Solving over- and underconstrained geometric models

Alex Noort, Maurice Dohmen and Willem F. Bronsvoort
Faculty of Technical Mathematics and Informatics
Delft University of Technology
Zuidplantsoen 4, NL-2628 BZ Delft, The Netherlands
e-mail:(noort/dohmen/bronsvoort)@cs.tudelft.nl

Abstract

New approaches to solving underconstrained and inconsistent overconstrained 3D geometric models are described. When a model is found to be underconstrained, the underconstrained parts of the model are determined and shown to the user. The user has to add constraints to these parts, to make the model well-constrained. When an inconsistency in the model is found, the set is determined of the constraints that together cause the inconsistency. The model is made well-constrained, either by modification of one of these constraints, or by removal of one of these constraints from the model.

The approaches make use of a dependency graph, which contains information about dependencies between model variables. When an inconsistency is detected, the set of conflicting constraints is determined by traversal of the dependency graph. When an underconstrained situation is detected, the model variables in the underconstrained parts of the model are determined using this graph.

The concept of storing information about dependencies between model variables in a graph, can be used with all non-iterative constraint satisfaction techniques. An implementation of the approaches has been made in a 3D constraint solver based on degrees of freedom analysis. It identifies rigidly connected parts of the model, and then solves the geometric constraints in these parts incrementally by geometric transformations, taking into account the translational and rotational degrees of freedom of the constrained model parts. The solver handles loops of constraints by rewriting some of the constraints in the loop.

1 Introduction

In *constraint-based feature modeling*, *features* are the basic modeling entities. Features contain *shape* and *non-shape* information. The non-shape information can be information about the function of the feature, the assembly or manufacturing of the part the feature belongs to, etc. Both shape and non-shape information are used to assist the user when editing the model, by ensuring the validity of the features in the model.

Both types of information are represented by *constraints*. To satisfy these constraints, they are solved by a *constraint solver*. In this paper we concentrate on geometric constraints, such as a constraint that specifies two planes to be at a given distance of each other. These constraints represent shape information of features, such as the parameters, i.e. position, orientation and dimension.

In the feature modeling system SPIFF (DdB96; KDB97), *high level* constraints are mapped to combinations of *low level* constraints which are solved by the constraint solver. Shapes

contained in the features used in SPIFF are mapped to low-level geometric objects that the constraint solver can deal with. The constraint solver is based on *degrees of freedom analysis* (Kra92).

The models can only be solved if they are *well-constrained*, which means that there are exactly enough constraints in the model to uniquely determine the parameters of all features in it. When some parameter is determined in different ways by multiple groups of constraints, the model is called *overconstrained*. When some parameter is not determined, the model is called *underconstrained*.

Section 2 presents the concept of degrees of freedom analysis which the constraint solver is based on. Section 3 presents the *dependency graph* that is used to remove under- and overconstrained situations from the constraint model. Sections 4 and 5 present the ways over- and underconstrained situations can be repaired, i.e. removed from the model. Section 6 presents results and conclusions.

Although the concept of a dependency graph containing information about dependencies between model variables is presented here for a geometric constraint solver based on degress of freedom analysis, it can be used with all non-iterative constraint satisfaction techniques.

2 Degrees of freedom analysis

This section describes the concept of degrees of freedom analysis (Kra92), on which the geometric constraint solver has been based. Degrees of freedom analysis is a structured constraint solving approach. It solves a constraint model bottom-up, by first solving and then combining subgraphs using geometric knowledge. Constraint models consist of *geoms* with *markers* that represent the geometric elements of the model, and geometric *constraints* between markers of different geoms.

Geoms contain a coordinate system that has a variable orientation and position with respect to a world coordinate system. Markers are represented by a position and an orientation, and are fixed with respect to the geom they are part of. The set of all constraints directly between two geoms is called a *joint*. A joint is called *rigid* if the position and orientation of one geom of the joint are completely determined relative to the other geom of the joint by the constraints in the joint.

Every geom initially has three *translational degrees of freedom* and three *rotational degrees of freedom*, from now on called TDOFs and RDOFs. These degrees of freedom describe the directions in which a geom can move and rotate without violating any constraint already satisfied. The (R/T)DOFs of a geom are mutually independent, i.e. perpendicular.

A *constraint graph*, consisting of geoms, markers and constraints, is used to represent a model to be solved. This constraint graph contains geoms as nodes and joints as edges. A constraint graph is solved by merging geoms that have a rigid joint between them into a *macro-geom*; the graph has been solved when there is only one geom left.

The constraint solver alternates between two tasks. The first task is to solve constraints of rigid joints and to merge the geoms of these joints into macro-geoms. The constraints are satisfied using action analysis. The second task is to rewrite constraints from one joint to equivalent ones in other joints of the model in order to introduce new rigid joints. This is done using locus analysis.

Action analysis solves the constraints in a joint incrementally. To solve the constraints of a joint, one geom of the joint is taken fixed, and the other one is allowed to move. The

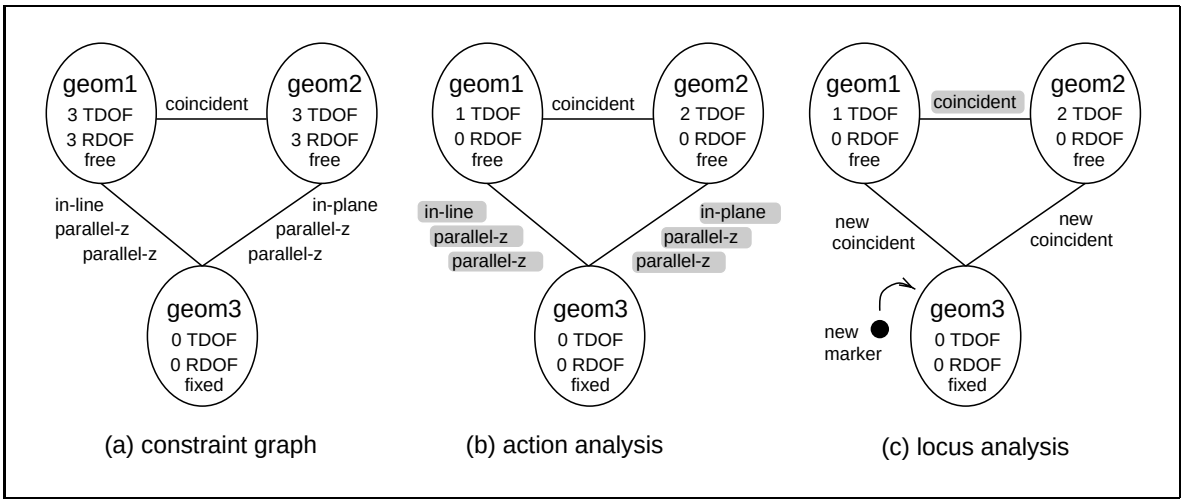


Figure 1: Locus analysis on three geoms.

constraints of the joint are solved one by one, by determining the DOFs left of the free geom, and then reducing these DOFs according to the constraint to be solved. The DOFs are reduced in such a way that the free geom will obey the constraints already solved.

The new DOFs of the free geom and the transformation needed to satisfy the constraints are registered in so-called *plan fragments*. For every DOF configuration of a free geom and constraint type to be solved, such a plan fragment has been implemented.

Locus analysis is performed between three geoms; in this process constraints from a joint between two of these geoms are rewritten to the joints on the third geom, see Figure 1. Before locus analysis can be performed, action analysis is done on the joints of the third geom, i.e. the joints with the marked constraints in Figure 1(b), with this geom being fixed. After action analysis has been performed, the two geoms of the joint on which locus analysis will be performed have certain DOFs left with respect to the third geom. Locus analysis will then be performed on the joint between these geoms, i.e. the joint with the marked constraint in Figure 1(c).

A locus is the area that can be reached by a geometric primitive, when moving according to its degrees of freedom. In locus analysis, the intersection of the loci of the markers of a constraint is the area where the markers have to be to satisfy the constraint. According to the intersection of the loci, new markers and new constraints are added to the model. These new constraints reduce the same DOFs between the two geoms of the original constraint as the original constraint does, but via the third geom.

In Figure 1(c), an example of locus analysis is given for a coincident constraint between a geom with one TDOF left and a geom with two TDOFs left after action analysis. The DOFs of these geoms are perpendicular to each other. The locus of the marker on the geom with one TDOF, is a line parallel to the DOF line and through the marker. The locus of the marker on the geom with two TDOFs, is a plane parallel to the two DOF lines and through the marker.

The intersection of those two loci is the point where both markers have to be in order to satisfy the coincident constraint. A new marker is added to the third geom with as position the intersection of the loci, and two coincident constraints are added between this new marker and the markers of the original constraint. These two constraints will reduce the same DOFs

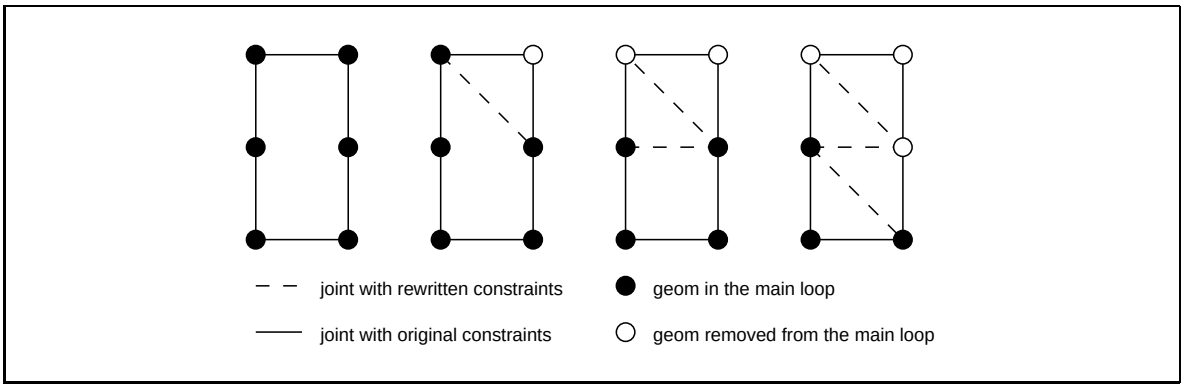


Figure 2: Splitting a loop with more than three geoms.

between the two geoms of the joint on which locus analysis has been performed as the original constraint.

As shown in the example of Figure 1, locus analysis can be used to solve loops with three geoms. To solve a loop with more than three geoms, it has to be reduced to a loop with three geoms. This is done by repeatedly replacing pairs of consecutive joints by a joint between the first and the last geom of the pair, thus removing a geom from the loop until there are three geoms left, see Figure 2.

The approaches to deal with over- and underconstrained models that have been presented by others are described in the next paragraphs. In dealing with over- and underconstrained models, three phases can be distinguished. In the first phase detection takes place. For an overconstrained model, the *involved constraints*, i.e. the constraints that cause the inconsistency, are detected. For an underconstrained model the undetermined properties are determined. In the second phase, a constraint needs to be chosen to repair the over- or underconstrained situation. For an overconstrained model, a constraint need to be chosen to be removed. For an underconstrained model, a constraint needs to be chosen to be added. In the third phase, the model needs to be re-solved.

To be able to detect an over- or underconstrained situation, the constraint solver has to provide facilities to register the geometric properties determined by individual constraints. In this way the solver is able to find the constraints involved in an overconstrained situation. Furthermore the solver must be able to partly solve a model in order to find the geometric properties not determined in the model. Such facilities can only be provided by structured, non-iterative constraint solvers such as used in (SAK90; KKP93; SB91; HB97).

To repair an overconstrained situation, a constraint has to be removed or edited. Removing a constraint really removes the overconstrained situation, whereas editing a constraint turns the inconsistent overconstrained situation into a consistent one. The choice of a constraint to be removed or edited can be done using many criteria. An example of such a criterium that is used by many systems, is the user's knowledge of its design goal, i.e. let the user choose a constraint (SAK90). A simple extension of this is to prohibit the user from choosing constraints of a particular type, e.g. constraints that determine topological relationships (KKP93). Removing topological relationships is often not desirable because it violates the integrity of the shapes of the model.

To repair an underconstrained situation, a constraint to be added can be chosen using default constraints (SAK90). In this approach, for each type of DOF, left between two

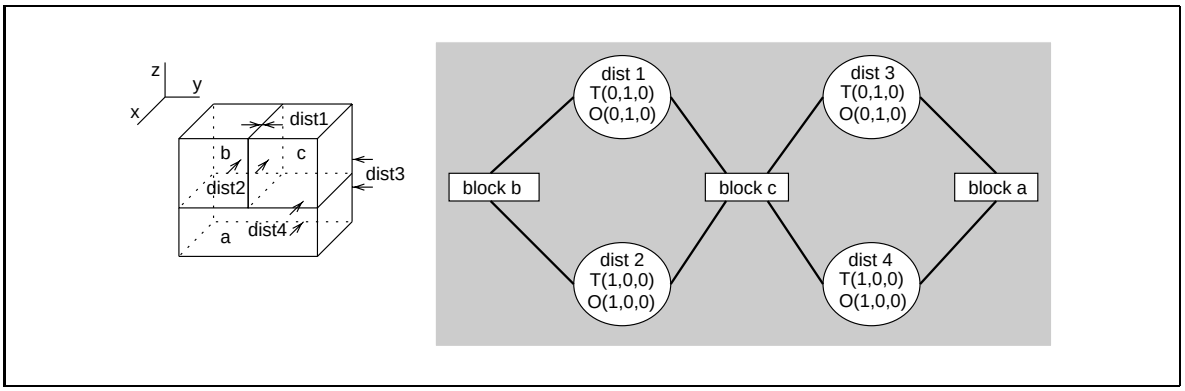


Figure 3: An example of a model and its dependency graph.

geometric elements, a constraint of a default type is put into the model to reduce the DOF.

3 Dependency graphs

This section describes the dependency graph, a data structure used to repair over- and under-constrained situations in a model. When the constraint solver has found an overconstrained situation, it should be able to find the constraints involved in this situation. These are the constraints that together reduce the same DOFs between two geoms as the constraint for which the inconsistency was detected. To find these constraints, the constraint solver has to know the DOFs reduced by each constraint. This *dependency information* is known when the constraints are solved, and can then be stored.

Dependency information is stored in dependency graphs that consist of geom nodes that represent the geoms of the model and constraint nodes that represent the constraints that reduced DOFs when the model was solved. In Figure 3, an example of a model and its dependency graph is given. In this example, the orientation and translation along the z-direction of the blocks have already been fixed. The constraint nodes contain the DOFs that the constraint reduced in solving the model, i.e. the *reduced DOFs*, and the *orientation* of the constraint, i.e. the orientation of the relevant marker of the constraint. The DOFs are denoted by T(x,y,z) for translation along a line with $(x, y, z)^T$ direction, and R(x,y,z) for rotation about an axis with $(x, y, z)^T$ direction; the orientation of the constraint is denoted by O(x,y,z). This orientation is used to be able to determine the *initially reduced DOFs* or *initial DOFs*, i.e. the DOFs a constraint reduces when no other constraint is involved. These initial DOFs depend on the constraint type and orientation. The reduced DOFs differ from the initially reduced DOFs when the initial DOFs of the constraints of the joint are dependent.

In Figure 4, an example of the initial DOFs of a constraint and its reduced DOFs in a joint with multiple constraints is given for a model in which the initial DOFs are dependent. The initial DOF of the dist1 constraint is T(0,1,-1), as can be found in the dependency graph of the first model. The initial DOF of the dist2 constraint of the second model is T(0,0,1). Because the initial DOFs are dependent, and the dist2 constraint happened to be solved first, the reduced DOF of the dist1 constraint differs from its initial DOF.

A model in which constraints exist whose reduced DOFs contain DOFs not initially reduced by the constraint is called *non-aligned*. An *aligned* model is a model of which the constraints reduce only DOFs that they reduce initially. This information is used when the

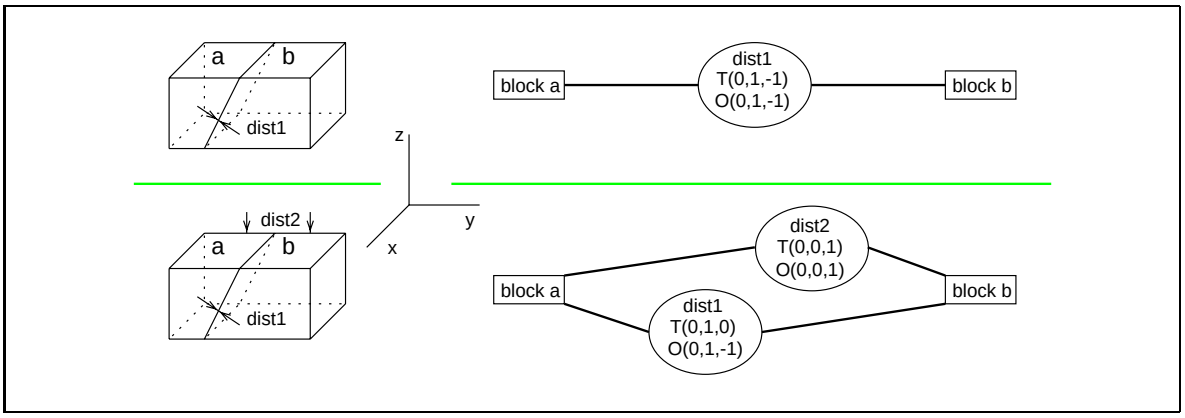


Figure 4: A model that shows the initial DOFs and one with dependent initial DOFs.

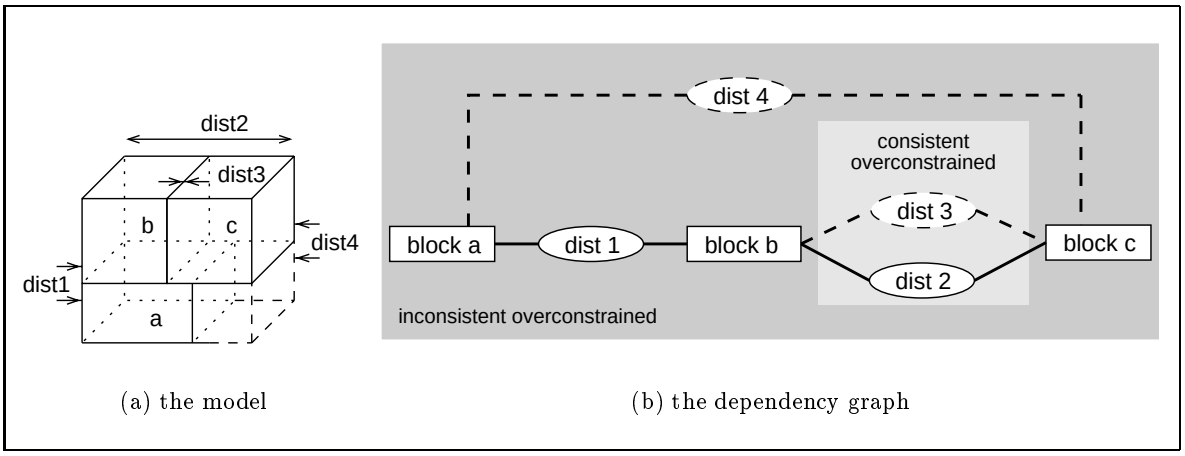
dependencies between two geoms are determined, because in a non-aligned model, geoms do not only depend for a given DOF reduction on the constraint reducing this DOF, but also on the constraints that enable the constraint to reduce the DOF. The model of Figure 3 is aligned, but the second model of Figure 4 is non-aligned, because the dist1 constraint reduces T(0,1,0) that it does not reduce initially.

The fact that constraint nodes represent constraints that reduced DOFs when the model was solved, implies that some constraints involving locally consistent overconstrained situations are not registered in the dependency graph. In such a situation, multiple constraints can reduce the same DOF. When solving, one of these constraints is used to reduce the DOF and is registered in the dependency graph, whereas the other constraints are not. These other constraints will therefore not be found when searching involved constraints.

An example of this situation is given in Figure 5, where the dist2 and dist3 constraints form a consistent overconstrained loop. Because of this, one of these constraints, dist3, will not reduce any DOFs and will therefore not be registered in the dependency graph. In case an inconsistent overconstrained situation will be found later that involves this consistent overconstrained situation, this constraint will not be recognized as an involved constraint, thus the overconstrained situation will not always be removed when removing an involved constraint from the model.

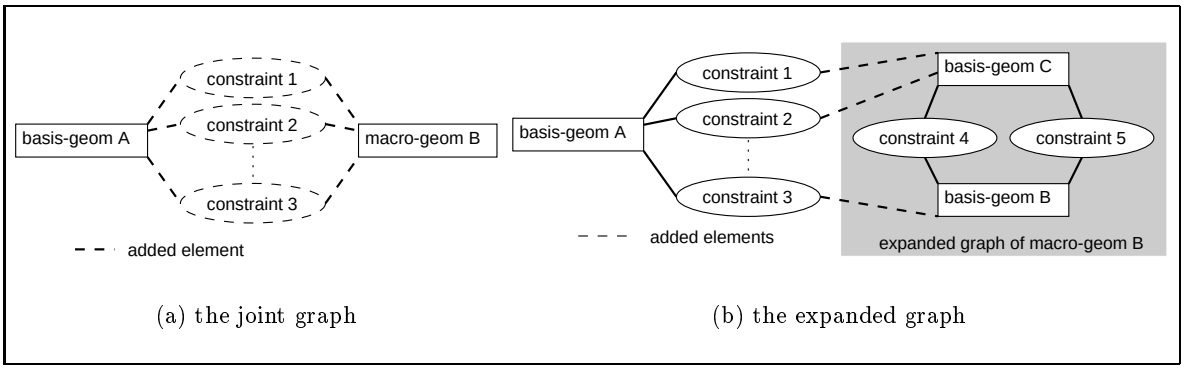
Our dependency graph is similar to the dependency graph as used in the system described by KKP93), whose constraint solver is similar to the one used in this system. The main difference between the approaches is that geometric elements in their constraint graph are fully determined by one constraint, whereas geometric elements in our constraint graph need multiple constraints.

Another application of dependency graphs is described by HB97). The similarity between their and our application is that they both involve the DOFs reduced between geometric elements of the model. The difference is that in the application described by Hsu and Brüderlin, the dependency graph is used to determine the part of the model to be re-solved after a geometric element has been changed, whereas in the application described in this paper, it is among other things used to find constraints involved in an overconstrained situation. Therefore, the dependency graph used in this system does not only register the number of DOFs reduced between pairs of geoms, but also which DOFs are reduced between them.



(a) the model (b) the dependency graph

Figure 5: Locally consistent overconstrained and globally inconsistent overconstrained.



(a) the joint graph (b) the expanded graph

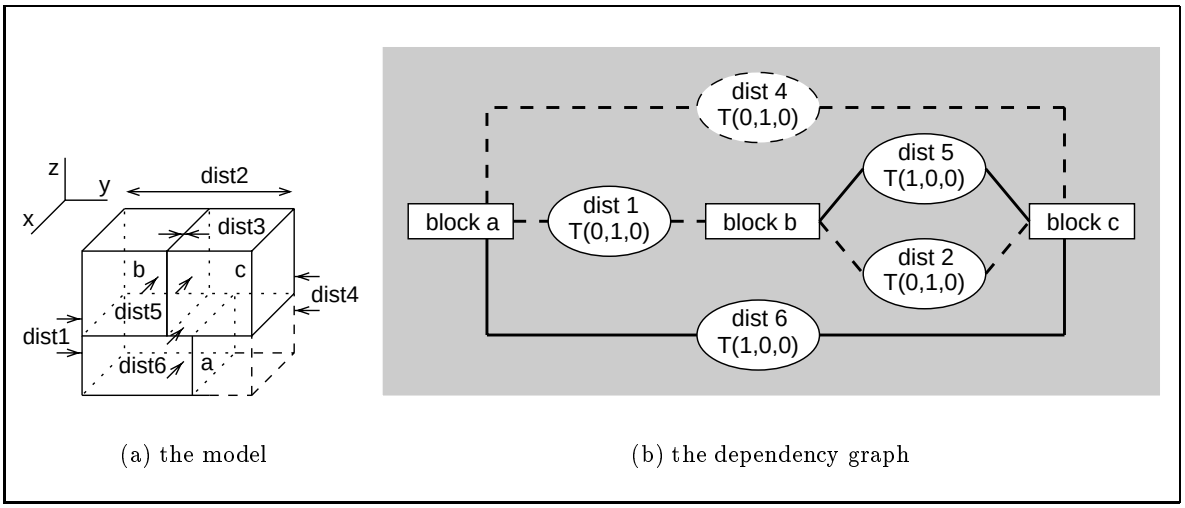
Figure 6: Building dependency graph in two phases.

Dependency graphs are built in two phases for each macro-geom that is created during the solving of a model. In the first phase a *joint graph* is built, and in the second phase this joint graph is expanded into an *expanded graph*. These two phases correspond to the way the solver works. The solver first solves the constraints between two geoms, and after that, in case the geoms have become rigidly connected, the solver merges them.

In the first phase, the solver does not change the structure of the constraint graph for the geoms of the joint. It only reduces the DOFs of the free geom of the joint. Similarly, the building of the joint graph does not change the structure of the dependency graphs of the geoms of the joint either, but only adds constraint nodes to the joint graph of this joint, containing the DOFs reduced by these constraints, see Figure 6(a).

In the second phase, the solver does change the structure of the constraint graph by merging the two geoms into a new macro-geom. The expanded graph built during this phase is created by replacing the macro-geom nodes of the joint graph by the expanded graphs representing these macro-geoms, see Figure 6(b). Every macro-geom in the model has its own expanded dependency graph describing the dependencies between all *sub-geoms* of that macro-geom.

These sub-geoms are all *basis-geoms*, i.e. geoms that are present in the constraint graph when the solving is started. In case an overconstrained situation is found in a joint on a macro-geom, the expanded dependency graph can be used to find all constraints that reduce given DOFs between given sub-geoms. In case an underconstrained situation is found, the



(a) the model (b) the dependency graph

Figure 7: Aligned model with its dependency graph. All blocks in the model have fixed dimensions.

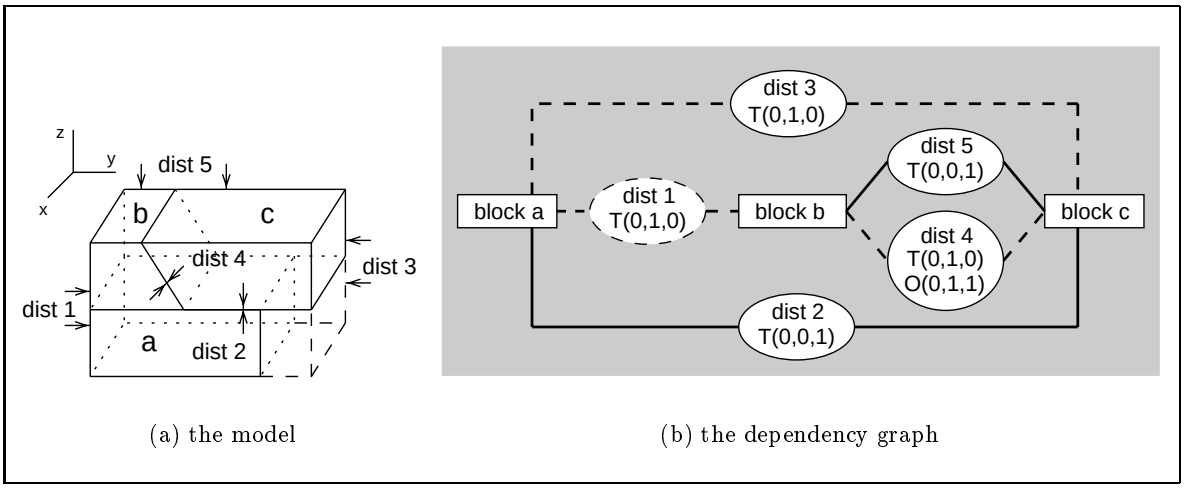
expanded dependency graphs of the macro-geoms are used to determine the basis-geoms that are fixed with respect to each other.

4 Overconstrained models

This section describes the detection and repairing of an overconstrained situation in a model. An overconstrained situation is detected by the constraint solver when it cannot determine a geometric transformation for a geom given its DOFs in order to satisfy the constraint that is solved. In such a case, another constraint has already removed DOFs that the constraint solver needs to satisfy this constraint. These DOFs are called the *overconstrained* DOFs. The set of involved constraints is built from the constraint that could not be solved, and the constraints that reduced the overconstrained DOFs between the two geoms. The overconstrained situation is repaired by choosing one of the involved constraints to be adjusted or removed from the model.

The constraints involved in an overconstrained situation are found in the dependency graph. To find them, the overconstrained DOFs need to be known first. These DOFs are detected by action analysis when it moves the free geom in order to satisfy the constraint, and by locus analysis when it tries to intersects the loci of the markers. After these DOFs have been detected, the other constraints that reduce these DOFs between the two involved geoms have to be found. These constraints are found using the expanded dependency graphs of the macro-geoms and the joint graphs of the treated joints. When searching in the dependency graphs, two cases have to be distinguished: searching in an aligned model and searching in a non-aligned model.

When the model is aligned, all involved constraints are found by searching for the constraints that reduce DOFs not perpendicular to the overconstrained DOFs between the two geoms. An example of finding the involved constraints in the dependency graph of an aligned model is given in Figure 7. This model consists of three fixed size blocks. The relative orientation and position along the z-axis of these blocks are fixed from the beginning, and the other DOFs are reduced by the distance constraints.



(a) the model (b) the dependency graph

Figure 8: Non-aligned model with its dependency graph. The dimensions of the blocks are fixed.

Let us assume that the constraint that could not be solved in this model is the dist4 constraint. The overconstrained DOF of this overconstrained situation is the translation DOF in the $(0, 1, 0)^T$ direction. The involved constraints are found in the dependency graph by finding the paths between the two geoms blockb and blockc that consist of constraints reducing DOFs not perpendicular to the involved DOF $(0, 1, 0)^T$. In this graph there is only one such path, which consists of the dist1 and the dist2 constraints, see Figure 7.

When the model is non-aligned, the technique used to find the involved constraints in an aligned model does not return the complete set of involved constraints. In a non-aligned model, besides the constraints that reduce DOFs not perpendicular to the overconstrained DOFs, other constraints can also be involved in the overconstrained situation. These other constraints are found by an extension of the approach for an aligned model.

For each of the constraints found by the algorithm for an aligned model, the initial DOFs are determined, see Section 3. When the reduced DOFs of such a constraint contain DOFs that are not initially reduced by the constraint, additional constraints are involved in the over-constrained situation.

To find these additional involved constraints, first the reduced DOFs causing this extra reduction need to be determined. These are the reduced DOFs that together with the DOFs reduced by the constraint can reduce the initially reduced DOFs of the constraint. After that, the constraints reducing these DOFs between the two geoms are searched for, and added to the involved constraints.

For an example of the repairing of an overconstrained situation for a non-aligned model, see Figure 8(a). In this model, the relative orientation and translation in the x-direction of the blocks have already been fixed. The model is similar to the model of Figure 7(a), except that blocks b and c now have two parallel oblique faces. Because of this, the dist4 constraint reduces a DOF that it does not reduce initially, which makes the model non-aligned.

Let us assume that an overconstrained situation has been found when constraint dist3 could not be solved. The overconstrained DOF is the translation in the $(0, 1, 0)^T$ direction. The involved constraints are found by first finding the constraints in the path between block a and block b that reduce this direction. This path is denoted by dashed lines, see Figure 8(b).

One of these constraints, the dist4 constraint, reduces a DOF that it does not reduce ini-

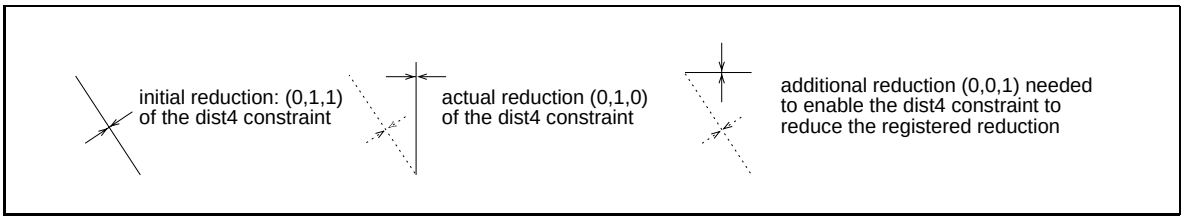


Figure 9: Additional reduction needed to reduce registered reduction.

tially. Therefore additional constraints have to be found that enable this constraint to reduce the DOF. These constraints need to reduce TDOFs in a direction such that the combination of these TDOFs and the reduced DOFs reduces the initial DOFs of the dist4 constraint. In this example, a translation reduction dependent on the $(0, 0, 1)^T$ is needed, see Figure 9. Therefore, the dist5 constraint is also an involved constraint in this overconstrained situation.

When the involved constraints have been found, one of them has to be chosen to be replaced or edited. Replacing a constraint involves constructing a constraint that only reduces the DOFs that are reduced by the chosen constraint that are not overconstrained, and replacing the chosen constraint by this new constraint. Editing a constraint involves adjusting the constraint parameters in such a way that the constraint is no longer inconsistent with the other constraints in the model.

If an overconstrained situation is to be repaired by editing a constraint, then the way the constraint parameter has to be adjusted needs to be known. This adjustment can be derived from the constraint that could not be solved, which is the dist3 constraint in Figure 8(b). The parameter of the chosen constraint, e.g. the dist1 constraint, needs to be edited in such a way that the dist3 constraint can be satisfied after the parameter has been changed.

The distadvantage of editing a constraint is that the model is still overconstrained. When later the parameter of a constraint involved in this overconstrained situation is changed, an inconsistency could be introduced. When the constraint is replaced, the overconstrained situation is removed from the model so no inconsistency can be introduced by changing a parameter of a constraint.

When the chosen constraint has been edited or removed, part of the model has to be re-solved to propagate the changes. Currently, the part of the model to be re-solved is the part of the model that is represented by the expanded dependency graph that contains the chosen constraint. A better strategy would be to re-solve only the smallest part of the model that depends, for its position and orientation relative to other parts of the model, on the chosen constraint. The smallest part can be determined using the dependency graph that contains the chosen constraint.

5 Underconstrained models

This section describes the detection and repairing of an underconstrained situation from a model. An underconstrained situation has been found by the constraint solver when not all geoms can be merged, because of one or more non-rigid joints. In this situation, multiple *groups* of geoms exist in the model that do not have a fixed relative position and orientation. The relative DOFs between two of these groups can be found by fixing one of them, and then

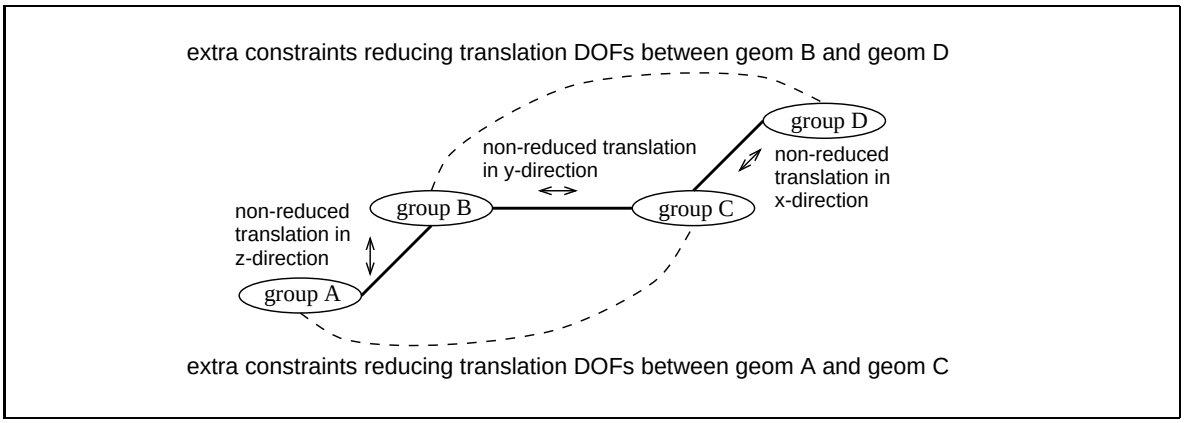


Figure 10: Introducing an overconstrained situation while repairing an underconstrained one.

solving all constraints between them using action analysis. The geoms of these groups are found by traversing the dependency graphs of the macro-geoms representing these groups.

Action analysis, however, solves constraints per joint, and is therefore not able to determine the relative degrees of freedom of two geom groups when multiple paths between them exist in the constraint graph. In such a case, a new joint, directly between the two groups, is added to the model. This joint introduces a loop in the model, and the locus analysis algorithm, see Section 2, is used to rewrite constraints from the other joints of the loop to the new joint.

After locus analysis has been performed, the new joint reduces the same DOFs as the other joints between the two geoms together. Then, action analysis is performed on the new joint which determines the relative DOFs left between the two geoms.

Underconstrained situations are repaired by adding constraints after the relative DOFs have been detected. When the relative DOFs of all pairs of geoms have been deduced, they are removed per pair by adding and solving constraints that reduce the DOFs between them. The DOFs have to be removed per pair of geoms, because otherwise an overconstrained situation could be introduced, since the relative DOFs of two different pairs of geoms can be the result of the same joint not reducing these DOFs. When the DOFs of this joint are reduced independently for multiple pairs of geoms, it could be done inconsistently.

An example of this is given in Figure 10. Four groups of geoms exists whose relative position and orientation have not been completely fixed. When the relative DOFs of group A with respect to group C, and of group B with respect to group D, are determined, they both contain a DOF that results from the joint between group B and group C. When this DOF is reduced by adding a constraint between group A and group C, and between group B and group D at the same time, an overconstrained situation is introduced into the model.

Currently, the constraints reducing the found DOFs are added by the user. The found DOFs could also be removed automatically using default constraints as suggested by SAK90).

6 Results and conclusions

To illustrate the approaches to solve over- and underconstrained models, an example of solving a model containing over- and underconstrained situations is given here. Figure 11 shows a

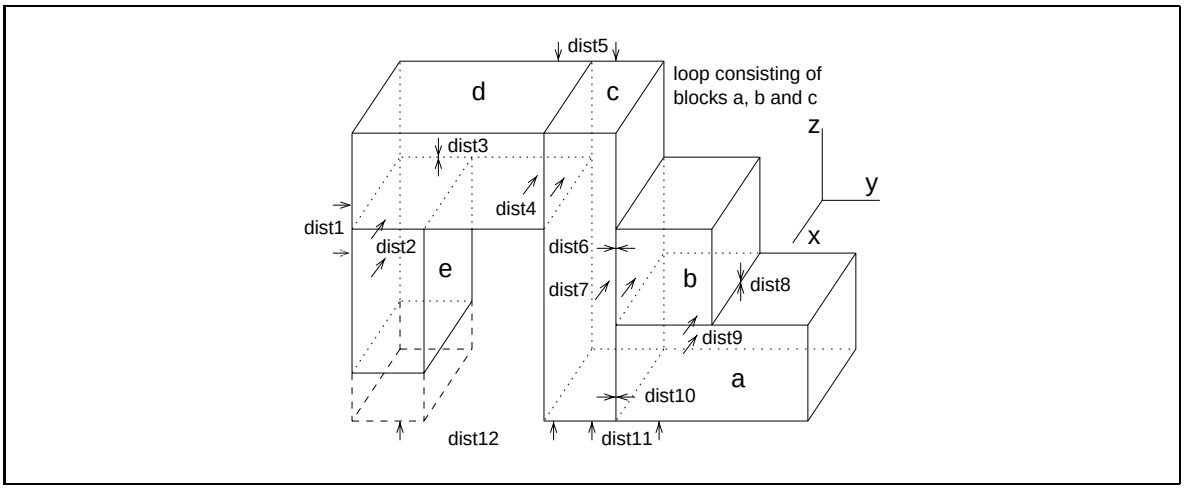


Figure 11: A model with under- and overconstrained situations.

model that consists of three blocks on the right hand forming a staircase, a horizontal block in the middle forming a bridge, and a block on the left hand that acts as a pier for the bridge.

The model contains a loop that consists of the blocks of the staircase, because they are not rigidly connected. The model further contains an underconstrained situation, because the bridge and its pier are not completely fixed with respect to the staircase. Finally, the model contains an overconstrained situation, because the height of the bridge part is not equal to the height of block c of the staircase.

The model is solved by first merging the blocks d and e into block ed, because they have a fixed joint between them. After that, the model contains a rigid loop consisting of block a, block b and block c. This loop is solved by taking block c fixed and then rewriting the dist8 and dist9 constraint from the joint between block a and block b to the other joints of the loop, making them rigid. As a result of this, the blocks of the loop are merged into block abc, see Figure 12.

After this, the model consists of two blocks that are connected by an under- and overconstrained joint, because there is no constraint between them reducing translation in the y-direction, whereas there are two inconsistent constraints between them reducing translation in the z-direction. The constraints involved in this overconstrained situation can be found in the dependency graphs of the blocks, see Figure 13. In this case only block ed contains an involved constraint whereas block abc does not because the constraints between block ed and block abc that are involved in the overconstrained situation work on its sub-block c.

Let us assume that constraint dist12 has been chosen and removed so that the overconstrained situation has been repaired. After that, the underconstrained DOFs are determined by taking one of the blocks fixed, the other free, and then performing action analysis on the constraints of the joint between them. The DOFs left of the free geom after the constraints have been solved are the underconstrained DOFs. Then constraint dist13 is added that reduces the underconstrained DOFs, and the last two blocks can be merged into block abcde, see Figure 14.

In this paper, a new approach has been described to solve over- and underconstrained models. The approach has been implemented in the geometric constraint solver that is used in the multiple-view feature modeling system SPIFF (DdB96; KDB97), but it can be used with

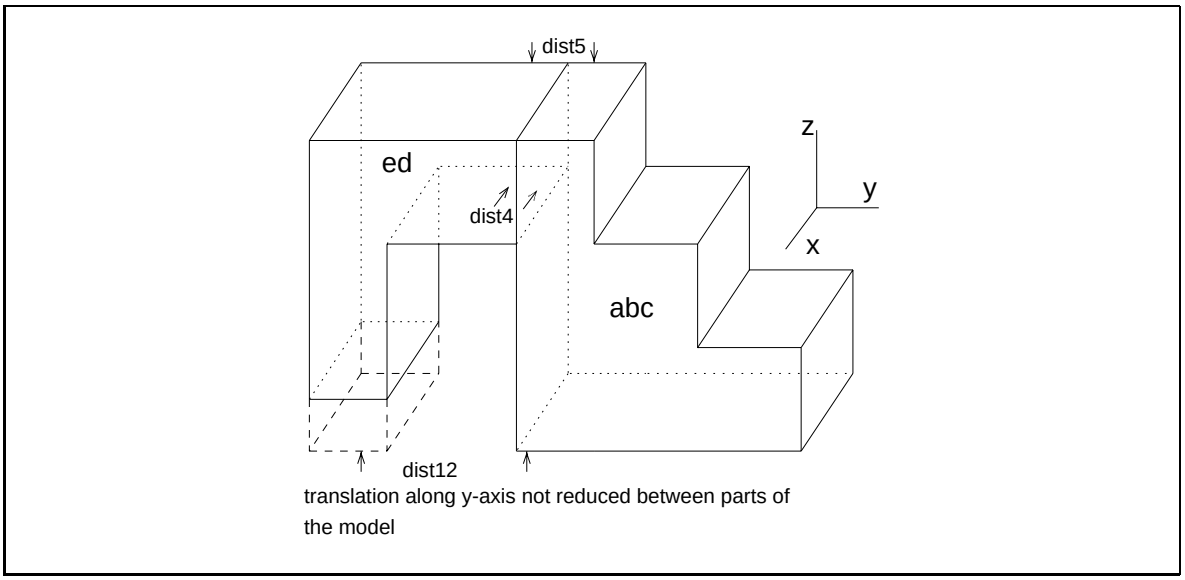


Figure 12: The model of Figure 11 after blocks with rigid joints have been merged, and loops have been solved.

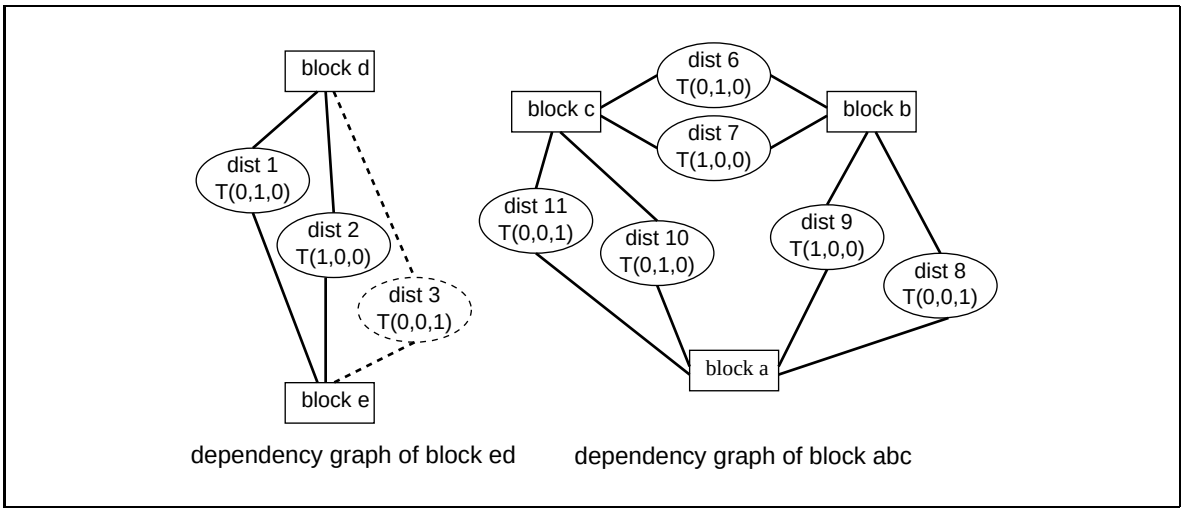


Figure 13: The expanded dependency graphs of the blocks of the model of Figure 11 when the overconstrained situation is found.

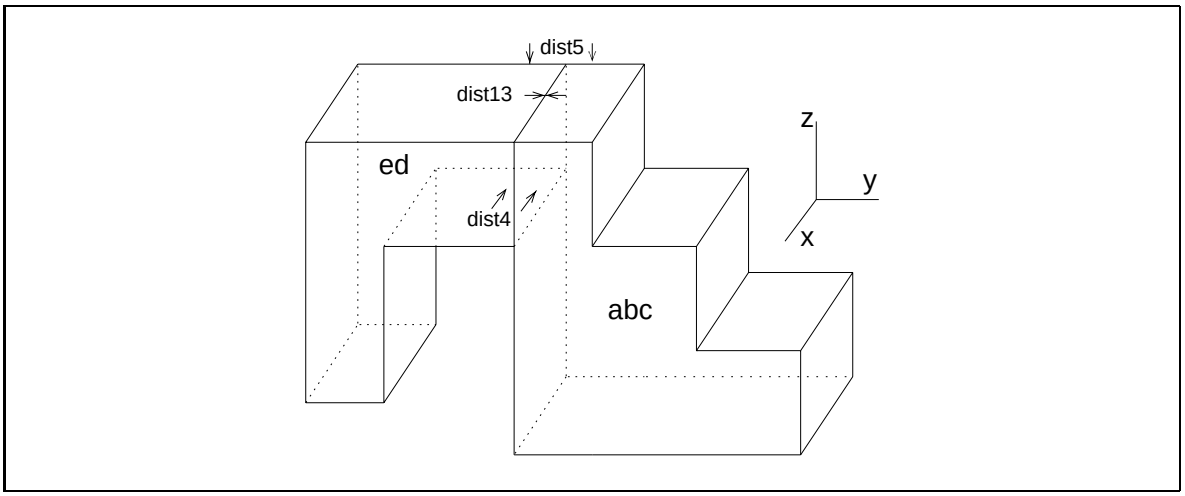


Figure 14: The model of Figure 11 after over- and underconstrained situations have been repaired.

all non-iterative constraint solvers. Future extensions to the presented approach include the following.

When a constraint is added to an underconstrained model to repair an underconstrained situation, this constraint should only reduce the underconstrained DOFs, because otherwise an overconstrained situation will be introduced in the model. To prevent this, the number of DOFs reduced by the constraints has to be taken into account by the system.

In case a constraint is removed from an overconstrained model, it should be done in such a way that no underconstrained situation is introduced. To achieve this, it should be tested whether the constraint to be removed only reduces the overconstrained DOFs, or also reduces other ones. If it does, the constraint should be replaced by a constraint that only reduces the other DOFs, so no underconstrained situation is introduced.

The system could be further improved by enabling it to automatically choose a constraint to be removed, in case of an overconstrained situation. The selection of the constraint could be done using several different criteria, such as removing the constraint that reduces fewest DOFs, removing the constraint that causes the least work to re-solve the model, and adding priorities to constraints and removing the constraint with the lowest priority.

As a final conclusion it can be stated that the capability to solve over- and underconstrained models has been shown to considerably enhance the usability of our DOF constraint solver.

References

M Dohmen, K J de Kraker, and W F Bronsvoort. Feature validation in a multiple-view modeling system. In J McCarthy, editor, *CD-ROM Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference.* ASME, August 1996.

C Hsu and Beat Brüderlin. A graph based degrees of freedom analysis algorithm to solve geometric constraint problems. In W Strasser, R. Klein, and R. Rau, editors, *Geometric modeling: theory and practice.* Springer-Verlag, 1997.

W T Keirouz, G A Kramer, and J Pabon. Exploiting constraint dependency information for debugging and explanation. In *Principles and practice of constraint programming; The Newport Papers*. MIT Press, Cambridge, Mass., 1995, April 1993.

G A Kramer. *Solving geometric constraint systems: a case study in kinematics*. The MIT Press, 1992.

K J de Kraker, M Dohmen, and W F Bronsvoort. Maintaining multiple views in feature modeling. In C Hoffmann and W Bronsvoort, editors, *Solid Modeling '97, Fourth Symposium on Solid Modeling and Applications*, pages 123–130. ACM Press, 1997.

H Suzuki, H Ando, and F Kimura. Geometric constraints and reasoning for geometrical CAD systems. *Computers & Graphics*, 14(2):211–224, 1990.

W Sohrt and B D Brüderlin. Interaction with constraints in 3D modeling. In J Rossignac and J Turner, editors, *Proceedings Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 387–396. ACM Press, 1991.