Modelling of Geometric Constraints in CAD-Applications

Manfred Rosendahl[1] , Roland Berling[2]

Introduction

Modelling in CAD-applications demands a mapping of the structure of the geometric objects and a description of the relation between them. In purely algebraic modelling there are equations between the elements of the geometric model defined, which are solved to find the characteristic points of the model[SeGo87]. Another way is to use geometric reasoning mechanism in which the dimensions and geometric relationships are defined as either facts or rules [RSV89,VSR92]. To model the relations constraint systems are well suited. In this paper we describe two methods.

The first approach, a type of ***constructive*** modelling, describes the geometric and non-geometric relations by construction operations, which will be used in building and modifying a geometry. The result is a mapping which describes the construction completely by a set of parameters. The modifying of dependent values is done either by reverse operations or by solving an equation system. Additional constraints can be defined on top of a given construction. Therefore constructions which need circular definitions can also be defined. This constructive approach, used in the **RelCAD** system, reduces the number of equations which have to be solved. Another feature which has been introduced to the constraint based geometric modelling is the Segment-concept. In

[1] Institut für Informatik, Univerität Koblenz, Germany
[2] Hewlett Packard, Böblingen, Germany

normal CAD-systems collections of elements, called group, block or segment are either programmed or the parameters can only be insertion point, angle, size. In the **RelCAD**-System a segment is a collection of objects that not only logically belong together, but also have geometric/numerical relationships among each other. They can even have relationships to objects outside the segment and can thereby be dependent on them. Any object of a segment definition can be treated as a parameter, not only dimensions or points but also other items and even segments [DRB93,RBD96].

In this approach the a model is developed in which the directions of the propagations are fixed. To evaluate a model against the direction in the constraint graph new constraints are added, which result in a graph with cycles. The second approach, a ***declarative*** modelling, describes the geometric relations by basis-constraints on points and dimensions as variables. The resulting constraint-graph will be evaluated by local propagation. The specific type of modelling makes it possible to use constructive operations, when evaluating the graph from any sufficient set of given variables. By this manner numerical methods like iterations can be avoided in most cases. This technique is developed out of the first approach and is suited for the early stages in a design process. This works especially because first a model with an undirected graph is developed, from which the free and derived variables can still be chosen. [BeRo93, Ber96].

Constructive Modelling: RelCAD Geometric modelling system

The **RelCAD** system was developed at the University of Koblenz as an application extension to the general 2D CAD-system **VarioCAD**, also developed there.

Each element of the geometric model is either an absolute one or defined by its relations to other geometric objects. So any changes in an object cause changes in all objects, which directly or indirectly depend on them. This is achieved because the dependencies built a directed acyclic graph (DAG). The method for changing an object, which is not a leaf in the graph, will be explained later.

The following objects are defined as absolute geometric and non-geometric objects:

Value (dimension), e.g. co-ordinate, length, distance, radius, angle but also non-geometric e.g. engineering values,
Point,
Line,
Circle,
Arc.

The relationships between the objects are typed according to these absolute object types. Only Arc is a subclass of Circle and can be used if a circle is required.

From these absolute classes, which have no relations the constructive classes with relations are derived. The relations are defined in all directions e.g. a point can be derived from geometric elements and also geometric elements can be derived from points.

Some examples of classes which are defined by the corresponding absolute class:

- Value:
  Co-ordinate, Distance, Angle, Expression
- Point:
  Intersection point, Tangent point, Centre point, Endpoint, Relative-point
- Line:
  Tangential line, Perpendicular Line, Parallel Line, Axis parallel line
- Circle:
  Given by centre and Radius, Tangential to 2 lines or circles and radius, Tangential to 3 lines or circles,
- Arcs:
  same arc circles

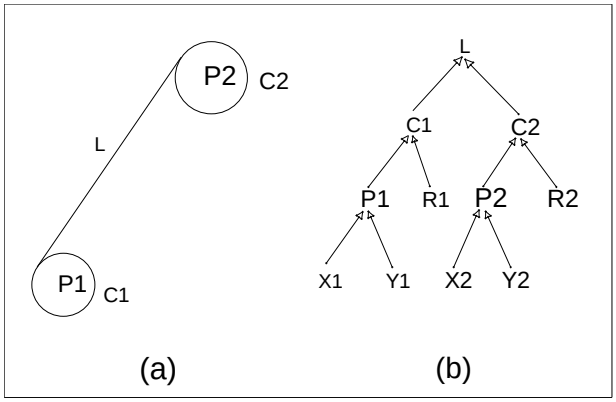

Figure 1

Figure 1 shows the definition of a derived element of the type Line_2objects. This element has 2 support elements C1 and C2.

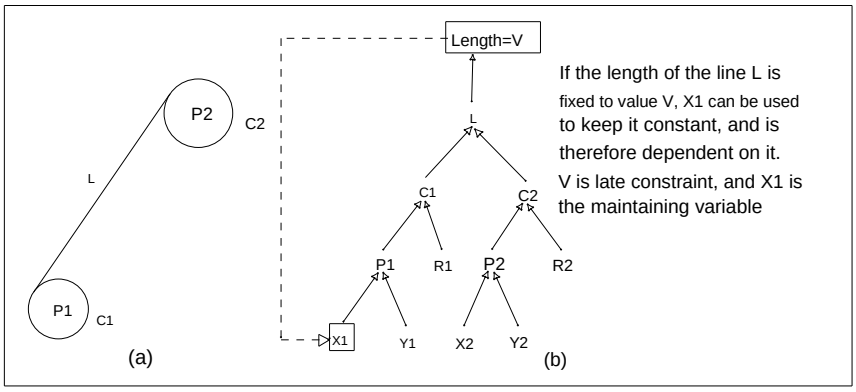

Figure 2

Up until now we have only discussed definitions with a-cyclic dependencies. There is also a class, which deals with cyclic dependencies. It is used to set extra dimensional restrictions on derived objects. For example, it could be used to fix the length of the line *L* in figure 2 This kind of constraint is called **late constraint,** because it is mostly assigned to the objects after they have been created. We call the derived object to which the late-constraint is assigned a **late-constrained-object**. A late constraint causes cyclic dependency among the geometric objects, because as an extra independent dimensional constraint it influences

the original dimension of a late-constrained-object and thereby the support objects of it, which again determine the original dimension.

For example, if the length of the line L in figure 2(a) is fixed, the positions and radii of both circles *C1* and *C2* are also restricted. Changing of them leads to a irregular value for the length of the line L which does not agree with the fixed length.

Figure 2.(b) illustrates the situation of cyclic dependency. In order to maintain the consistency of the late constraint a free dimension which directly or indirectly supports the late-constrained-object should be selected as a maintaining variable. A maintaining variable of a late constraint has the following tasks:
1) When the value of the late constraint has changed the maintaining variable should also be changed so that one of the support objects of the late-constrained-object gets the new data which leads to a consistent late constraint.
2) When the support objects of the late-constrained-object have changed the maintaining variable should be changed so that the late constraint remains satisfied.

A dimension is no longer free if it is selected as the maintaining variable of a late constraint. Figure 2(b) shows an example where the coordinate X1 is selected to be the maintaining variable if the length of the line L is fixed to be the value V.

The evaluation of a late-constrained-object is more complicated than the evaluation of other geometric objects because of the cyclic dependency among the late-constrained objects, its late-constraint, and the maintaining variable of the late-constraint. An iterative procedure is used to find an appropriate value for the maintaining variable to satisfy the late-constraint. More than one late constraint can exist in a model at the same time and some of them may be related to each other when the maintaining variable of one late constraint is the support object of another late-constrained-object. The related late constraints should be satisfied simultaneously, which means we have to solve a set of (non-linear) constraint equations, where the unknowns are the maintaining variables. For each geometric model there is one process to satisfy all late constraints (independent and related) by using the Newton-Raphson iterative method. When evaluating a geometric model this process will not be run until all other objects of the model have been evaluated.

Because the late constraints are necessary only when the geometric model can not be constructed in sequential order any more and because

they are assigned to the model after the main part of the model has been sequentially constructed, the number of the constraint equations which have to be solved simultaneously can be reduced by the user.

Late constraints are also used to change the data of conditioned objects, that means objects, which are not a leaf in the relation graph. If the user wants to change such an object, he has to choose, which independent objects(leafs) should be given free. Then a temporarily late constrained is solved to find a solution with the new values of the dependent objects.

Parametric design using segments

A family of design parts with various dimensions are often needed during the design. The concept segment serves this purpose.

About segments

The concept of the segment is similar to the concept of the *procedure* in high-level programming languages, *e.g.* Pascal. Therefore we introduce our segment by comparing it with the procedure concept.

The segment has two related aspects: segment scheme and segment instance. Segment scheme describes the inner structure of the segment. It determines what the components of the segments are, *e.g.* the number of the components, the type of each component and the relationships among the components. A segment instance is a graphical realisation of a segment schema. It is derived from the segment scheme and the actual parameters and is an instance and a graphical representation of the segment. Segment schemes correspond to *procedure declarations* and segment instances correspond to *procedure calls*.

In the following discussion we just use the word 'segment' when it is not necessary to differentiate the segment scheme from the segment instance. We say 'the components of the segment' when it is not necessary to know exactly how the components are defined within the segment scheme and the segment instance.

Another significant feature of the segment is that it can also have parameters. Similar to the parameters of the *procedure* there are two different forms of parameters for the segment: formal parameters and actual parameters. The formal parameters exist in the segment scheme and determine the types and the sequences of the actual parameters.

The actual parameters do not belong to the segment instance, but are used by the segment instances to determine the size, the variational shape and the position of the segment instance. The actual parameters define the relations of the segment to the rest of the model.

Each *procedure* can be called many times in a program. Thus, a segment scheme can also be associated with many segment instances, which means that some segments may have different graphical representations. This can happen, when the data of the actual parameters are different for each segment instance with the same inner structure.

The class segment

The graphical representations of the segments are called 'segment instances' because they could be treated as the instances of the class 'segment scheme' from the point of view of the object-oriented methodology. In our approach we define these two concepts as separate classes and set up a connection between them.

Two new classes are defined. They are also called segment scheme and segment instance. A segment scheme contains
(a) a list of formal parameters, and
(b) a list of components.
The component objects are mostly related to each other and some of them have relationship with formal parameters.

A segment instance contains
(a) a list of actual parameters,
(b) a corresponding segment scheme
Segment scheme and segment instance are also treated as classes (or types) like the other classes. An instance of the segment scheme class is a concrete segment scheme with a definite number of geometric objects as components and a number of formal parameters of certain types. A concrete segment scheme does not appear in the geometric model. An instance of the segment instance class is the representation of a certain concrete segment scheme in the geometric model. Figure 2 shows the segment scheme class, segment instance class and their instances.

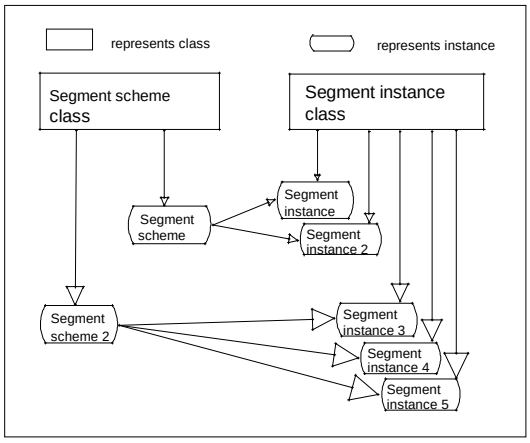

Figure 3

In the following discussion we call the instance of the segment scheme class **segment scheme** and the instance of the segment instance class **segment instance** when there is no misunderstanding.

The parameters of the segment

The formal parameters can be either
(1) of any basic type, *e.g.* value, point, line, circle or arc.
(2) of a segment instance.
Formal parameters are support objects of some component objects in a segment scheme. Therefore they determine the graphical data of these component objects.

An actual parameter must be of the same type with the corresponding formal parameter or a type derived from it, *i.e.* a object in the same absolute object-class. Therefore an actual parameter can be a derived object. Actual parameters do not belong to the segment instance. They are local geometric objects in the geometric model. If the formal parameter is a segment instance the actual parameter must be an instance of the same segment schema.

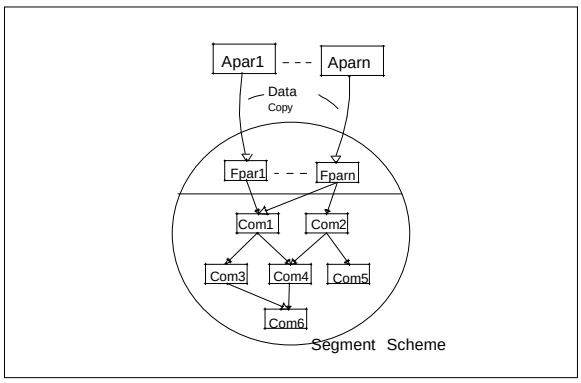

Figure 4

This is also the way compatible elements are defined.

Definition: 2 elements are compatible if they are either

- Derived from the same absolute object class
- Instances of the same object sheme

The data of the actual parameters determine the size, the variational shape and the position of a segment instance through formal parameters. The computing process of a segment instance does the following operations:
(a) It copies the graphical data of the actual parameters into the formal parameters of the associated segment scheme;
(b) It runs all the computing processes of the component objects in the associated segment scheme to generate the graphical data for each component;
(c) It returns all the graphical data obtained in (b) to the segment instance.

Figure 4 shows the actual and formal parameters and the segment scheme. The arrows within the segment scheme illustrate the supporting relation of component objects. With different actual parameters variations on the segment scheme can be generated. When a segment instance uses local objects of a model as actual parameters it is fully embedded in the geometric model.

The external object of the segment

Although the component objects of the segment are geometrically related to each other it is still possible for them to access the objects out-

side the segment, which means that a component object of a segment has objects outside the segment as its support objects. Such 'outside' object is called **external object**. This is equivalent to the global references in procedures. In procedures there is no method (but also no need) to access the local variables from outside. But in CAD accessing component objects of a segment from outside must be possible. An access to a component object of a segment is realised as an access to a substitute object of this component object. Also segments with alternatives and iterations are realised in the RelCAD system. For a more detailed description of the segment-concept see [RMD96].

Currently the concepts of the RelCAD system are transferred to a 3D system. Here new 3D oriented relations have to be defined, e.g. a cone defined by an apex point and a tangent sphere or a cone defined by two tangent spheres. Moreover a lot of 3D geometry is defined by 2D contours, e.g. a 3D object defined by the sweep of a 2D contour. Here the concepts of the 2D system can be used for defining the contour.

Declarative Modelling

In contrast to the constructive approach, the declarative approach describes the relations by a set of equations. Geometric relations between the lines, circles, etc., will be reduced to a few basis-relations between points and dimensions. The constraint model contains a set of basic geometric elements like:

- Segment(P1, P2): line segment from point P1 to P2
- Circle(M, r): circle centre point M and radius r
- Arc(P1, P2, M, r): arc from point P1 to P2 with centre M and radius r

and a set of constraint relations like:

- dp(P1, P2, d): distance between point P1 and P2 is d
- dl(P, Pa, Pe, d): distance between point P and line from Pa to Pe is d
- a3(P1, P2, P3, α): the angle between the line P1 to P2 and the line P1 to P3 is α
- a4(P1, P2, P3, P4, α): the angle between the line P1 to P2 and the line P3 to P4 is α
- equ(<expression>): the value of the expression is 0. If the expression contains n variables, it must be possible to compute one variable if the other n-1 variables are known.

- fix(<value>): a constraint without input values and just one output value.

Constraints between the geometric elements can be described by one or more of these basic constraints. Constraints with more than one degree of freedom are also described by a graph with these basic constraints.

This model in figure 5 can be derived by the following constraints:

dp(M1,T1,r1), dp(M2,T2,r2), dl(M1,T1,T2,r1), dl(M2,T2,T1,r2), circle(M1,r1), circle(m2,r2), line(T1,T2)

Non-geometric constraints can be described, too. For example, expressing that the length of the tangent-line is related to the sum of the radii by a factor f can be done by the following constraints:

dp(T1,T2,d), equ(d-(r1+r2)*f)

The constraint system is represented by an undirected graph $G(V,E,\varnothing)$ with nodes $V=P\cup D\cup C$ representing the points (P), the dimensions(D) and the constraints(C), and edges $E\subset C \times (P\cup D)$ representing the constraints bindings. There are 3 types of edges between a constraint and a variable.

none: undirected edge.

c_in: directed edge from the variable to the constraint.

v_in: directed edge from the constraint to the variable.

To define algorithms on the structure graph, we need operations to define the edges of certain types adjacent to a node. Therefore we define the following mappings:

Λ: $V \times \text{typeId} \rightarrow P(E)$: set of edges of typeId adjacent to node V

Γ: $V \times \text{typeId} \rightarrow P(V)$: set of nodes adjacent to V via edge of typeId

δ: $V \times \text{typeId} \rightarrow N_0$: number of edges of typeId adjacent to node V

With respect to the orientation of the edges the following mappings are defined:

Λ_{in}: $V \rightarrow P(E)$: set of oriented edges incoming to a variable or a constraint .

Λ_{out}: $V \rightarrow P(E)$: set of oriented edges outgoing from a variable or a constraint .

Λ_{none}: V $\rightarrow$ P(E): set of non-oriented edges adjacent to a variable or a constraint .

Analogously the mappings $\delta_{in}(v)$, $\delta_{out}(v)$ and $\delta_{none}(v)$ are defined as the number of edges adjacent to v with the equivalent predicate. The mappings $\Gamma_{in}(v)$, $\Gamma_{out}(v)$ and $\Gamma_{none}(v)$ are defined as the sets of nodes adjacent to v via an edge with the equivalent predicate.

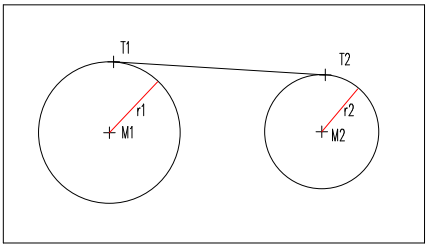

Figure 5

Figure 6 shows the graph for the example in figure 5.

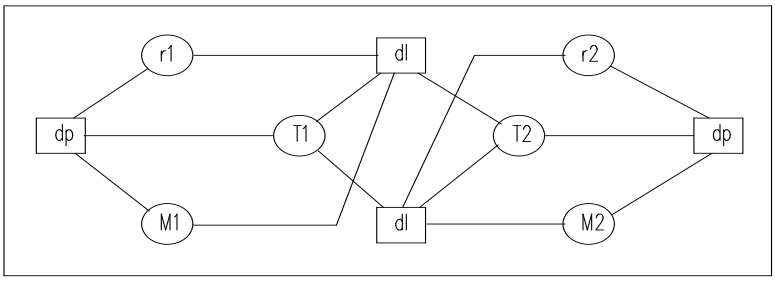

Figure 6

The undirected graph unveils the structure of the possible equation systems which can be formed by instantiating new values for parameter vertices or changing old ones. To fix a dimension or point, one respective two constraint fix with an edge to the fixed object will be added. By means of the constraint satisfaction process the undirected graph is transferred to a directed graph, which represents the evaluation of the geometry. This evaluation can be done by local propagation as long as there are no cycles in the directed graph, otherwise iterative methods have to be used. Therefore the goal of the orientation is to obtain as few cycles in the graph as possible.

The oriented graph has to fulfil the following restrictions.

$\forall\, c \in C: \delta_{out}(v) = 1$

$\forall\, v \in P: \delta_{in}(v) = 2$

$\forall\, v \in D: \delta_{in}(v) = 1$

Each point needs two values to be determined and each dimension one. These are the incoming edges. The values are then propagated through the outgoing edges. Each variable has a structural degree of freedom (sdf), which gives the degree of freedom, that is not yet restricted by constraints, i.e. by incoming edges.

$\forall\, v \in P: sdf(\{v\})=2-\delta_{in}(v)$

$\forall\, v \in D: sdf(\{v\})=1-\delta_{in}(v)$

$\forall\, M \subseteq P \cup D: sdf(M)=\sum_{v \in M} sdf(\{v\})$

In an undirected constraint graph $G(V,E,\varnothing)$,$V=C \cup P \cup D$ exists a complete matching of the set of constraint nodes C in the set of variable nodes $P \cup D$ if and only if to each constraint subset $N \subseteq C$ in the adjacent variable set there exist at least as many structural degrees of freedom as there are elements in N.

The constraint satisfaction process determines (in three phases) a complete allocation of the constraints to the variables. The allocation is defined by an orientation of the edges. The method identifies subgraphs which are structurally over-determined. To determine the allocation graph, algorithms to compute maximal matching [MH90] can be used. Serano and Gossard use this in their approach [SeGo87]. The method here uses propagation to get the smallest constraint graph on which a matching algorithm has to be applied. The allocation of a constraint to a variable will be expressed by the edge attribute v_in. All other edges adjacent to this constraint are then input values, which are attributed c_in.

A constraint graph is valid, if the following holds:

- Constraint node c: At most one v_in edge. If there is one v_in edge, all other edges are oriented.
- Variable node v: At most as many v_in edges as there are degree of freedom. If there are exactly as many (sdf(v)=0), then all edges are oriented.

During the propagation process the graph has to be always valid. The three phase of the method are:

- Propagating the constraints
- Propagating the degrees of freedom
- Determining the maximal constraint matching

The first two steps are used to minimise the constraint set, which has be applied to step 3. The propagating of constraints is continued as long as there are constraints, which can be mapped to one variable. The propagating of the degrees of freedom is continued as long as there are variables, which have more degrees of freedom as there are still non mapped constraints in the adjacent constraint-set. For the remaining not yet assigned constraint set an assigning is made by the maximal matching method.

Propagating of Constraints

Algorithm p-con:

```
K := {c∈C | δnone(C )=1}
while K≠∅ do begin
  c:= extract-any(K);
  for e∈Λnone( c) do e.eo := v_in;
  for v∈Γout( c) and sdf(v)=0 do begin
    for all e∈Λnone( v) do e.eo:=c_in;
    for all c∈Γout( v) and δnone( c)=1 do K:=K∪{c};
  end;
end;
```

K starts with the set of all constraints of the type fix. With this K the computation begins. The edges are directed to the adjacent variable v. If the v has no more structured degree of freedom, all other non-oriented edges in v are directed as outgoing. If a constraint c adjacent to v has now only one non-oriented edge, c is included in K.

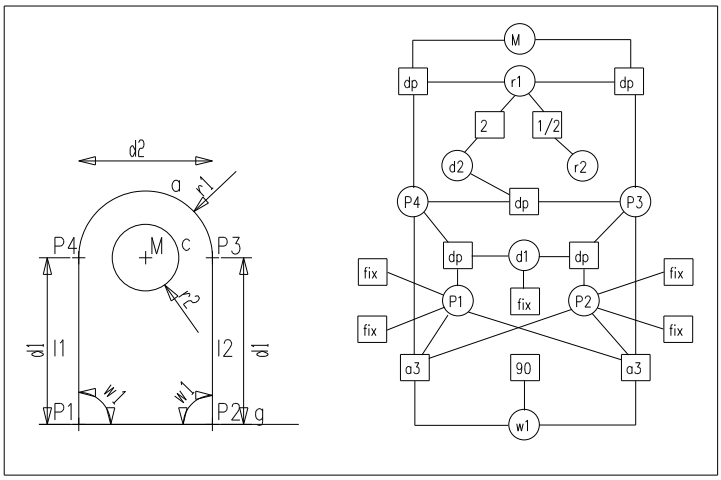

Figure 7

To show the method, we give an example. Figure 7 shows the model with the undirected graph. The lines l1 and l2 have the same length and are perpendicular to g. The radius r2 of c is half of the radius r1 of c. The arc a is tangential to the lines l1 and l2.

The algorithm starts with K built from the constraints of the types fix and 90-degree, which is a special type of fix. Figure 8 shows the propagating. After applying the constant-constraints of P1 and P2, the angles w1 and d1 no longer have a structured degree of freedom, so the remaining edges can be oriented as c_in-edges to the adjacent dp- and a3-constraints, whose constraint-nodes are then included in the set K (left). The propagating of these constraints allows it to include the dp-constraint between P3 and P4 in K (middle). Its propagating allows it to propagate the constraints holding the radii-relations. At the end the constraints adjacent to M can be propagated (right).

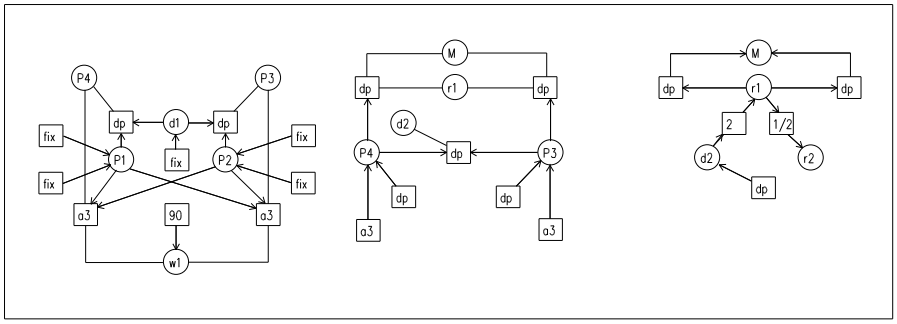

Figure 8

After applying p-con there are no constraints, to which variables can be applied in a deterministic manner. The remaining sub-graph is invariant to the order in which the constraints were chosen of K.

Propagating the degree of freedom

```
algorithm p-sfd
K := {v∈V | 0<δnone(v )<=sdf(v)}
while K≠∅ do begin
  c:= extract-any(K);
  for all e∈Λnone( v) do e.eo := v_in;
  for all c∈Γin( v) do begin
    for all e∈Λnone( c) do e.eo:=c_in;
    for all w∈Γin( c) and ϑnone( w)<=sdf(w) do K:=K∪{w};
  end;
end;
```

K is the set of variables for which edges can still be directed. The edges from the adjacent constraints are oriented as v_in. The remaining edges of the adjacent constraint can now be oriented as c_in. The variables which are adjacent over this c_in edges are included in the set K, if their degree of freedom can now be propagated. After applying the p-sdf algorithm we get a valid constraint-graph which has no sdf, which ca be propagated. The resulting graph is independent of the order in which the nodes are extracted from K.

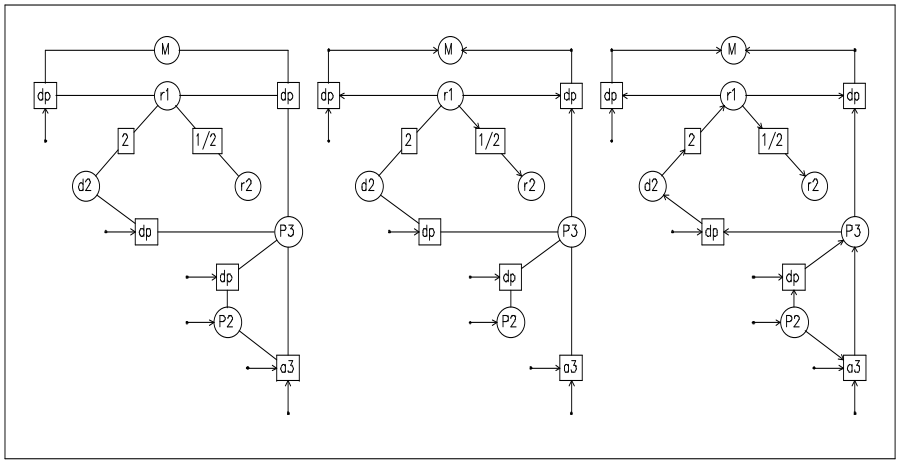

Figure 9

To illustrate the method, we continue the above example. The left part of figure 9 shows the rest of the non-oriented graph after propagating the constant constraints w1, P1 and P4. The arrows represent the oriented edges, which are not shown. At the start K={r2,M}, because only r2 and M meet the requirements for propagation. The middle part shows the constraint graph after propagating r2 and M. The adjacent constraints are assigned and the edges oriented. The variable r1 is included in K. After propagating r1, d2 meets the requirement and then P3.The right part shows the complete oriented constraint-graph after propagating P3.

Determining the maximal constraint matching

The algorithm to determine the matching is adapted from the Edmonds method for bipartite graphs [MH 90].

```
algorithm match

R := {c∈C | δnone(c )≠0}
for all c∈R do
for all e∈Λnone( c) do e.eo:=c_in;
while R≠∅ do begin
  c:= extract-any(R);
  if find/c,v) then apply(c,v);
end;
```

In the first step all incident non-oriented edges are oriented as incoming edges. By this the graph becomes completely oriented. To find a path in the graph, which extends the matching find(c,v) looks for a variable, which still has an sdf. Then apply(c,v) inverts the orientation on the path from c to v.

After these three steps we get a graph which is completely oriented, and the variable and constraint nodes have a valid orientation. This graph can be propagated in topological order, if there are no cycles. If there are cycles, the variables on the cycles must be solved simultaneously by solving an equation system either algebraically or with iteration. The propagation outside the cycles is possible, because the constraints can always be computed with any edge as outgoing edge.

Conclusions

Two systems for constraint geometric design were explained. The first system differs from other systems mainly by the contained segments and the introduction of segments with alternatives and with repetitions. The second system easily allows changing the input and output variables of a geometric system.

References

[BeRo93] Berling R., Rosendahl M., 'Geometry Modelling using Dimensional Constraints'. In CARs&FOF'93, 9th International Conference on CAD/CAM, Robotics & Factories of the Future, Newark, NewJersey, USA, August 18-20,1993

[Ber96] Berling R. 'Eine Constraint-basierte Modellierung für Geometrische Objekte', Dissertation,1996.

[DRB93] Du C, Rosendahl M. and Berling R., 'Variation of Geometry and Parametric Design', Proc. 3rd. International Conference on CAD and Computer Graphics, Beijing, Aug. 23-26, 1993, pp 400-405, International Academic Publishers, 1993

[MH90] McHugh J., 'Algorithmic Graph Theory', Prentice-Hall, New Jersey 1990.

[RBD96] Rosendahl M., Berling R.,Du C., 'A Generalised Segment Concept',Proceedings of the Dagstuhl Seminar 'CAD Tools for Products' August 95,Springer.

[RSV89] Roller D., Schonek F. and Verroust A., 'Dimension-driven Geometry in CAD: a survey' in Strasser W. and Seidel H.-P. (Eds.) Theory and Practice of Geometric Modelling, Springer Verlag, 1989, pp 509-523.

[SeGo87] Serrano D. and Gossard D., 'Constraint Management in Conceptual Design' in Sriram D. and Adey R.A. (Eds.) Knowledge Based Expert Systems in Engineering: Planning and Design. Computational Mechanics Publications, 1987.

[VSR92] Veroust,A. Schonek,F., Roller,D., 'Rule oriented Method for Parametrized Computer-Aided Designs, CAD, Vol.24, No.10, pp531-540.