in the end, only the intended solution results, instead of several solutions. See [14] for details on this. If solving a cluster with a triangle solving rule yields no solutions, then the cluster is *incidentally over-constrained*. This happens, for example, if the sum of two distances is smaller than the third distance.

Three three-point clusters, resulting from solving triangular sub-problems, are merged into a tetrahedron if they share four points. The three clusters that are to be merged are rigidly transformed, i.e. translated and rotated, in such a way that the shared points coincide, and a tetrahedron is formed. In merging three three-point clusters, again two solutions may be possible, and again one of these solutions is selected using the prototype.

If the three three-point clusters cannot be merged, because there is no rigid transformation of the clusters such that the shared points form a tetrahedron, then the cluster is *incidentally over-constrained* again.

Two 3D clusters, which can be tetrahedrons or more complex clusters that share three points are merged by rigidly transforming one cluster such that the shared points coincide. The required transformation is straightforward, and a new cluster is formed in this way.

To merge clusters that share three points, the relative position of these points in each cluster must obviously be equivalent; otherwise a rigid transformation cannot be determined. If they are not equivalent, then the system of constraints is *incidentally over-constrained* again. Cluster merging can also fail here because a set of three points does not always form a triangle. A triangle degenerates to a line if one of the edges has zero length, and it degenerates to a point if all edges have zero length. When merging two clusters based on a set of three shared points, and these points are degenerate in both clusters, then the merge cannot take place because one or more rotational degrees of freedom are left. In this case, the system is considered *incidentally under-constrained*.

Structurally over- and under-constrained situations can be resolved by removing, respectively, adding constraints. Incidentally over- and under-constrained situations, on the other hand, can be resolved by changing the values of parameters of existing constraints. As constraint problems are often very complex, it may become difficult for a user to locate where a problematic situation occurs, and how to resolve it. The workbench for geometric constraint solving, introduced in the next section, can be very helpful in this respect.

3 THE WORKBENCH

The workbench for geometric constraint solving consists of several components. These components, and how they relate to each other, are shown in Fig. 4. The part of the diagram that is shaded, is the GUI from which the user can take actions, the other part is the graph constructive constraint solver.

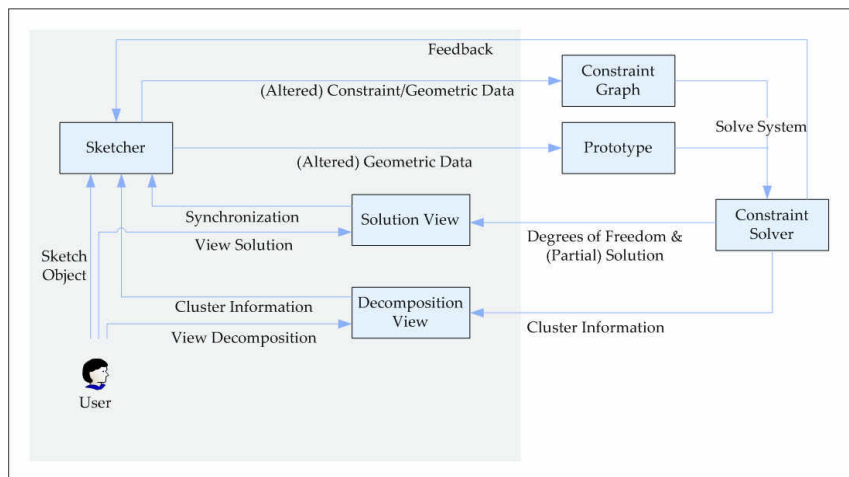


Fig. 4: Workbench components.

The 3D sketcher provides the user with an interface to create a rough sketch by placing points and lines. During or after placing these points and lines, the user can add constraints to the model. During the construction of the sketch, the constraint data is stored in the constraint graph, and the sketch without the constraints in the prototype. Performing actions in which the constraints are altered will directly result in an update of the constraint graph. When the user issues

the command to solve the system of geometric constraints, then this system consisting of the constraint graph and prototype information will be used to solve the system. After the solving process, the result is returned. This initiates other processes, to create visual feedback. Different views are used to return feedback to the user about the system of geometric constraints and to help producing the desired sketch. The views are used to visualize the information retrieved from the solver and to exchange information with the sketcher.

In the *decomposition view*, the decomposition of the system of constraints into clusters, as returned by the solver, is shown. This is very useful when a system is under- or over-constrained. Individual clusters can be selected in this view, which will result in a similar selection in the sketcher.

If the system of geometric constraints is well-constrained, the final solution can be visualized in the *solution view*. In this view, the sketch is transformed into a model where all constraints are satisfied. The sketch can be synchronized with the model in the solution view. The decomposition view and the solution view, and the synchronization of the latter with the sketch, will be discussed in Section 4. Here the functionality of the sketcher will be outlined. Another sketcher for defining systems of constraints, somewhat similar to ours, is presented in [12]. However, that sketcher is used to support solution selection, whereas our workbench is intended to support creation of well-constrained systems. Basically, the functionality of our solver consists of the following:

- the usual user interface components, such as a taskbar, a toolbar, viewports, an object panel, and a status bar;
- points can be created, and are visualized as small spheres;
- lines can be created between two points, and are visualized as thin cylinders;
- the sketch is basically a wire frame, consisting of points and lines;
- different viewports can be used to simultaneously display a side, a top, a front and a perspective view of the sketch, for the creation and exploration of the sketch;
- facilities for zooming, panning and rotating in these viewports;
- a distance constraint can be created by selecting two points or a line;
- a distance constraint is visualized as a thin cylinder between the two points, which may replace an existing line between the points, in a color different from the one used for lines;
- an angle constraint can be created by selecting three points or two lines (or distance constraints);
- an angle constraint is visualized as a transparent circle section between two lines (or distance constraints);
- the possibility to fix a particular point, which will fix its position and give it a different color;
- the ability to specify and consult specific information, e.g. the position of a point, the distance parameter in a distance constraint, and the angle parameter in an angle constraint, in a dialogue box.

The user interface of the sketcher is shown in Fig. 5. The points are visualized as small spheres, and the lines and distance constraints as thin cylinders, to create depth in the sketches, in particular a better distinction between overlapping elements, and to make selection of elements easier.

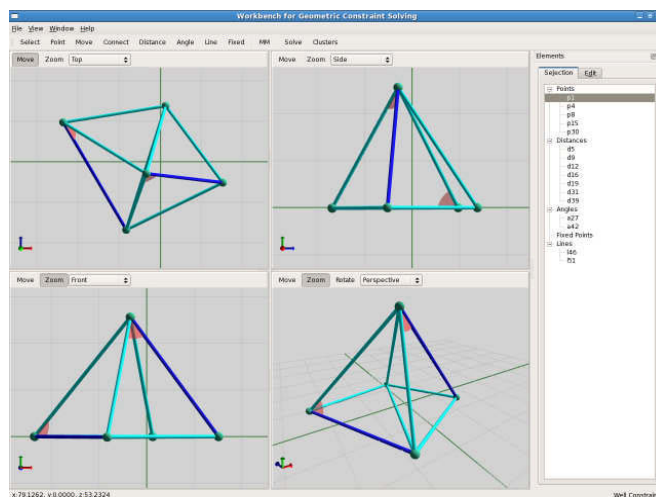


Fig. 5: User interface of the sketcher.

The workbench has, just like the constraint solver, been implemented by using Python, an object-oriented programming language. Qt [16] has been used to develop the user interface, making use of PyQt bindings [17]. For the 3D visualizations, OpenGL has been used, making use of PyOpenGL bindings [18].

4. VISUAL FEEDBACK

Once a system of geometric constraints has been created in the sketcher, it can be sent to the constraint solver to be solved. The result returned by the solver is often difficult to interpret, and it would be very helpful if the user would get feedback on how the system is constrained, and, if it is well-constrained, on what the final solution looks like. In this section, we discuss two views for the user to help in this respect: the decomposition view and the solution view. These views are fully integrated with the sketcher, for which also several feedback facilities are presented.

First of all, the user gets a textual indication on how the system is constrained: well-constrained, (structurally or incidentally) under-constrained, or (structurally or incidentally) over-constrained. Notice that if the system is not well-constrained, it may be simultaneously under- or over-constrained in several parts, or even under-constrained in one part and over-constrained in another part.

The *decomposition view* shows the decomposition of the system into clusters in the form of a tree. The decomposition for the well-constrained tetrahedron of Fig. 1a is shown in Fig. 6. It resembles the decomposition shown in Fig. 2.

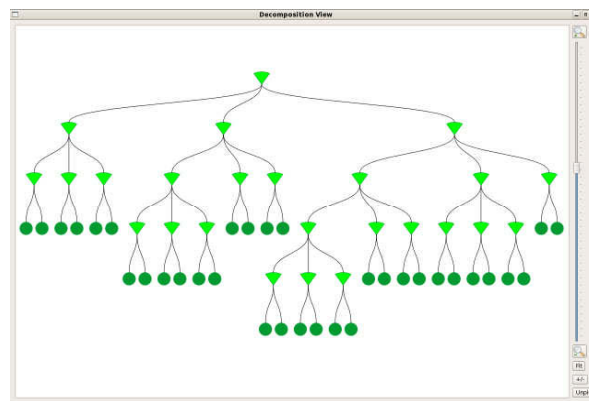


Fig. 6: Decomposition view.

Large models lead to large decompositions and need basic, but essential interaction techniques to allow the user to investigate the decomposition. The whole tree or parts of it can be shown in the view. Together with a zoom option, translation sliders can be used to zoom in on a specific part of the tree. In addition, branches of the tree can be collapsed, so the user can concentrate on other branches. Collapsing part of the tree often results in white space, which is re-used: the tree is regenerated on the basis of the visibility of the clusters.

Clusters do not only hold information on how they relate to other clusters, but they also contain information that can be shown, after selection, in a transparent box (see Fig. 7), so it does not occlude parts of the tree. The box shows the constrainedness of the cluster, and the names of the points belonging to that cluster. If an individual point is selected, the name and position of the point are shown.

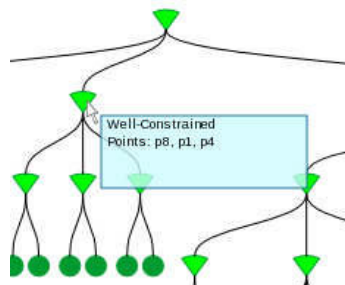


Fig. 7: Cluster information.

Although the decomposition view can be useful to get feedback about the constrainedness of a system of geometric constraints, we also enable the user to get feedback in the sketch. Cluster information can be visualized in the sketch by a *cluster hull*, a convex hull containing the points that are part of the cluster. A cluster hull can consist of only one point or one distance constraint, but can also contain several points, lines and constraints. When the user selects a cluster in the decomposition view, the cluster will be highlighted in the sketch, so that it can be easily located (see Fig. 8).

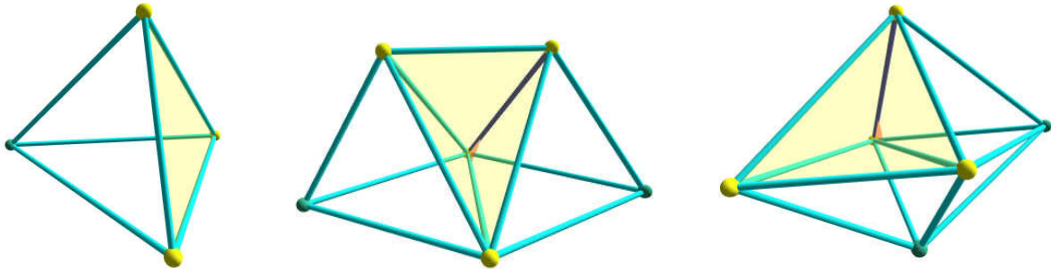


Fig. 8: Highlighted clusters in sketches.

There are many other facilities available, including simultaneously highlighting several clusters in the sketch, and limiting the points, lines and constraints visualized in the sketch to certain clusters, either highlighted or other ones. In this way, it is possible to concentrate on particular clusters in the sketch, and how these are related to each other, without being hampered by other information in the full sketch.

If the outcome of the geometric constraint solver is that a system of geometric constraints is under-constrained, then there are two ways to inform the user about the whereabouts of the problem: in the decomposition view and in the sketch.

In the decomposition view, clusters will turn red if they are under-constrained. The full decomposition tree will only be partially expanded, to where the problems arise. In Fig. 9a a simple sketch is shown of five distance constraints, defined between four points. The two connected triangles in 3D form a structurally under-constrained system. Once the decomposition view is opened, the user can clearly see in the partially expanded tree that the model is under-constrained, and the problematic clusters are shown (see Fig. 9b). The two green clusters (which are the triangles in the sketch) cannot be combined, which is indicated by the red root cluster. The full decomposition of the system is visualized in Fig. 9c.

We have already seen that we can highlight clusters from the decomposition view in the sketch. By applying different colors to cluster hulls in the sketch, we can inform the user about the constrainedness of a system of constraints. For under-constrained systems, we automatically visualize the cluster hulls at the level where the decomposition has failed, i.e. the clusters that cannot be merged (see Fig. 9d). These clusters are the same as the bottom two clusters, visualized green, in the decomposition view in Fig. 9b. They are by themselves well-constrained, but are nevertheless drawn red in the sketch, because they cannot be merged. This gives more insight than visualizing the cluster hull of the top red node from the decomposition view, which contains all the points that cannot be merged.

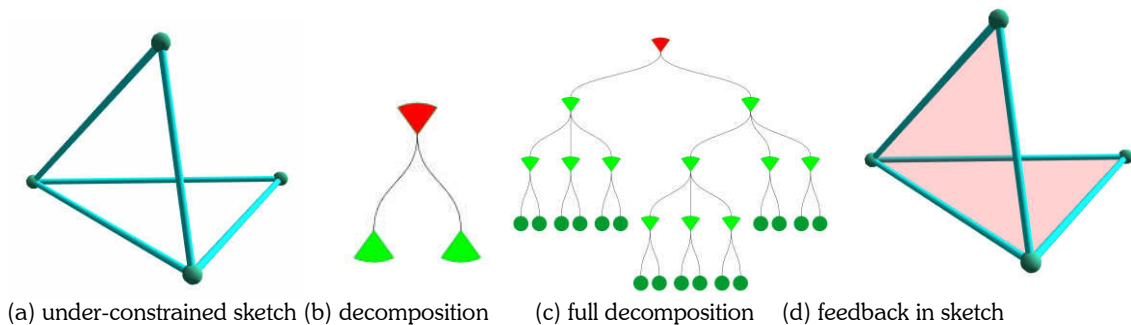


Fig. 9: Structurally under-constrained sketch with feedback after solving.

The visualization techniques for over-constrained systems are similar to those for under-constrained systems, but there are some important differences, e.g. which clusters are visualized.

Clusters for systems that are over-constrained, are colored blue in the decomposition view. The full decomposition tree will again only be partially expanded, to where the problems arise. A simple example of a structurally over-constrained system is given in Fig. 10a. There are four points and seven constraints between the points: six distance constraints and one angle constraint. The decomposition view visualizes the over-constrained clusters in blue (see Fig. 10b). The information box that is shown for the lowest over-constrained cluster in the tree states that the cluster is structurally over-constrained, and gives a list of the points that are part of the cluster. In the full decomposition (see Fig. 10c), there are more blue nodes. The extra nodes all correspond to the same lowest over-constrained cluster, which is referenced by several other clusters (see Section 2).

For the visualization of over-constrained systems in the sketch, the relevant cluster hulls are colored blue too (see Fig. 10d). The visualized clusters are the lowest blue clusters in the decomposition view. In the lowest blue cluster in Fig. 10b, three distance constraints and one angle constraint occur in a triangle, which is a structurally over-constrained situation. With the exact cluster where the problem occurs indicated, the user can easily remove one of the constraints in that cluster.

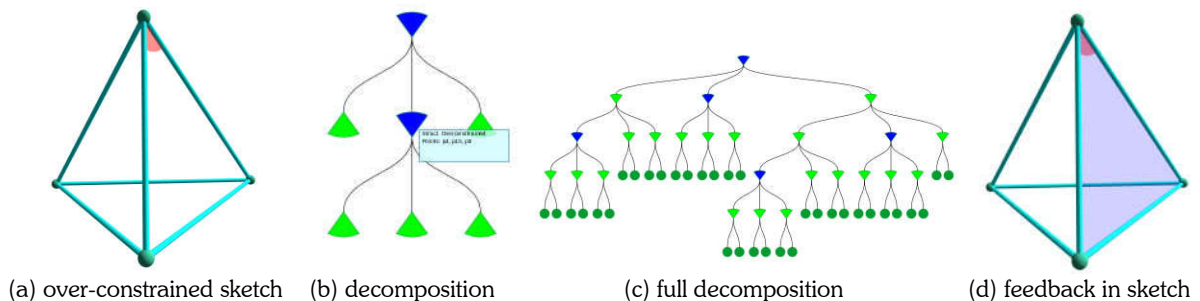


Fig. 10: Structurally over-constrained sketch with feedback after solving.

A well-constrained system of geometric constraints is the simplest case. Because there is no constrainedness problem, no clusters will be automatically visualized in the sketch. However, the clusters can still be seen and selected in the decomposition view (see Fig. 6 and Fig. 7), and related with the sketch by highlighting, which may be useful for further changes.

In the examples given above, structurally under- and over-constrained situations occurred. In case of incidentally under- and over-constrained situations, similar feedback and interaction are provided.

Because combinations of (structurally and/or incidentally) under- and over-constrainedness can occur in a single system, for each cluster in the decomposition the constrainedness can be obtained from the information box, shown when the cluster is selected.

As already indicated, structurally under- and over-constrained situations can be resolved by adding, respectively, removing constraints; incidentally under- and over-constrained situations by changing the values of parameters of existing constraints. As constraint systems are often very complex, and even combinations of under- and over-constrainedness can occur in a system, it may be difficult to make a system well-constrained. For example, removing one of the constraints in a structurally over-constrained situation may lead to a structurally under-constrained situation. The workbench can effectively support the user in creating a well-constrained system. In the following section, a case study will be described to illustrate this.

Once the system of constraints is well-constrained, the solution obtained from the solver is shown in the *solution view*. In this view, the new point positions, computed by the solver, are visualized. During solving, one of the points will be chosen to be at the origin of the coordinate system. From there the positions will be determined for the other points, relative to the one in the origin. Lines between the points are also visualized in the solution view, but the constraints are not. The solution view gives insight into the resulting configuration of the points.

Once a solution has been determined and visualized in the solution view, this solution can be transferred back to the sketch. Synchronizing the solution with the sketch can be helpful in cases where the solution is better visualized than the originally created sketch. In certain cases where distances or angles are very small in the solution, we might like to keep the sketch as it is, because it is easier to interact with when further changes are required.

5 CASE STUDY

In the previous sections, some examples of simple systems of geometric constraints have already been given. Different situations of constrainedness were discussed, and how they can be located in the decomposition view and in the sketch. In this section, a more complex case study is presented. The model will represent a tent, consisting of a hexagon as a base and a pyramid as a roof. During the construction of the model, the facilities of the workbench are used to create the final, well-constrained model.

First, the 2D hexagonal base of the model is sketched, and distance constraints are added between the points. Also, a point is created in the center of the hexagon, and additional distance constraints are added between this point and the points that form the base. Another six points are added to the model, with distance constraints between the points in the base and the new points, and distance constraints to mutually constrain the new points (see Fig. 11a).

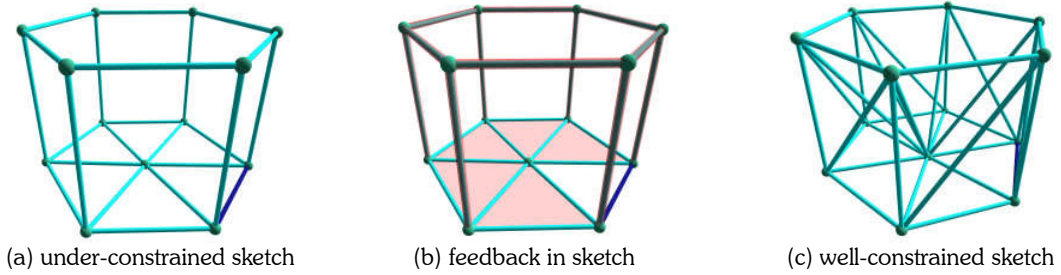


Fig. 11: Basic configuration.

When the sketch is sent to the solver, it turns out that it is structurally under-constrained. The hulls of the involved clusters are visualized in the sketch (see Fig. 11b). The base consists of five clusters, and the other twelve distances are represented as clusters on their own. All the clusters are turned red, which indicates that the geometric constraint solver could not create and combine tetrahedrons to create a solution. There are too few constraints defined, which is also indicated in the status bar of the workbench. If the decomposition view is opened, a red root node is visualized with seventeen clusters below it: the five clusters of three points (the cluster hulls at the base), and the twelve clusters of two points (connected by a distance constraint). These clusters cannot be merged.

To resolve the problem, new constraints should be defined. With the methods used by the geometric constraint solver in mind, tetrahedrons can be constructed to obtain a well-constrained model, as illustrated in Fig. 11c. Obviously, this is not the only way to correctly constrain the model.

In the next step, a roof is added to create the final model. By adding an apex point, and connecting new distances to it, the sketch is constrained and the roof is created (see Fig. 12a). Although the roof seems nicely modeled and constrained, the solver returns that the sketch is structurally over-constrained, and in the sketch it is clearly shown that the over-constrainedness occurs in the added roof (see Fig. 12b).

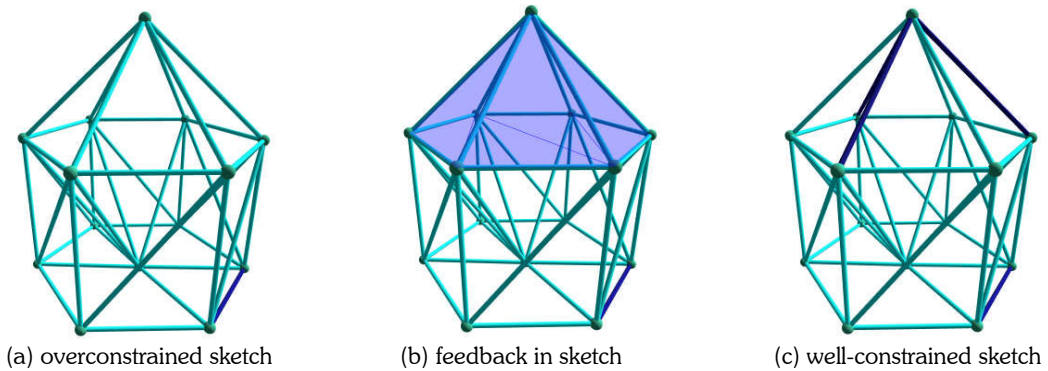


Fig. 12: Complete configuration.

The whole roof cluster is turned blue, thus too many constraints were added here. Removing the distance constraints one by one, it turns out that three distance constraints are enough to constrain the roof (see Fig. 12c). Lines were used to replace the redundant constraints, so the intended sketch is maintained.

6. CONCLUSIONS

A workbench to support the user of a geometric constraint solver, in particular a graph constructive solver, was presented. A sketcher can be used to define a model, with 3D points, lines and geometric constraints. It offers several useful facilities to intuitively create and interact with a system of 3D geometric constraints. After a sketch has been created by the user, the geometric constraint solver can be initiated. Feedback from the solver is difficult to interpret. Therefore two views were introduced, the decomposition view and the solution view, to present the user more information about the system of geometric constraints.

The decomposition view visualizes the decomposition of the problem into clusters. If the system is under- or over-constrained, the user can directly see the specific location where the problem occurs, because only that part of the tree is visualized and the involved clusters are given a special color. However, (part of) the tree can be quickly expanded or collapsed to obtain or hide information. From the decomposition view, there is a connection with the sketch: selection of one of the clusters in the decomposition view, will lead to the corresponding cluster hull being highlighted in the sketch. This offers direct feedback on the cluster's contents and location, which can be very useful when browsing the graph.

In case of a constrainedness problem, the involved clusters are also visualized in the sketch, via their hulls, in a special color. However, information about the whole decomposition is only available in the decomposition view, to limit the amount of information in the sketch.

Once the model is well-constrained, the user can obtain the final solution in the solution view. From this view, the model can be synchronized with the original sketch, so a better sketch can be obtained to further develop the intended model.

Many of these facilities are also applicable with other geometric constraint solvers. The sketcher is independent from the solver. The decomposition view is only applicable when the solver returns a decomposition, which is not only the case for graph constructive solvers, or bottom-up decomposition solvers, but also for top-down decomposition solvers [5, 11]. The same applies to the cluster feedback in the sketch, which is tied to a decomposition. For other solvers, however, other feedback might be useful. The solution view is again independent from the solver. The additional implementation effort required for another solver thus depends on the character of the solver.

Several extensions of the workbench are possible, including the following.

A decomposition is now shown as a tree, which implies that some clusters occur several times. This leads to a rather complex decomposition, which would not be necessary if it would be visualized as a graph instead of a tree. Less nodes would be visualized and the representation would, in fact, also correspond better to the actual solving strategy.

It is attractive to have control over the basic elements of a sketch, i.e. the points, lines and constraints between them, especially when one has to create basic shapes or test a geometric constraint solver. However, when large models have to be created, standard 3D shapes, which contain several predefined constraints, are indispensable. A model could have a simpler representation, i.e. it might not show all the constraints defined in it, and the user could modify the object on a less detailed level. This would simplify the creation process, and the user could concentrate on problems like correctly connecting different shapes. The workbench can be very supportive in this too.

Currently, only points, lines and two types of constraints can be defined. With these constraints, almost any model can be created. However, to give a user more choice of primitives and constraints, could lead to a simpler representation and a better understanding of a model.

When a user tries to resolve an under- or over-constrained system, he still needs to have some understanding of the applied solving method. Otherwise, wrong choices can easily be made when adding or removing constraints, or when changing values of parameters. With the introduction of, for example, an expert system, the user might be advised how to change the system to make it well-constrained.

Altogether, the workbench enables a user to intuitively create a sketch with 3D constraints, and visually informs the user where problems occur, if the constraints are not satisfied. This can substantially help to create well-constrained systems in an easier way. Further development of the workbench along the proposed lines, will result in a version that is even more user-friendly than the current one, and can be used to specify more complex constraint systems.

7. ACKNOWLEDGEMENTS

Hilderick A. van der Meiden's work is supported by The Netherlands Organisation for Scientific Research (NWO).

8. REFERENCES

- [1] Ait-Aoudia, S.; Mana, I.: Numerical solving of geometric constraints by bisection: a distributed approach, *International Journal of Computing & Information Sciences*, 2, 2004, 66–73.
- [2] Borning, A.: The programming language aspects of Thinglab, a constraint-oriented simulation laboratory, *ACM Transactions on Programming Languages and Systems*, 3(4), 1981, 353-387.
- [3] Brüderlin, B.: Constructing three-dimensional geometric objects defined by constraints, In *Symposium on Interactive 3D Graphics '86: Proceedings of the 1986 Workshop on Interactive 3D Graphics*, 111–129, New York, NY, USA, 1987, ACM Press.
- [4] Dohmen, M.: A survey of constraint satisfaction techniques for geometric modeling, *Computers & Graphics*, 19(6), 1995, 831-845.
- [5] Fudos, I.; Hoffmann, C.M.: A graph-constructive approach to solving systems of geometric constraints, *ACM Transactions on Graphics*, 16(2), 1997, 179–216.
- [6] Hoffmann, C. M.; Vermeer, P. J.: Geometric constraint solving in R^2 and R^3 , In *Computing in Euclidean Geometry, Second Edition*, 266–298, Singapore, 1995, World Scientific Publishing.
- [7] Joan-Arinyo, R.; Soto-Riera, A.: Combining constructive and equational geometric constraint-solving techniques, *ACM Transactions on Graphics*, 18(1), 1999, 35-55.
- [8] Kondo, K.: Algebraic method for manipulation of dimensional relationships in geometric models, *Computer-Aided Design*, 24(3), 1992, 141-147.
- [9] Lamure, H.; Michelucci, D.: Solving geometric constraints by homotopy, In *Proceedings Third Symposium on Solid Modeling and Applications*, 263-269, New York, NY, USA, 1995, ACM Press.
- [10] Light, R. A.; Gossard, D. C.: Modification of geometric models through variational geometry, *Computer-Aided Design*, 14(4), 1982, 209-214.
- [11] Owen, J. C.: Algebraic solution for geometry from dimensional constraints, In *Proceedings Symposium on Solid Modeling Foundations and CAD/CAM Applications*, 397-407, New York, NY, USA, 1991, ACM Press.
- [12] Sitharam, M.; Arbree, A.; Zhou, Y.; Kohareswaran, N.: Solution space navigation for geometric constraint systems, *ACM Transactions on Graphics*, 25(2), 2006, 194-213.
- [13] Sutherland, I.E.: Sketchpad: a man-machine graphical communication system, In *AFIPS Conference Proceedings* 23, 1963, 323-328.
- [14] van der Meiden, H. A.; Bronsvort, W. F.: An efficient method to determine the intended solution for a system of geometric constraints, *International Journal of Computational Geometry and Applications*, 15(3), 2005, 279-298.
- [15] Wu, W.: Basic principles of mechanical theorem proving in elementary geometrics, *Journal of Automated Reasoning*, 2(3), 1987, 221–252.
- [16] Qt distributed by Trolltech: "<http://trolltech.com/products/qt>".
- [17] PyQt distributed by Riverbank. "<http://www.riverbankcomputing.co.uk/>".
- [18] PyOpenGL: "<http://pyopengl.sourceforge.net/>".