1991

R.C. Veltkamp, F. Arbab

# Geometric constraint satisfaction

Computer Science/Department of Interactive Systems Report CS-R9141 September

# Geometric Constraint Satisfaction

Remco C. Veltkamp　　Farhad Arbab

CWI, Department of Interactive Systems
Kruislaan 413, 1098 SJ Amsterdam, the Netherlands
e-mail: {remco,farhad}@cwi.nl

**Abstract**

This report presents an overview of geometric constraint satisfaction techniques and a new incremental approach that is suitable for interactive design. This approach categorizes solutions into so-called quanta. A quantum is a range of solutions with uniform geometric characteristics. Using this approach a system can keep the intermediate solutions in the geometric domain, so that new geometric constraints can be interpreted on the same high level of abstraction. Propagation of information is performed by 'propagation of known states', and the information itself results from 'solution set inference', where a solution set is represented in geometrical form by unions and intersections of quantum labels.

This approach leads to a number of advantages, including graceful handling of under-constrained specifications, the ability to perform satisfaction locally and incrementally, support of constraint inference and geometric reasoning, and preserving the declarative semantics of constraints.

*1980 Mathematics Subject Classification (1985):* 68T20, 68U05
*1991 Computing Reviews Classification:*
I.3 [Computer Graphics] Graphics utilities, Computational geometry and object modeling.
J.6 [Computer Aided Engineering] Computer Aided Design.

*Key Words and Phrases:* geometric constraints, constraint satisfaction, geometric reasoning, computer aided design.

*Note:* this report is a compilation of [Veltkamp, 91a], [Veltkamp, 91b], and [Veltkamp and Arbab, 91].

## 1. *Introduction*

Constraints specify dependency relations between objects which must be satisfied and maintained by a system. Constraint systems are used in a wide range of applications, including user interfaces [Szekely and Meyers, 88], simulation [Steele Jr. and Sussman, 79], animation [Badler and Kamran, 87], and geometric modeling. Geometric constraints can fix one or more degrees of freedom for positioning, orientation and dimensioning. For example, when a circle of fixed radius is constrained to be tangent to a fixed line segment, the position of the circle center is restricted to two line segments parallel to the given one, at a distance equal to the radius.

A constraint satisfaction system relieves some of the burden of its users: problems can be solved by specifying constraints, users need not specify how to solve the constraints. It is easier to state constraints than to satisfy them. But even if a system cannot satisfy all constraints that can occur in a given domain, it can free its users from the error-prone process of solving the many little but time-consuming simpler problems.

Incremental satisfaction of constraints arises naturally in geometric design applications. A good deal of work has already been done on constraint satisfaction in general

Report CS-R9141
ISSN 0169-118X
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

[Mackworth, 77] [Leler, 88], and constraint satisfaction in the geometric domain in particular [Sutherland, 63] [Borning, 81] [Nelson, 85] [Rossignac, 86] [Arbab and Wang, 89] [Emmerik, 90] and [Ruttkay, 91]. Both areas are known to be difficult and no robust methods exist for sufficiently general applications.

Incremental satisfaction of geometric constraints poses additional complications in constraint problem solving. There are two important factors that set incremental satisfaction of geometric constraints (in the context of computer aided design or geometric modeling) apart from other constraint satisfaction methods:

- The constraint satisfaction system must perform useful work and provide *meaningful* feedback to its users, especially, in under-constrained situations.
- The feedback provided by the system must be primarily geometric.

While a design is in progress, it is natural to expect under-constrained situations to be predominant: often not enough information is available to make solutions more specific.

Geometric reasoning and constraint satisfaction are facilities that can be part of a basic layer of a design system, but this layer in itself is not an application system. It provides a good basis for CSG modeling and Euler operators [Mäntylä, 88], feature-based modeling [Pratt, 87], and intelligent human-computer interaction interfaces [Helander, 88].

The next sections are about the following topics:

- ▷ in Section 2 we provide an overview of satisfaction techniques and specific constraint systems.
- ▷ in Section 3 we present some problems that motivated our research.
- ▷ in Section 4 we introduce the quantum approach.
- ▷ in Section 5 we give a formal model of the quantum approach.
- ▷ in Section 6 we present the underlying propagation algorithm.
- ▷ in Section 7 we analyze the termination properties of the propagation algorithm.
- ▷ in Section 8 we discuss implied constraint inference.
- ▷ in Section 9 we discuss completeness.
- ▷ in Section 10 we illustrate the quantum approach by two non-trivial examples in 2D and 3D.
- ▷ in Section 11 we discuss the relationships between constraints and both the imperative nature and the information hiding principle of object-oriented programming.
- ▷ in Section 12 we give some concluding remarks.

## 2. Overview

We can classify satisfaction techniques into *structured* and *unstructured* methods. An orthogonal classification can be made into *numerical computation* and *deductive* techniques. Structured deductive techniques can exploit several methods for *propagation of results.*

In the Section 2.1 we describe these techniques in more detail, and in the Section 2.2 we mention some constraint systems that use these techniques.

### 2.1. Satisfaction techniques

#### 2.1.1. Unstructured methods

Unstructured methods do not group dependent constraints into sets, and do not solve the independent sets separately. Solving the *overall set of equations* comprising all constraints is perhaps the simplest way of constraint satisfaction. One way to solve the overall set of equations is by relaxation, see the following subsection.

Among the unstructured deductive systems are algebraic manipulation, logic programming, and term rewriting. *Algebraic manipulation* or symbolic algebra systems are able to solve complex sets of algebraic constraints at a symbolic level [Davenport et al., 88]. They are usually slow.

*Logic programming* can be classified into functional programming, based on equational logic (e.g. LISP), and relational programming, based on Horn clause logic (for example Prolog).

*Augmented term rewriting* as developed by [Leler, 88] supports three features in addition to term rewriting: abstract data types, typing of variables, and binding of values to variables. Abstract data types are defined by rules. The strong ties with equational logic gives augmented term rewriting a solid theoretical foundation. As a consequence the control mechanism is separated from the problem solving rules.

Obvious drawbacks of unstructured techniques are their computational complexity and their inefficiency for interactive applications: each single change leads to re-solving the whole set of constraints.

### *2.1.2. Structured methods*

Structured methods impose a structure on the set of constraints by grouping them into sets of dependent constraints. These sets are satisfied independently.

Numerical computation of a local set of constraints is often done by relaxation. *Numerical relaxation* makes an initial guess at the values of the variables in an equation, and estimates the error by some heuristic. The guesses are adjusted accordingly and the new errors are estimated. This repeats until the error is minimized. A disadvantage of this method is that it converges to only one of the roots of an equation. Moreover, *which* root is found depends on the initial value of the variable. This makes the solution unpredictable in an under-constrained situation. Numerical relaxation is also computationally expensive, and can be used only in continuous numeric domains. A form of relaxation is the Newton-Raphson iteration technique for finding the root of a function. It is faster than general numerical relaxation.

A set of dependent constraints can be represented as a *network* of constraints. Structured deductive systems use some method to assimilate results of the inference process throughout the network. We mention two such methods.

*Propagation of known states,* or just local propagation, can be performed when there are parts in the network whose states are completely known (have no degrees of freedom). The satisfaction system looks for one-step deductions that will allow the states of other parts to be known. This is repeated until all constraints are satisfied or no more known states can be propagated. If not all constraints can be satisfied, the remaining constraints must be resolved by, for example, numerical relaxation. Most constraint satisfaction systems use some form of local propagation.

*Propagating degrees of freedom* amounts to discarding all parts of the network that can be satisfied easily and solving the rest by some other method. This method identifies a part in the constraint network with enough degrees of freedom so that it can be changed to satisfy all its constraints. That part and all the constraints that apply to it are then removed from the network. Deletion of these constraints may give another part enough degrees of freedom so as to satisfy all its constraints. This continues until no more degrees of freedom can be propagated. The part of the network that is left is then satisfied by some other method, if necessary, and the result is propagated towards the discarded parts, which are successively satisfied (propagation of known states).

The above techniques are two methods of propagation, independent of *what* is actually propagated. We distinguish the following types of information inference and propagation.

*Solution set inference* makes deductions on the set of possible solutions, which are restricted by the constraints. Most frequently used are discrete value sets, or intervals of numerical values, see [Davis, 87].

In *single solution inference*, constraint variables get assigned a single value, often numeric. The *operational approach* [Rossignac, 86], [Arbab and Wang, 89] performs single *geometric* solution inference. It satisfies constraints sequentially by performing operations (translation, rotation, etc.) on the geometric objects involved. An already satisfied constraint either tolerates an operation on one of its operands, or must propose a transformation to satisfy the constraint again. In this way operations can be propagated through a constraint network until all operations are tolerated.

*Local algebraic expression* inference is a means to deal with loops in propagating numeric values. [Steele Jr. and Sussman, 79] used powerful algebraic manipulation, but found that these techniques are not powerful enough to solve many interesting problems that people can solve. The way people usually solve these problems is by organizing the solution so that simple canned algebraic solutions suffice.

In *constraint* inference, implied constraints are derived and explicitly added to the network. Implied constraints can be recognized by a unification mechanism or by the use of multiple redundant views [Steele Jr. and Sussman, 79]. Finding implied constraints can be used to avoid extensive manipulations in cases where local propagation does not suffice. Stating constraints in a different way with the same meaning can help the system to solve the constraints locally. Otherwise it may have to resort to techniques such as relaxation.

## *2.2. Constraint systems*

Some systems that deal with geometry and some general purpose constraint languages are mentioned below in chronological order.

Sketchpad [Sutherland, 63] was the first constraint-based drawing system. It satisfies constraints using propagation of degrees of freedom. When this fails, it resorts to relaxation.

Variational geometry [Lin et al., 81] translates dimensional constraints into an overall system of equations, which is solved numerically by the Newton-Raphson method. The dimensional constraints are defined by equations on coordinates of characteristic points. Each time a dimensional value is changed, the whole system of equations must be solved.

ThingLab [Borning, 81] enlarges the possibilities of Sketchpad with extensibility and object-oriented techniques, so that new classes of objects and constraints can be defined. It uses both propagation of degrees of freedom and propagation of known states.

Juno [Nelson, 85] is a simple system based on one geometric primitive: the point. It uses a Newton-Raphson iteration technique to solve constraints. The user must supply an initial value to start the iteration.

[Rossignac, 86] presents an operational interpretation of constraints in CSG modeling. Constraints are specified by the user in terms of relations between boundary features, and are transformed by the system into rigid motions of parts of the CSG tree. An under-constraint situation can simply not occur. The user must specify the order of evaluation, and is responsible for solving conflicts.

Real general purpose languages for constraint logic programming (CLP) can be used in a wide range of applications, but are usually limited in their satisfaction power in each specific domain. Most constraint languages are biased to a more specific domain: CLP($\mathbb{D}$) [Cohen, 90]. For numeric constraints this yields CLP($\mathbb{R}$) over the domain of real numbers [Heintze et al., 87]. This is not a symbolic algebra system; it uses no unification, but numeric deductions.

Bertrand [Leler, 88] is a rule-based system that uses augmented term rewriting. It has a form of abstract data types, which allows to define new types and constraints. It is a general purpose constraint language, but deals primarily with numeric constraints.

OTP (Operational Transformation Planning) [Arbab and Wang, 89] also provides an operational interpretation of constraints. It exploits solution reduction, one solution is presented in an under-constrained specification (single solution inference). The satisfaction process is planned through symbolic reasoning on the geometric level, that is, by geometric reasoning [Arbab and Wing, 85]. This can also involve the inference of implied constraints.

In [Emmerik, 90] constraints relate coordinate systems. Constraints between degrees of freedom (for example between the x- and y-coordinate because of a distance constraint) are evaluated after lower-order constraints (for example one that uniquely determines the x-coordinate). This is a form of delayed satisfaction. Selection among alternative solutions to constraints (single solution inference) is based on the minimal resulting disturbance. So, a single solution is derived for each constrained variable.

[Ruttkay, 91] presents a co-operative, or intelligent, graphical editor for constrained graphical objects. Solutions for the positions of the objects are represented in numerical form by a single range of values (numerical solution set inference). An explicit network of constraints and variables is derived from a symbolic representation when needed.

[Owen, 91] presents a system with single numeric solution inference, in which the numerical solutions are derived by algebraic methods. The propagation mechanism employed is 'propagation of degrees of freedom'.

## *3. Motivation*

In this section we mention some problems that motivate our current research.

Objects can be related to each other by constraints, without logically being involved in a part-whole relation. We do not want to be forced to create an artificial whole consisting of parts, in order to be able to express a constraint, as in ThingLab. Specifying constraints has a declarative flavor, and somewhat conflicts with part-whole hierarchies and information hiding, which are the hallmarks of object-oriented systems. The relation between constraints and objects is discussed in Section 11.

Many systems detect over-constrained situations, but cannot handle under-constrained cases. Indeed numerical methods cannot solve a set of under-constrained equations. A CLP system for solving numeric constraints returns a set of deduced equations that constitutes the solution. In a geometric context, however, we prefer a geometrical form of the solution. The operational approach and selection of alternatives based on minimal disturbance yield only one of the solutions. This may give rise to two problems:

- The proposed solution is as it was intended, but the user is not aware of the ambiguity of the specification. This may cause problems for post-processing.
- The solution is not as intended, and the user must interfere because the system cannot propose alternative solutions.

We distinguish several types of alternative solutions:

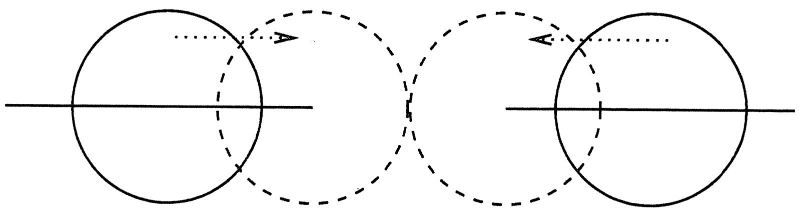


Figure 1: Simultaneous evaluation of mutually constrained circles.

1. Several objects meet a specification of a variable involved in a constraint, and one or more of them must be addressed. For example, *any* circle or *all isosceles* triangles are constrained.
2. A constraint can have several discrete solutions. For example a circle through two points, and with a fixed diameter greater than the distance between the points, can have two positions.
3. A constraint can have a continuous range of solutions. For example the locus of a point having a fixed distance to a fixed point, is a circle.

Whether or not the first type of alternative solutions can be handled depends on the programming environment, and is not within the scope of this report.

Some systems mentioned above deal only with numeric constraints. Most graphics-oriented constraint systems directly translate the (geometric) constraints into a set of numeric constraints or equations. Often only linear equations can be solved. Instead, we want to solve constraints at a high level of geometric abstraction. This allows powerful reasoning which can avoid some of the problems that occur with numeric constraints: recall that even a simple distance constraint results in a quadratic equation.

Simultaneous evaluation of mutually constrained objects is a difficult problem. Look at Figure 1 for an example. Each of the two circle centers must lie on a line segment; then, the circles are constrained to be tangent to each other on the outside. It is too complex for an operational technique to translate both circles along the line segment simultaneously to let them touch. Delayed evaluation is then preferred to relaxation in order to deal with a geometrically meaningful representation as long as possible.

In this report we focus on providing ranges of solutions and discrete alternative solutions in under-constrained situations by means of a quantum approach. In combination with geometric reasoning and delayed satisfaction, this gives a more powerful problem solving capability than usually provided by constraint systems.

## *4. Quantum approach*

We first give an informal presentation, but the main ideas are formalized in the subsequent sections.

Constraints work on constraint variables, which are geometric primitives such as a point, a line (segment), or a circle.

Our approach to geometric constraint satisfaction is based on reduction of both problem complexity and solution domain complexity by geometric simplification. Problem complexity reduction is achieved by expressing geometric primitives in terms of characteristic parts. For example, the position of a circle is reduced to the position of a point: the circle center. Constraint solving amounts to recording the solution of the constrained geometric primitives, in terms of their characteristic parts.

The unit of the solution domain is a 'quantum', which is itself a geometric primitive:

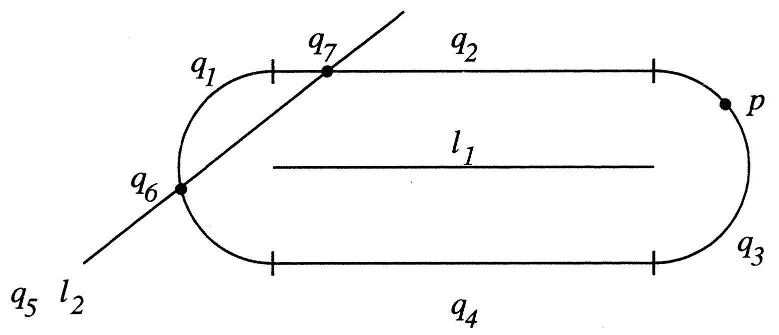


Figure 2: Quanta resulting from the constraints distance$(p, l_1, d)$ and on$(p, l_2)$.

> A quantum is a geometric primitive describing a part of the solution set with uniform geometrical characteristics.

We use the term quantum because it captures both 'interval' and 'region', and is more descriptive than 'solution set'. The nature of a quantum is that it describes a sharply bounded quantity of some phenomenon, in our case the geometric properties of a solution set. Note that quanta are not indivisible, nor need they be disconnected.

For example, the locus of a point $p$ having a fixed distance to a line segment $l_1$ consists of the union of two line segments and two half circles, see Figure 2. This solution is split into the quanta $q_1, \ldots, q_4$, representing the two line segments and two half circles. These quanta represent alternative sets of solutions. If $p$ is further constrained to lie on a line $l_2$ (another quantum), the final solution is restricted to the intersection of $l_2$ with all the alternative solutions so far. Since calculating the intersection of a line and a circle differs from intersecting two lines, we take advantage of the subdivision into quanta $q_1, \ldots, q_4$.

Some simple constraints (with a fixed second operand) and the associated quanta are listed below:

| primitive | constraint | primitive | quanta |
|---|---|---|---|
| point | distance | point | circle |
| point | distance | line segment | 2 line segments + 2 half circles |
| point | left/right of | line | half-plane |
| circle | tangent | line | 2 lines |
| circle | tangent | circle | 2 circles |

The same principle can be used in 3-space. In general, however, a solution set may not be representable with a fixed set of geometric primitives, especially in 3-space. In that sense the set of constraints and geometric primitives should agree with each other. A set of useful constraints in 3D is listed below and illustrated in figure 3 (again the second operand is fixed):

| | primitive | constraint | primitive | quanta |
|---|---|---|---|---|
| 1 | line | angle | line | cone |
| 2 | line | angle | plane | plane |
| 3 | point | distance | point | sphere |
| 4 | point | distance | line | cylinder |
| 5 | line | distance | line | cylinder |
| 6 | sphere | tangent | line | cylinder |
| 7 | sphere | tangent | sphere | 2 spheres |

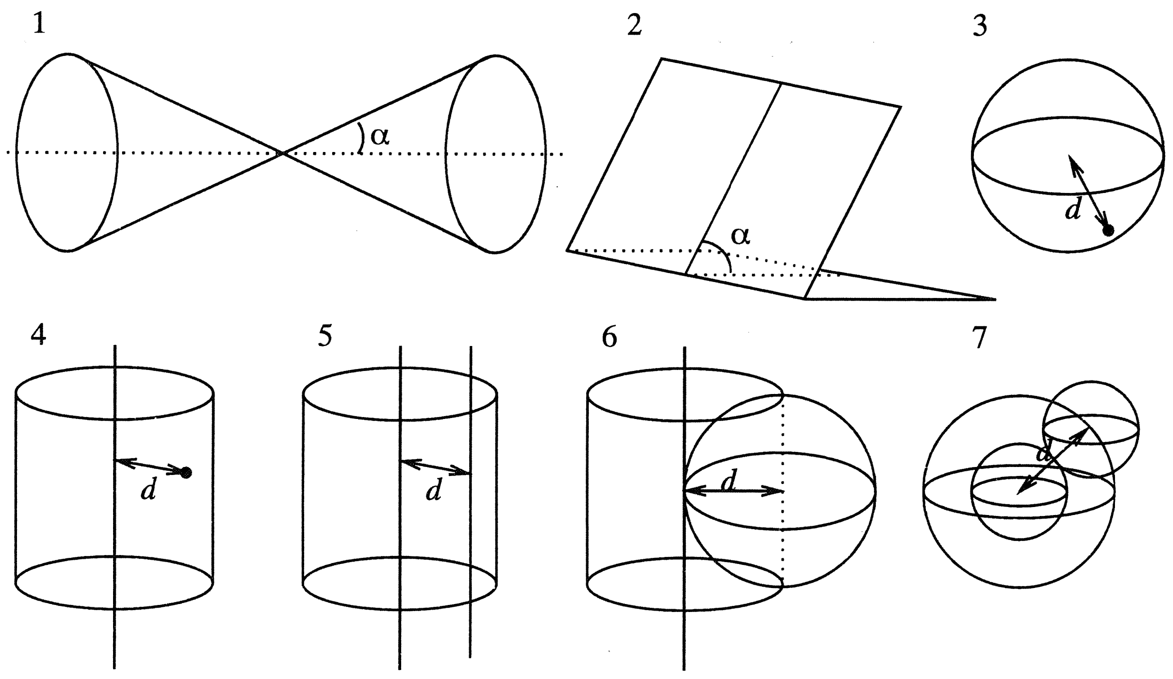


Figure 3: Quanta involved with 3D constraints.

In Section 10 we give an example using 3D constraints.

A set of constraints and geometric primitives can be represented as a network, such that the constraint node is connected to its variables. The network can be disconnected, in which case the variables in one connected component and the variables in other connected components are not related by constraints. Constraints are represented in rectangular boxes, geometric primitives in circular nodes. A quantum for a variable $v$, consistent with constraint $c(\ldots, v, \ldots)$, can be considered as a label on an edge between $\textcircled{v}$ and $\boxed{c}$. Being a geometric primitive, a quantum is represented as a circle but with the edge that it labels drawn through.

In order to represent unions and intersections of quanta, the special Boolean set operation nodes $\lhd\!\boxed{\cup}$ and $\lhd\!\boxed{\cap}$ are used. These nodes are operators whose operands are the quantum labels of edges coming in at the flat sides of the nodes and whose results are the quantum labels of the edges going out at the sharp ends of the nodes. If there are no quantum labels on the edges going out, these quanta are not (yet) computed. Similarly, if there are no quantum labels on the incoming edges, they are not yet generated or computed. In both cases, the logical relation between these forthcoming quanta is already incorporated in the network.

Between a constraint and an associated variable is a sequence of (zero or more) quanta interleaved with Boolean set operation nodes, on a path of edges. In the direction from the constraint to the variable each quantum includes the next one. Figure 4 shows a network that corresponds to the example of Figure 2. The network shown is an intermediate result in the satisfaction process.

Our approach to geometric constraint satisfaction is thus based on solution set inference, and performs propagation of known states (see the next section).

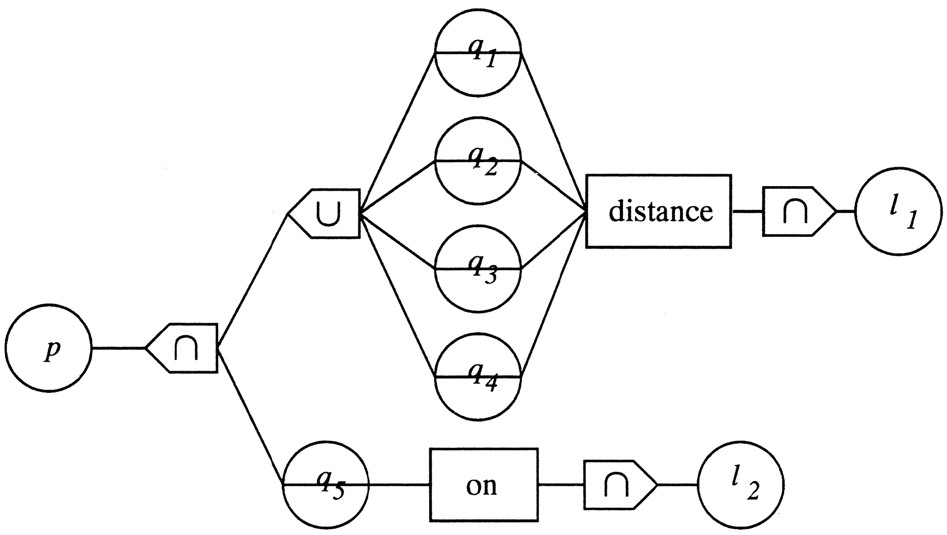


Figure 4: Intermediate network corresponding to Figure 2.

## 5. *Formal model*

Our model consists of a set of constraints $C$, a set of constraint variables $V$, a set of quanta $Q$, a quantum generating function $G$, and a function Tolerate($c$) that satisfies a constraint $c$ and performs propagation. The constraint variables and quanta are geometric primitives.

A constraint is a relation between a number of variables from $V$. A constraint can involve any number of variables, depending on its type. Constraints are multi-directional in the sense that each of the involved variables constrains the others. In contrast to many other systems, many variables may be undetermined. Constraints are specified incrementally, so that every variable in $V$ is involved in an ordered set of constraints. Each constraint can be either *restrictive* or *alternative.* A restrictive constraint further restricts the degrees of freedom of the involved variables. An alternative constraint provides alternatives to the solution of *previously* specified constraints on that variable. Considering a single variable and only constraints on that variable, the constraints thus form an expression of the form

$$((c_1 \text{ AND/OR } c_2) \text{ AND/OR} \ldots) \text{ AND/OR } c_k.$$

A constraint expression like $(c_1 \text{ OR } c_2) \text{ AND } (c_3 \text{ OR } c_4)$ for example is not possible in this scheme.

The quantum generating function $G : (c, v_i) \mapsto (q_1, \ldots, q_n)$ generates the set of quanta for $v_i$ that are consistent with the constraint $c(v_1, \ldots, v_k)$ and all the variables $v_j$, $j = 1, \ldots, k$, $i \neq j$, with their current quanta, regardless of the current quanta of $v_i$. Note that not all combinations of types of constraints and variables are geometrically meaningful; $G$ is defined only for a finite number of combinations. The generation of the quanta involves geometric reasoning, taking into account the nature of the constraint and the current quanta of the variables. Figure 5 illustrates the addition of a constraint to a network and the subsequent generation of quanta. Only the part of the network in the neighborhood of variable $v$ is depicted.

When a new constraint is added to the network, the quantum generating function is applied to each constrained variable. For a single variable, the union of the resulting quanta, $\cup(q_1, \ldots, q_n)$, is the solution to that one constraint. If the constraint is restrictive,

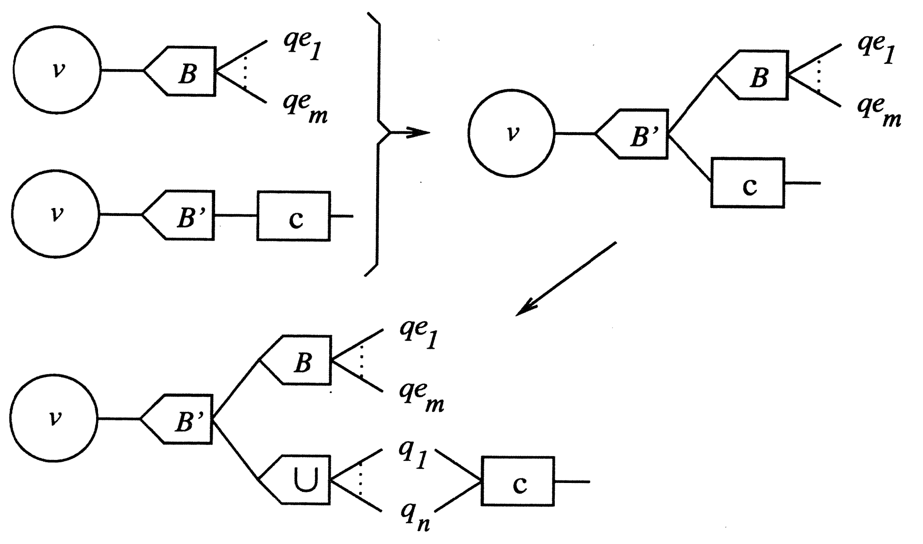


Figure 5: Addition of $c(v, \ldots)$ to a network, and generation of quanta.

$q_1, \ldots, q_n$ must be intersected with the variable's current quanta. If it is an alternative constraint, we take the union of the new and the current quanta. In this way we get an expression of quanta, which is the solution set of the constraints so far. In general, this quantum expression is of the form $B(qe_1, \ldots, qe_m)$, where $B$ is one of the Boolean set operators union or intersection, and each $qe_i$ is a quantum expression.

After addition of a new constraint, generating new quanta $q_1, \ldots, q_n$, the solution set becomes $B'(B(qe_1, \ldots, qe_m), \cup(q_1, \ldots, q_n))$. For a restrictive constraint $B'$ is $\cap$, for an alternative constraint $B'$ is $\cup$. This expression is simplified according to the following network transformation rules:

1. $\bigcup(\cup(q_1, \ldots, q_n)) = \cup(q_1, \ldots, q_n)$
2. $\bigcap(\cup(q_1, \ldots, q_n)) = \cup(q_1, \ldots, q_n)$
3. $\bigcup(\cup(qe_1, \ldots, qe_m), \cup(q_1, \ldots, q_n)) = \cup(qe_1, \ldots, qe_m, q_1, \ldots, q_n)$
4. $\bigcup(\cap(qe_1, \ldots, qe_m), \cup(q_1, \ldots, q_n)) = \bigcup(\cap(qe_1, \ldots, qe_m), q_1, \ldots, q_n)$
5. $\bigcap(\cup(qe_1, \ldots, qe_m), \cup(q_1, \ldots, q_n)) = \bigcup_{i,j}(q_i \cap qe_j)$
6. $\bigcap(\cap(qe_1, \ldots, qe_m), \cup(q_1, \ldots, q_n)) = \bigcup_{i=1}^{n} \cap(qe_1, \ldots, qe_m, q_i)$

Rules 1 and 2 apply when the constraint is the first one connected to $v_i$.

Each $qe$ in the other rules is either a single quantum or an intersection of other $qe$-s, because of the effect of rule 3. Any intersection $q_i \cap qe_j$ on the right-hand sides of equations 5 and 6 that can be computed, is evaluated into a union of quanta: $q_i \cap qe_j = \cup(q'_1, \ldots, q'_k)$. It may be difficult to compute the intersection, for example if one of the quanta is a circle that can float around. In that case we leave the intersection unevaluated (see also Section 9). This is why a $qe_j$ can be an intersection of quanta or other $qe$-s: $qe_j = \cap(qe'_1, \ldots, qe'_k)$.

The right-hand side of equation 6 may be split into an intersection that can be computed and a part that remains unevaluated.

Figure 6 shows the network transformations that correspond to the evaluation rules.

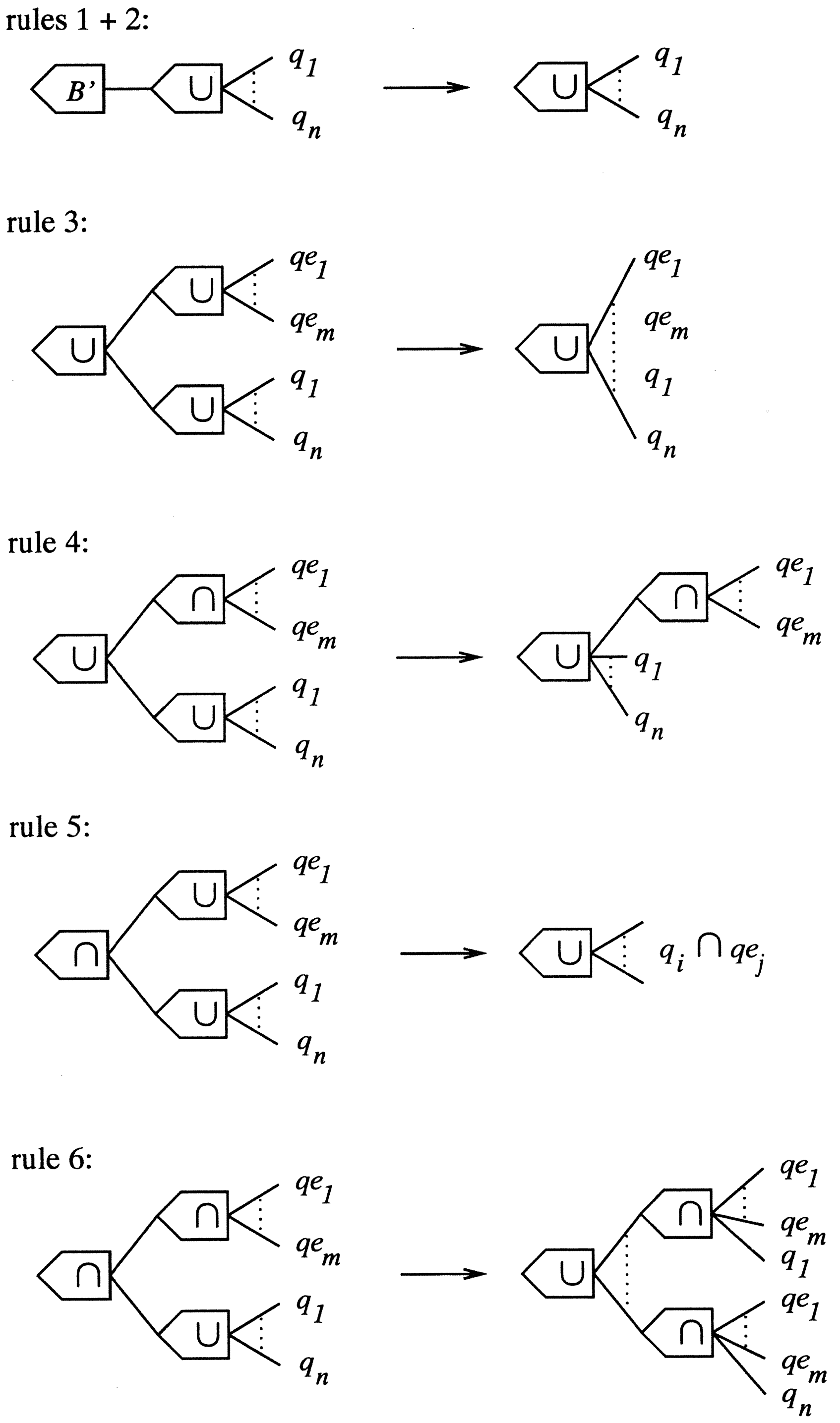


Figure 6: Network transformations corresponding to the evaluation rules.

Let $S_i = G(c, v_i)$. We let the procedure Evaluate($v_i, S_i$) compute intersections, transform the network accordingly as described in the previous section, and return EMPTY if the resulting solution set of $v_i$ is empty, CHANGED if it has been changed, and UNCHANGED if it is unchanged. We can call $G(c, v_i)$ and apply Evaluate($v_i, S_i$) for each variable of $c$.

If one of the variables gets assigned an empty solution set, the constraint expression is inconsistent — the specification is over-constrained. If the solution set of a variable $v_i$ has been changed and $v_i$ is involved in another constraint $c'$, the other variables of $c'$ need also be updated. These variables must therefore be recorded. The following procedure Revise($c$) updates all variables of $c$, and returns the set of variables whose solution set is changed (adapted from [Davis, 87]):

```
Revise ($c$)
{
  Changed = $\emptyset$
  for each $v_i$ of $c$
    $S_i = G(c, v_i)$
  for each $v_i$ of $c$
    case Evaluate ($v_i, S_i$)
    {
      EMPTY: Report (OVER-CONSTRAINED)
      CHANGED: add $v_i$ to Changed
      UNCHANGED:                                  // do nothing
    }
  return Changed
}
```

When a new constraint is specified, it must be added to the network, and the resulting changes must be propagated to related constraints. This may cause new changes which in turn must be propagated, and so on. This is performed by the following algorithm:

```
Tolerate($c$)
{
  Fifo = [$c$]                              // first-in-first-out queue
  while Fifo $\neq \emptyset$
  {
    $c'$ = FirstElementOf (Fifo)
    remove $c'$ from Fifo
    Changed = Revise ($c$)
    for each $v \in$ Changed
      for each $c'' \neq c'$ having $v$ as parameter
        add $c''$ to Fifo
  }
}
```

Revise($c$) makes the solution set of every variable of $c$ consistent with $c$ and its other variables using their current quanta. That is, the solution is made locally consistent. When Tolerate($c$) terminates (see next section), and an over-constrained case has not been detected, the solution sets of all variables are locally consistent (this is equivalent to arc consistency in [Mackworth, 77]).

When only restrictive constraints are added to a network, Tolerate($c$) amounts to the propagation algorithm of [Waltz, 72], filtering the solution sets to get subsets which are locally consistent.

Because constraints that have variables with changed quanta are placed in a first-in-first-out queue (as opposed to an unordered list), they are guaranteed to be revised again, whether the propagation is finite or infinite. This is called a fair propagation in [Güsgen and Hertzberg, 88].

## 7. *Termination*

This section discusses termination of the propagation in Tolerate($c$). We treat the cases where $c$ is a restrictive or an alternative constraint separately.

### 7.1. *Restrictive constraint*

When a restrictive constraint $c(v_1, \ldots, v_k)$ is added to the network, the solution sets of variables do not get larger. More formally, Evaluate$(v_i, G(c, v_i))$ is then monotonic in the following sense: let $\{[S_1, \ldots, S_k]\text{Evaluate}(v_i, G(c, v_i))\}$ denote the solution set of $v_i$ after applying Evaluate$(v_i, G(c, v_i))$ with $S_j$ the current solution set of $v_j$, $j = 1, \ldots, k$. Observe that:

1. $\{[S_1, \ldots, S_k]\text{Evaluate}(v_i, G(c, v_i))\} \subseteq S_i$
2. if $S'_j \subseteq S_j$, $j = 1, \ldots, k$, then
   $\{[S'_1, \ldots, S'_k]\text{Evaluate}(v_i, G(c, v_i))\} \subseteq \{[S_1, \ldots, S_k]\text{Evaluate}(v_i, G(c, v_i))\}$.

Because of the monotonicity of Evaluate$(v_i, G(c, v_i))$, and the fairness of propagation in Tolerate($c$), results from [Güsgen and Hertzberg, 88] (adapted to our context) apply: if there exists some terminating propagation resulting in a locally consistent solution, then Tolerate($c$) will find it. Moreover, the solution is unique.

Still, the propagation in Tolerate($c$) need not terminate. A simple example is where two line segments are related by a constraint that requires their lengths to be equal, and a constraint that sets one length to half the length of the other (a pure numeric analogue is the case of two constraints $x = y$ and $x = 2y$). This can lead to an infinite loop halving each line segment in turn.

A loop occurs when a constraint is revised more than once, in one invocation of Tolerate(). When this happens, it is not clear whether this is the prefix of an infinite loop or not, but we must break the loop after a finite number of times. One possibility is to terminate propagation by force when a constraint is addressed a fixed number of times. A more sophisticated approach is to break a loop when there are no more substantial changes made to the solution set. Specifically, when none of the quanta of a variable change type (for example change from a line segment into a point), and no quantum changes more than a certain 'geometric margin', we can break a loop. After a loop has been cut-off, the current quanta of the variables in the loop form a super-set of the real solution. There are several things we can do with such a super-set:

1. present the super-set to the user, which is still better than many graphics systems can provide,
2. compute the solution set numerically, using the super-set to obtain initial values and bounds, if necessary,
3. compute a single solution from the solution set,

4. test whether a solution exists instead of computing a solution (set).

### *7.2. Alternative constraint*

If an alternative constraint is added to the network, the solution sets of variables may get larger. This can more easily lead to an infinite loop than when the solution is restricted. When a variable involved in a circular chain of constraints gets assigned alternative quanta, these constraints may produce alternative quanta for their variables ad infinitum. Again, a loop need not be the prefix of an infinite loop, but propagation must be stopped. A loop can be stopped after a constraint is addressed a fixed number of times, or when the increase of the solution set exceeds a certain 'geometric margin'.

A (possibly infinite) increase of the solution may not have been the intention of the user, and the system can ask whether the alternative must be redrawn. In any case, after a loop has been cut-off, the current quanta of the variables in the loop form a sub-set of the real solution. We can use this result to:

1. present the sub-set to the user, which is still better than many graphics systems can provide,
2. try to generate a super-set of the solution, and proceed with propagation in order to restrict the solution.

## *8. Implied constraints*

Our system uses geometric simplification, for example by shrinking a circle to a point, and regarding the radius as a distance. In this way the constraint that a circle $c_1$ is tangent to another circle $c_2$, is regarded as the constraint that the shrinked $c_1$ has a distance to the shrinked $c_2$ that is equal to the sum of their radii $r_1 + r_2$. That is, $c_1$ lies on the circle $q_1$ with radius $r_1 + r_2$, and concentric with $c_2$.

In order to simplify geometric reasoning, we exploit implied constraints: constraints that can be derived from other constraints and variables. Implied constraints can operate on variables and quanta. Indeed a quantum is a geometric primitive, and can thus act as a constraint variable, while it still represents a solution to another constraint. The user has no direct access to these implicit constraints.

See Figure 7 for a situation where two tangent circles and their corresponding network explicitly represent some useful implied constraints. The quanta $q_1$ and $q_2$ are circles concentric with $c_2$ and $c_1$, respectively. As $c_1$ moves around, for example along line $l$, so does $q_2$.

An implied constraint is derived from other constraints and variables, and so it does not introduce new facts. Its purpose is to ease the problem solving by an extra simplification. In our example, the coincidence of the centers of $c_1$ and $q_2$ causes changes to $c_1$ to mechanically carry over to $q_2$.

The type of an implied constraint (restrictive or alternative) must be consistent with the constraint(s) it is derived from, otherwise it would provide incorrect restriction or alternatives. Again, this introduces no more restrictions or more alternatives to the variables of the implicit constraint. For example if parallel$(l_1, l_2)$ AND parallel$(l_2, l_3)$, then parallel$(l_1, l_3)$ is implied, and must also be restrictive.

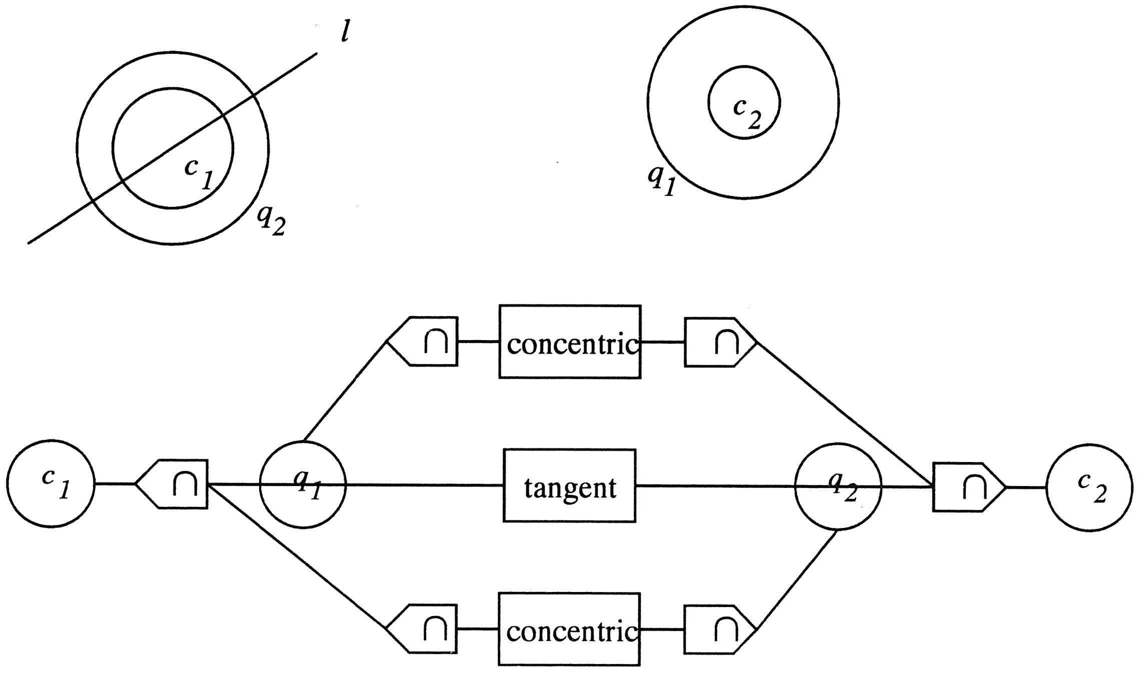


Figure 7: Implied constraints concentric$(c_1, q_2)$ and concentric$(c_2, q_1)$.

## 9. *Completeness*

Completeness of our constraint system depends on the set of geometric primitives, the allowable constraints, the power of geometric reasoning behind the quantum generating function, and the system's competence to intersect quanta. This last aspect is illustrated by the example in Figure 7. As the circle $c_1$ moves along line $l$, $q_2$ moves accordingly, thus sweeping out a whole area of solutions. Although the area has a simple shape, its definition in terms of a swept circle makes intersection with subsequent quanta non-trivial. In this example the intersection is clearly computable, but a particular system may not be able to perform this intersection.

When we cannot compute an intersection of two quanta, we leave the intersection unevaluated, and represent it with a $<\!\boxed{\cap}$ node in the network. A following intersection may resolve the problem, as the following extension of our example illustrates. If $c_2$ is further constrained to lie on line segments $m_1$ and $m_2$, and the intersection of $m_1$ with the swept circle $q_2$ is left unevaluated, the next intersection $(q_2 \cap m_1) \cap m_2$ is transformed into $(q_2 \cap m_2) \cap (m_1 \cap m_2)$. If $m_1 \cap m_2 = \emptyset$, so is the final solution. If $m_1 \cap m_2$ is a point, it is easier to test if it is included in $(q_2 \cap m_2)$ than to compute $(q_2 \cap m_2)$ itself.

In this way, delayed satisfaction of constraints is handled uniformly, thereby enhancing the satisfaction capabilities of the system.

In our examples, we used only points, lines, and circles as geometric primitives, and tangent(), parallel(), on(), and distance() as constraints. With these collections of primitives and constraints, we can compute the solution sets when the system gets into trouble, by turning the geometric information into equations and solve them algebraically [Owen, 91] or numerically.

In the case of more general primitives and constraints, the system may resort to some analytical method or simply specify to the user what caused the problem.

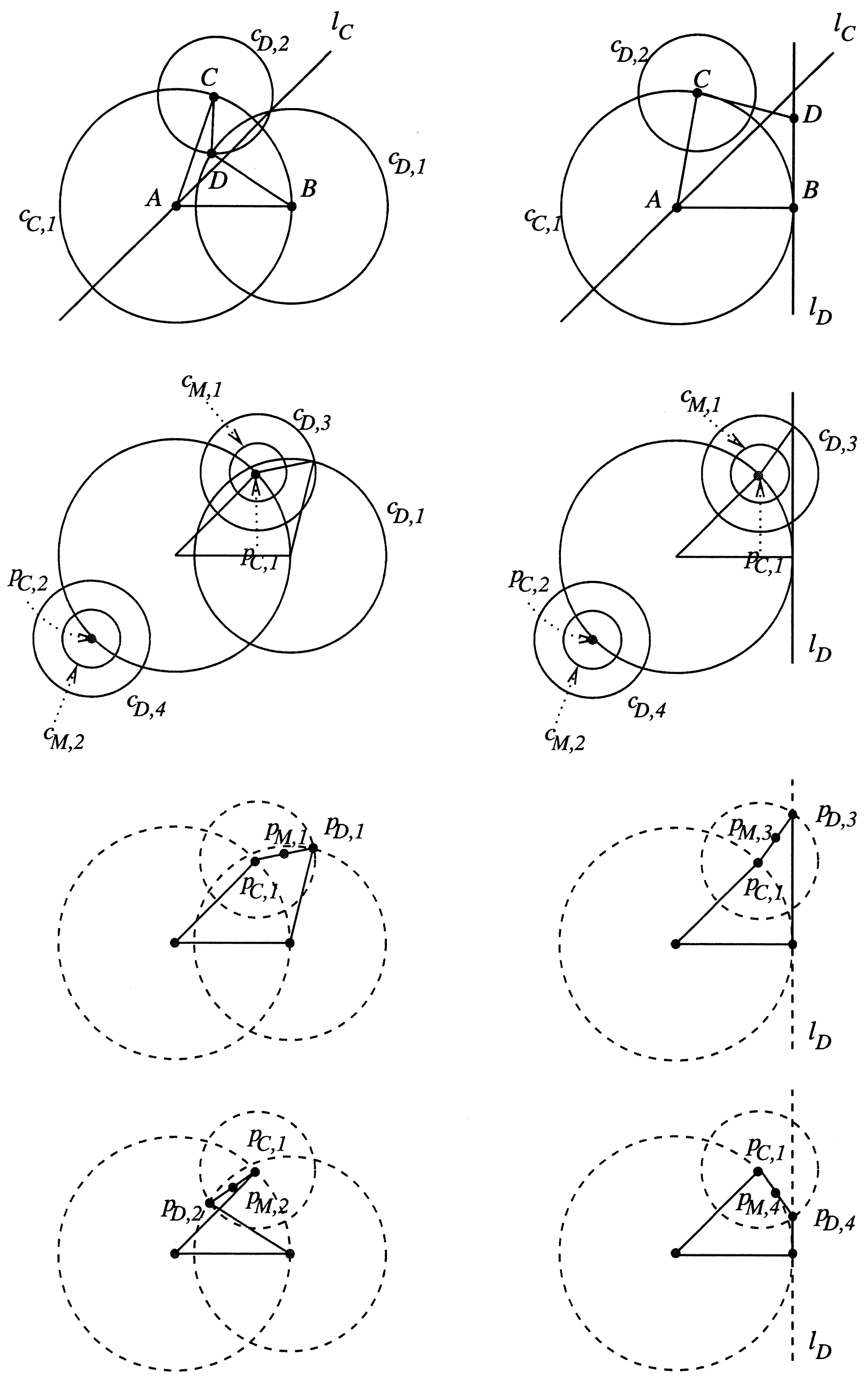


Figure 8: A four-bar linkage; $A$ and $B$ are fixed.

## *10. Examples*

### *10.1. Planar linkage*

Consider the four-bar linkage of Figure 8 (top left), involving points $A$, $B$, $C$, $D$, and $M$. The points $A$ and $B$ are fixed and a distance $d_1$ apart. The constraints on $C$, $D$, and $M$ are the following:

constraint 1: the distance between $A$ and $C$ is $d_1$;
AND constraint 2: the distance between $B$ and $D$ is $d_2$;
OR constraint 3: the link $BD$ is vertical;
AND constraint 4: the distance between $C$ and $D$ is $d_3$;
AND constraint 5: $M$ is the midpoint of $C$ and $D$;
AND constraint 6: the line through $A$ and $C$ has a fixed slope.

These constraints are specified in the given order, and we are interested in the position of $C$, $D$, and $M$.

This problem can be solved algebraically by computing the locus of $C$, $D$, and $M$ each time a constraint is specified. As an illustration, set $A$ to $(0,0)$ and $B$ to $(0,d_1)$, without loss of generality, and ignore constraint 3 for the moment. Then the locus of $M=(x,y)$ just after specifying constraint 5 is given by a polynomial of degree six. If $d_2=d_3$ this polynomial reduces to:

$$16(x^4+y^4)+32x^2y^2-16d_1(x^3+xy^2)-(12d_1^2+4d_2^2)(x^2+y^2)+(8d_1^3-4d_1d_2^2)x=d_1^2d_2^2-4d_1^4$$

(see [Chou and Gao, 89]).

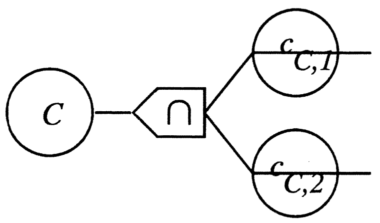
This equation can be derived by an algebraic manipulation system, but that is of little use here because it does not interact in a simple way with the subsequent constraint.

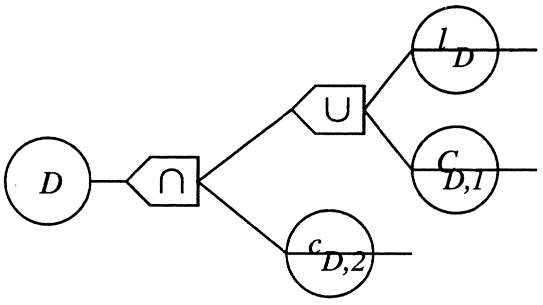
Alternatively, our constraint management system comes up with the following quanta:

by constraint 1: $C$ is restricted to a quantum that is the circle $c_{C,1}$ centered at $A$ and having radius $d_1$;
by constraint 2: $D$ is restricted to the circle $c_{D,1}$ centered at $B$ with radius $d_2$;
by constraint 3: $D$ is restricted to the vertical line $l_D$ through $D$;
by constraint 4: for $C$: a circle $c_{C,2}$ with radius $d_3$ which has no fixed position, but is centered at $D$; similarly for $D$: a circle $c_{D,2}$ concentric to $C$, having radius $d_3$;
by constraint 5: because $M$ is the midpoint of two points whose positions are unknown, no quantum is generated yet for $M$;
by constraint 6: since the line through $A$ and $C$ must have a fixed slope, another quantum is associated to $C$: line $l_C$.

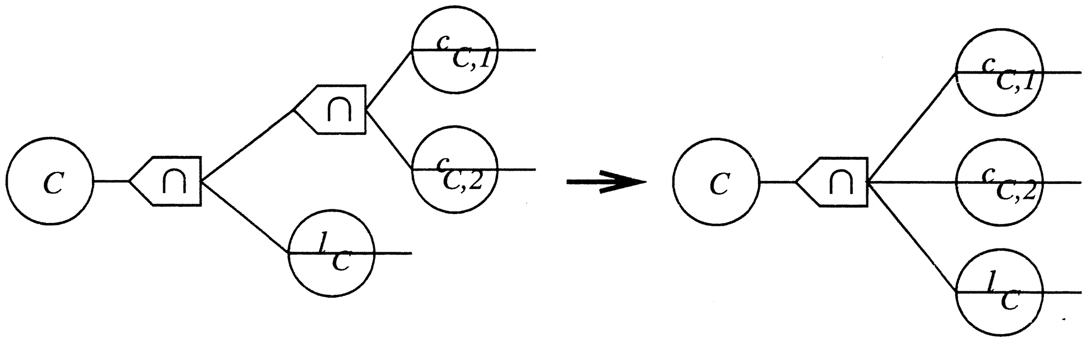
The quanta $c_{C,1}$, $c_{D,1}$, $c_{D,2}$, $l_C$, and $l_D$ are shown at the top row of Figure 8, where the two possible cases induced by the alternative constraint are depicted in separate columns.

The quanta resulting from constraint 4, $c_{C,2}$ and $c_{D,2}$, are not fixed but can move around. Consequently, it is difficult to intersect these quanta with the other quanta $c_{C,1}$, $c_{D,1}$, and $l_D$. The intersection remains unevaluated, but is represented in the network:

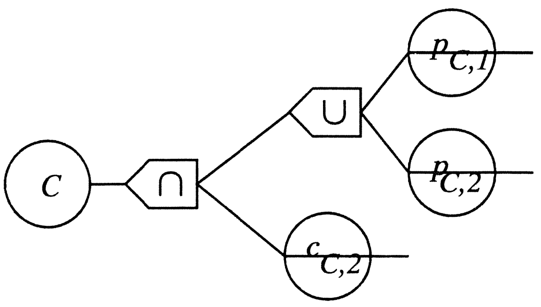




The next quantum for $C$, $l_C$, must be intersected with the current quanta. Application of rule 6 (with $n = 1$) gives $\bigcap(\cap(c_{C,1}, c_{C,2}), l_C) = \cap(c_{C,1}, c_{C,2}, l_C)$:

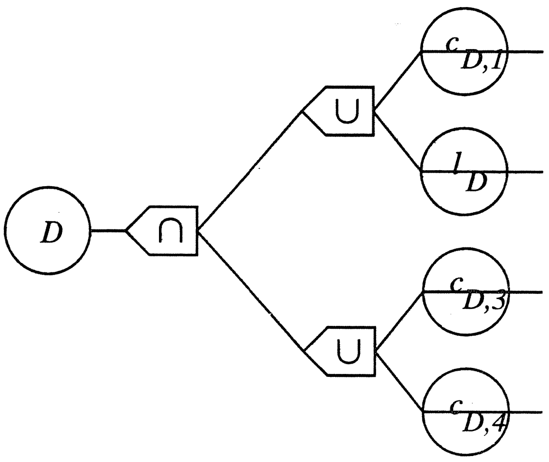


Intersection $l_C \cap c_{C,1}$ can be computed and gives two points $p_{C,1}$ and $p_{C,2}$, the intersection with $c_{C,2}$ is delayed:



This intermediate result must be propagated to objects related to $C$. Because $C$ is now restricted to at most $p_{C,1}$ and $p_{C,2}$, the movable quantum $c_{D,2}$ of $D$ is replaced by two circles $c_{D,3}$ and $c_{D,4}$ centered at $p_{C,1}$ and $p_{C,2}$ respectively. These circles can be seen as two specific instances of $c_{D,2}$. Also, since $M$ is the midpoint of $C$ and $D$, it is now restricted to two circles $c_{M,1}$ and $c_{M,2}$, concentric with $c_{D,3}$ and $c_{D,4}$ but with half their radius. The new quanta are depicted in the second row Figure 8.

Quanta $c_{D,3}$ and $c_{D,4}$ are intersected with $c_{D,1}$ and $l_D$:



Application of rule 5 gives $\bigcap(\cup(c_{D,1}, l_D), \cup(c_{D,3}, c_{D,4})) = \bigcup(\cap(c_{D,1}, c_{D,3}), \cap(c_{D,1}, c_{D,4}), \cap(l_D, c_{D,3}), \cap(l_D), c_{D,4})$, which can be computed, yielding $p_{D,1}, \ldots, p_{D,4}$.

Again this intermediate result must be propagated. Since $D$ is now restricted to $p_{D,1}, \ldots, p_{D,4}$, the quanta $p_{C,2}$ and $c_{M,2}$ are deleted, and $c_{M,1}$ is replaced by $p_{M,1}$ and $p_{M,2}$. The four possible solution configurations are shown in the two lower rows of Figure 8.

### *10.2. Articulated structure in space*

The application in this subsection is illustrated in figure 9. The 3D scene in this example contains a plate $P$, a spherical source of radiation $R$, and an articulated structure supporting $P$, consisting of two arms $A_1$ and $A_2$, having balls $B_1$ and $B_2$ respectively at their endings. $P$, $R$, $A_1$ and $B_1$ are totally fixed. The constraints on $A_2$ and $B_2$ are the following:

constraint 1: $A_2$ must have a fixed angle $\alpha$ with respect to $A_1$;
AND constraint 2: $B_2$ must be at a fixed distance $d$ from $R$;
AND constraint 3: $B_2$ must be tangent to $P$ at the bottom side.

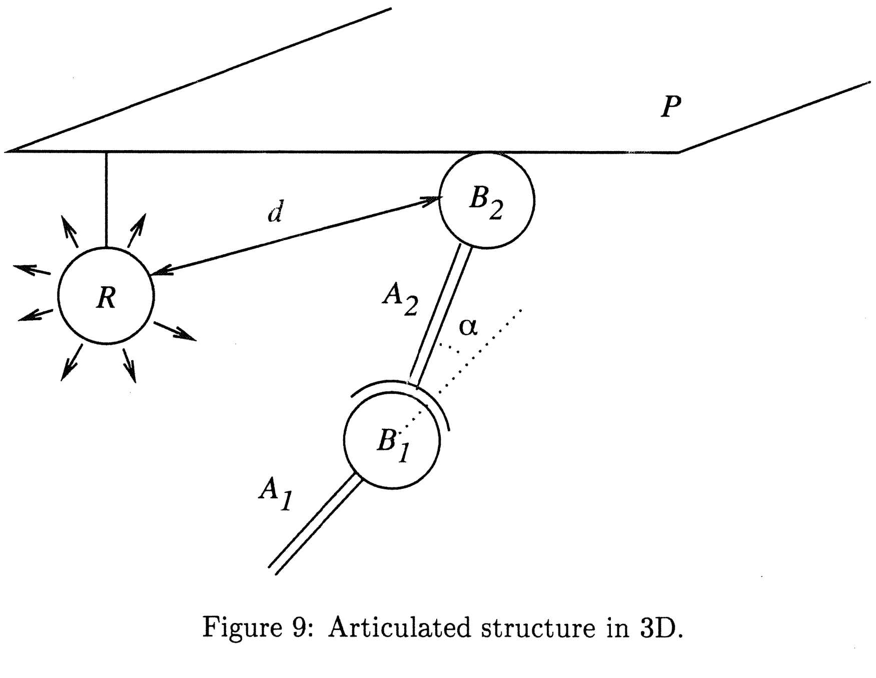


Figure 9: Articulated structure in 3D.

The constraints are incrementally specified in this order. We want to know the allowable positions of ball $B_2$.

The center of $B_2$ is restricted to the following quanta:

by constraint 1: a cone $C$ with its top at the center of $B_1$;
by constraint 2: a sphere $S$, concentric with $R$, having radius $d$.
by constraint 3: a plane $P'$ below $P$, at a distance equal to the radius of $B_2$;

Intersecting these quanta in this order gives $(C \cap S) \cap P'$. In general, however, the intersection of a cone with a sphere is not a simple primitive object. Applying delayed evaluation again results in the actual intersection of $(C \cap P')$, an ellipse, with $(S \cap P')$, a circle. Assuming that both intersections are not void and the ellipse is non-degenerate, the intersection of the circle and the ellipse gives two, one, or no solutions, depending on the actual positions and orientations of $R$, $P$, and $A_1$.

## *11. Constraints and objects*

The geometric representation of the constraint variables and quanta naturally corresponds to object-oriented graphics concepts. On the other hand, relationships between objects other than 'is-a' and 'part-of' hierarchies, such as constraints, cannot be suitably represented by message passing.

We distinguish two possible incompatibilities between constraints and object-oriented concepts:

- a constraint solver looks at, and sets, the constraint variables' internal data, which conflicts with the information-hiding concept in the object-oriented paradigm;
- object-oriented programming is imperative, while constraint programming is declarative.

If one is to use object-oriented methods to manage complexity in a geometric modeling system, and also wants to provide constraints to its users as a tool to manage the complexity of problem solving, then constraints and objects must be friends. However, message passing for constraint handling, as in [Laffra and van den Bos, 90], is either limited to constraint maintenance, or against the object-oriented philosophy: providing all objects subject to constraints with interface methods to get and set their internal data, grants every other object to get and set their values.

One way to restrict this, is to have an object allow value setting only when its internal constraints remain satisfied, see [Rankin, 91]. By contrast, access to private data can be limited to constraint-objects (or the constraint solver-object) only. For example C++ provides the 'friend' declaration to grant functions access to the private part of objects, see Section 11.1. This is also comparable to [Cournarie and Beaudouin-Lafon, 91], where special variables (slots) are accessible by constraints only.

One can argue that the encapsulation is still violated (and specifically that a C++ friend is not intended to change the state of an object). Alternatively one can see constraints more as a means to access information in an orderly and restricted way, than that they violate the information hiding principle. At least the responsibility for integrity is shifted from the constraint user to the constraint implementer. A problem that remains is the difficulty of debugging a constraint system, due to the global effects of constraints.

Object-oriented languages are imperative, and thus use a notion of state, particularly represented by objects. On the other hand, pure constraint languages are declarative, and thus specify one single timeless state: the solution to the specified problem. However, both paradigms can be combined as in [Freeman-Benson, 90], where an imperative assignment to a variable sets a value at one moment in time, and a declarative constraint dictates a value from that moment on. However, if the solution depends on the order in which constraints are solved, the declarative semantics is destroyed. The constraint satisfaction with quantum labels finds whole solution sets. An intermediate result may be a sub-set or a super-set of the true solution set, but the final solution does not depend on the order of satisfaction, so that delayed satisfaction has no side effects. As a result, the declarative semantics of constraints are preserved.

## *11.1. Implementation*

We distinguish the following relevant objects classes: GeometricPrimitive, Quantum, and Constraint. Instances of objects are created upon user demand, typically via a graphics interface.

A quantum can be designed as a polymorphic object which refers to a line, circle, polygon, etc. An alternative in environments supporting multiple inheritance is to let a quantum object inherit properties from geometric primitives, as well as additional properties that are specific for quanta.

The exact geometric primitive type of each individual quantum object must be known to enable correct use of the primitive. The primitives provide this type-information by means of the method whatami(). When using an object-oriented curve intersection method such as [Rankin and Burns, 90], primitives also provide a method nearestpoint() which returns the point on the curve that is nearest to a given point. A draw() method typically calls the drawing method of the geometric primitive after a drawing style is specified, to distinguish a quantum from a design primitive.

All constraints are subclasses of the base class Constraint, and have methods draw() and satisfy(). The draw() method visualizes the constraint by connecting the operands with an arc that is labeled with a corresponding constraint icon.

The system is currently being implemented in C++ [Stroustrup, 86] and Quintus Prolog [Quintus, 90], which provides an interface with C. Class methods, in particular the constraint method satisfy(), may contain assertions and goals in Horn clause logic. This provides a way to exploit the geometric context of the constraint and variables, and to make the system more intelligent by geometric reasoning.

In a typical C++ implementation the above mentioned functions are virtual, to indicate that the base (super) class has a general version, and the derived classes can have different own versions:

```
class GeometricPrimitive
{
  friend class Constraint;                  // grant access to private data
  private:
     Frame frame;                           // modeling coordinate system
     Color color;
     // ...
  public:
     virtual GPType whatami();                  // geometric primitive type
     virtual Point nearestpoint();
     virtual void draw();
     // ...
}
```

This is only one of the many possible ways to implement the quantum approach, exploiting both object-oriented concepts and relational logic. The alternatives include an extension of (Concurrent) Prolog with object -oriented features, see for example [Zaniolo, 84], [Shapiro and A.Takeuchi, 83], and the language Oar [Arbab and Wang, 89]. Objects can also be combined with a functional language, see for example Common Lisp Object System (CLOS) [Moon, 89]. All these languages specify objects, hierarchies, and methods in a declarative way.

## *12. Concluding remarks*

Operations on a variable, such as translate, scale, mirror, and project, can be handled by performing proper corresponding operations on its quanta. These new quanta must then be propagated through the network.

For interactive CAD purposes, a constraint must also be removable. In our system, deletion from the network, involves removal of the constraint and the sequence of quanta and Boolean set operation nodes towards its involved variables. Sequences from other constraints to the same variables must be 'repaired' by intersecting the proper quanta. The resulting new quanta must be propagated through the new network. Note that the explicitly represented implied constraints must also be removed.

The quantum approach can also be applied to pure numeric constraint problems. Instead of geometric variables and quanta, intervals are concerned. Algebraic reasoning then takes the place of geometric reasoning.

More research is needed to further exploit the capabilities of geometric reasoning. In particular implicit constraints (relations between quanta to which the user has no direct access) can be used by the satisfaction system to solve complex constraints more directly.

## *Acknowledgements*

This research is part of the IIICAD project, supported by NWO (Dutch Organization for Scientific Research) under Grant NF-51/62-514.

## *References*

[Arbab and Wang, 89] Arbab, F. and Wang, B. A geometric constraint management system in Oar. In ten Hagen, P. J. W. and Veerkamp, P. (editors), *Intelligent CAD Systems III - Practical Experience and Evaluation (Proceedings 3rd Eurographics Workshop on Intelligent CAD Systems, 1989).* Springer-Verlag, 1989, 231 - 252.

[Arbab and Wing, 85] Arbab, F. and Wing, J. M. Geometric reasoning: A new paradigm for processing geometric information. In Yoshikawa, H. and Warman, E. A. (editors), *Design Theory for CAD (Proceedings of the IFIP W. G. 5.2 Working Conference, 1985).* Elsevier Science Publishers, 1985, 145 - 165.

[Badler and Kamran, 87] Badler, N. I. and Kamran, H. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics & Applications (special issue on articulated figure animation*, 7(6), 1987, 28 - 38.

[Borning, 81] 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.

[Chou and Gao, 89] Chou, S.-C. and Gao, X.-S. A collection of 120 computer solved geometry problems in mechanical formula derivation. Technical Report TR-89-22, University of Texas, Department of Computer Science, 1989.

[Cohen, 90] Cohen, J. Constraint logic programming languages. *Communications of the ACM*, 33(7), 1990, 52 - 68.

[Cournarie and Beaudouin-Lafon, 91] Cournarie, E. and Beaudouin-Lafon, M. Alien: a prototype-based constraint system. In [Laffra and Blake, 92].

[Davenport et al., 88] Davenport, J. H., Siret, Y., and Tournier, E. *Computer Algebra — systems and algorithms for algebraic computation.* Academic Press, 1988.

[Davis, 87] Davis, E. Constraint propagation with interval labels. *Artificial Intelligence*, 32, 1987, 281 - 331.

[Emmerik, 90] Emmerik, M. J. G. M. v. A system for interactive graphical modeling with 3D constraints. In *Proceedings CGI '90.* 1990.

[Freeman-Benson, 90] Freeman-Benson, B. N. Kaleidoscope: Mixing objects, constraints, and imperative programming. *(ECOOP/OOPSLA '90 Proceedings) SIGPLAN Notices*, 25(10), 1990, 77 - 88.

[Güsgen and Hertzberg, 88] Güsgen, H.-W. and Hertzberg, J. Some fundamental properties of local constraint propagation. *Artificial Intelligence*, 36, 1988, 237 - 247.

[Heintze et al., 87] Heintze, N. C., Michaylov, S., and Stuckey, P. J. CLP($\mathbb{R}$) and some problems in electrical engineering. In Lassez, J.-L. (editor), *Proceedings of the 4th International Conference on Logic Programming.* MIT Press, 1987.

[Helander, 88] Helander, M. (editor). *Handbook of Human-Computer Interaction.* North-Holland, 1988.

[Laffra and Blake, 92] Laffra, C. and Blake, E. (editors). *Advances in Object-Oriented Graphics II (Proceedings of the second Eurographics Workshop on Object-Oriented Graphics, Texel, The Netherlands, June 1991)*, EurographicSeminars Series. To be published by Springer-Verlag, 1992.

[Laffra and van den Bos, 90] Laffra, C. and van den Bos, J. Propagators and concurrent constraints. In *Proceedings of the Object Based Concurrent Systems workshop at ECOOP/OOPSLA '90, Ottawa.* 1990.

[Leler, 88] Leler, W. *Constraint Programming Languages, Their Specification and Generation.* Addison-Wesley, 1988.

[Lin et al., 81] Lin, V. C., Gossard, D. C., and Light, R. A. Variational geometry in computer-aided design. *Proceedings SIGGRAPH '81, Computer Graphics*, 15(3), 1981, 171 - 177.

[Mackworth, 77] Mackworth, A. K. Consistency in networks of relations. *Artificial Intelligence*, , 1977, 99 - 118.

[Mäntylä, 88] Mäntylä, M. *An Introduction to Solid Modeling.* Computer Science Press, 1988.

[Moon, 89] Moon, D. A. The common lisp object-oriented programming language. In Kim, W. and Lochovsky, F. (editors), *Object-Oriented Concepts, Databases, and Applications.* ACM Press/Addison Wesley, 1989.

[Nelson, 85] Nelson, G. Juno, a constraint-based graphics system. *Proceedings SIGGRAPH '85, Computer Graphics*, 19(3), 1985, 235 - 243.

[Owen, 91] Owen, J. C. Algebraic solution for geometry from dimensional constraints. In *Proceeings of the ACM Conference on Solid Modeling.* ACM Press, 1991, 397 - 407.

[Pratt, 87] Pratt, M. J. *Form Features and their Applications in Solid Modeling, Tutorial SIGGRAPH '87.* ACM, 1987.

[Quintus, 90] Quintus (1990). *Quintus Prolog User Manual.* Quintus Computer Systems, Inc.

[Rankin, 91] Rankin, J. R. A graphical object oriented constraint solver. In [Laffra and Blake, 92].

[Rankin and Burns, 90] Rankin, J. R. and Burns, J. Coordinate frames and geometric approximation in graphics object oriented programming. In Blake, E. and Wißkirchen, P. (editors), *Advances in Object-Oriented Graphics I (Proceedings of the Eurographics Workshop on Object-Oriented Graphics, 1990)*, EurographicSeminars Series. Springer-Verlag, 1990, 131 - 148.

[Rossignac, 86] Rossignac, J. R. Constraints in constructive solid geometry. In Crow, F. and Pizer, S. M. (editors), *Proceedings of the 1986 ACM Workshop on Interactive 3D Graphics.* ACM Press, 1986, 93 - 110.

[Ruttkay, 91] Ruttkay, Z. A co-operative graphical editor based on dynamically constrained objects. In [Laffra and Blake, 92].

[Shapiro and A.Takeuchi, 83] Shapiro, E. and A.Takeuchi. Object-oriented programming in concurrent Prolog. *New Generation Computing*, 1(1), 1983, 25 - 48.

[Steele Jr. and Sussman, 79] Steele Jr., G. L. and Sussman, G. J. CONSTRAINTS. *APL Quote Quad*, 9(4–Part 1), 1979, 208 - 225.

[Stroustrup, 86] Stroustrup, B. *The C++ Programming Language.* Addison Wesley, 1986.

[Sutherland, 63] Sutherland, I. Sketchpad: A man-machine graphical communication system. In *Proceedings of the Spring Joint Computer Conference (IFIPS).* 1963, 329 - 345.

[Szekely and Meyers, 88] Szekely, P. and Meyers, B. A user-interface toolkit based on graphical objects and constraints. In *Proceedings of the 1988 ACM Conference on Object-Oriented Programming Systems, Languages, and Applications.* ACM, 1988, 36 - 45.

[Veltkamp, 91a] Veltkamp, R. C. A quantum approach to geometric constraint satisfaction. In [Laffra and Blake, 92].

[Veltkamp, 91b] Veltkamp, R. C. Geometric constraint management with quanta. In Brown, D. C., Waldron, M., and Yoshikawa, H. (editors), *Proceedings of the IFIP TC5/WG5.2 Working Conference on Intelligent Computer Aided Design, Columbus, Ohio, September 1991).* To be published by North-Holland.

[Veltkamp and Arbab, 91] Veltkamp, R. C. and Arbab, F. Geometric constraint propagation with quantum labels. In *Preliminary proceedings of the Eurographics Workshop on Computer Graphics and Mathematics, Genova, Italy, October 1991.*

[Waltz, 72] Waltz, D. L. *Generating Semantic Descriptions from Drawings of Scenes with Shadows.* PhD thesis, MIT, 1972.

[Zaniolo, 84] Zaniolo, C. Object-oriented programming in Prolog. In *Proceedings of the IEEE International Symposium on Logic Programming.* IEEE, 1984, 265 - 270.