Numerical Solving of Geometric Constraints by Bisection: A Distributed Approach
Samy Ait-Aoudia and Imad Mana
Pages 66 - 73

Numerical Solving of Geometric Constraints by Bisection: A Distributed Approach

Samy Ait-Aoudia and Imad Mana
INI - Institut National d'Informatique, BP 68M - Oued Smar 16270 Algiers Algeria

Abstract: *In computer-aided design, geometric modeling by constraints enables users to describe shapes by relationships called constraints between geometric elements. The problem is to derive automatically these geometric elements. Many resolution methods have been proposed for solving systems of geometric constraints. Geometric methods can be very efficient but are only applicable to particular kinds of problems (typically rule and compass constructive problems). Some schemes can't be solved by any geometric method. A numerical method is then necessary. We use the bisection method to solve these cases. The bisection method enables users to reliably find all solutions to a system of non-linear equations within a region defined by bounds on each individual co-ordinate of the geometric objects. But bisection is very time consuming. To overcome this problem, we present first a method that speed-up the resolution process. This improvement is still insufficient; we propose then to distribute the processing on several PC machines operating in parallel.*

Keywords: *Geometric Modeling, Constraints, Bisection method.*

Received: *February 18, 2004* | **Revised:** *January 13, 2005* | **Accepted:** *March 15, 2005*

1. Introduction

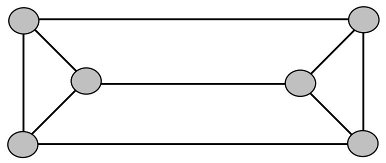
In computer aided design, geometric modeling by constraints enables users to describe shapes by specifying a rough sketch and adding to it geometric constraints i.e. a set of required relations between geometric elements. For example, the set of elements might be three points, with three constraints that specify the distance between each pair of points. The constraint solver must derive automatically the correct shape needed.

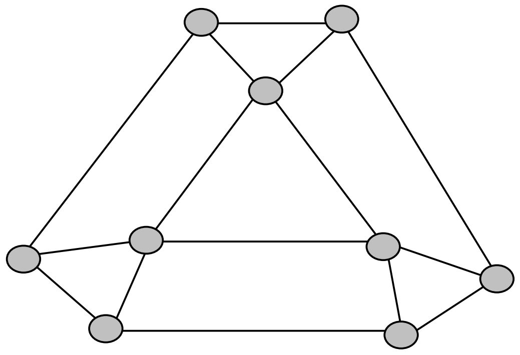
Many programming styles or languages have been investigated: imperative, object oriented, rules driven. Many resolution methods have been proposed for solving systems of geometric constraints. We classify the resolution methods in four broad categories: geometric, symbolic, rule-oriented and numerical.

In symbolic methods (see [4]), the constraints are translated into a system of equations. Methods such as Gröbner bases or elimination with resultants are applied to find symbolic expressions for the solutions. These methods are "extremely" time consuming.

Rule-based solvers rely on the predicates formulation (see [18,19,20]). Although they provide a qualitative study of geometric constraints, the "huge" amount of computations needed (exhaustive searching and matching) make them inappropriate for real world applications.

Geometric methods can be very efficient but are only applicable to particular kinds of problems, typically rule and compass constructive problems (see [2,3,5,6]). The geometric designs shown in figure 1 (these schemes are well constrained) can't be solved by any geometric method (see [14] for the proof). The nodes of theses graphs represent points and the edges represent distance constraints in 2D space.





(a) (b)

Figure 1: Ruler and compass non-constructible configurations.

Thus, numerical methods are very useful to solve "difficult" configurations. Geometric constraints are, as in symbolic methods, translated into a system of equations and then solved with a numerical method. The most popular numerical method is Newton-Raphson's iteration. It was used, among many others, by Serrano [16] and Perez et al [15]. This method needs an initial guess, typically given by the sketch of the desired geometric scheme.

However, there is a well-known problem. If Newton-Raphson's method often works well, sometimes it does not converge or it converges to an unwanted solution. In this last case, the user changes his initial guess until Newton-Raphson's method works if it does.

In this paper, we describe a constraint-based modeler that uses a bisection method instead of Newton-Raphson's iteration to solve the geometric constraints when there is no possibility to solve them by a geometric method. The bisection method avoids the drawbacks of the Newton iteration. It enables users to reliably find all solutions to a system of non-linear equations within a region defined by bounds on each individual co-ordinate of the geometric objects. But the major drawback of the bisection method is the fact that it is very time consuming. So speeding-up the resolution process is an unavoidable matter.

This paper is organized as follows. Correspondence between geometric constraints and equations is explained in section 2. In section 3, we briefly describe the mathematical framework of the bisection method. The decomposition method, of the system of equations into well-, under- and over constrained parts, is given in section 4. We present in section 5, the method used for "algorithmically" speeding-up the resolution process. In section 6, we describe how to distribute the computational task over a network of personal computers. Some results are shown in section 7. In section 8 we give conclusions.

2. Constraints and equations

In a numerical constraint solver, the geometric constraints are first translated into a system of algebraic equations (linear or not). This system is then solved by applying a numerical method. For each constraint correspond an algebraic equation. As an example of this correspondence, the equation corresponding to "internal" and "external" tangency between two circles (see figure 2), one centred at A with radius r_1, the other centred at B with radius r_2 is the following :

$$(x_B - x_A)^2 + (y_B - y_A)^2 - (r_1 \pm r_2)^2 = 0.$$

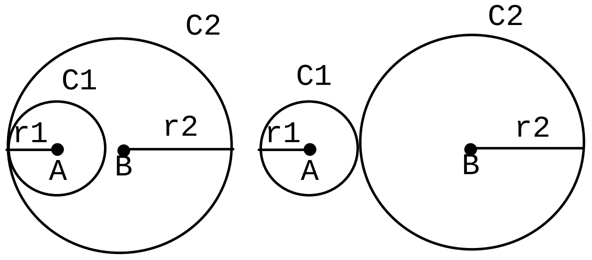


Figure 2: Tangency between two circles.

Example 1:

A constrained scheme is shown in figure 3. It consist of three points (P_1, P_2, P_3) and three distance constraints. The point P_1 lies at the origin and the point P_3 lies on the positive x-axis. The corresponding set of equations is given hereafter.

$eq_1 : x_1 = 0.$
$eq_2 : y_1 = 0.$
$eq_3 : y_3 = 0.$
$eq_4 : y_2 - y_1 - d_1 = 0.$
$eq_5 : (x_3 - x_2)^2 + (y_3 - y_1)^2 - d_3^2 = 0.$
$eq_6 : x_3 - x_1 - d_2 = 0.$

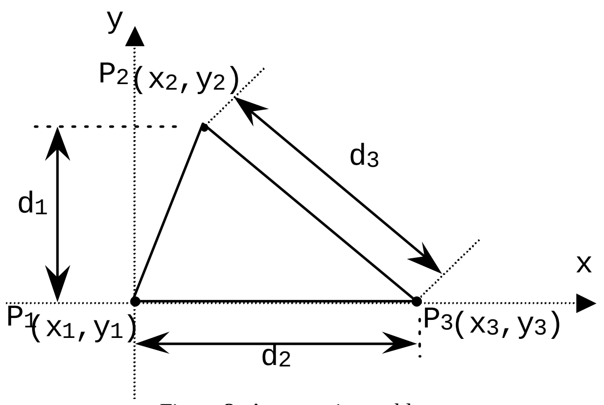


Figure 3: A constraint problem.

3. Bisection method

3.1. Terminology and definitions

An interval, X=[a,b] is a subset of R defined as:

$$[a,b] = \{x \mid a \le x \le b,\ x,a,b \in R\}$$

The numbers a and b are called the bounds of the interval; a is called the lower bound, written lb[a,b], and b, the upper bound, written ub[a,b] (see [17]).

An interval vector X is an ordered n-tuple of intervals: $X = (X_1, X_2, \dots, X_n)$ where $X_i = [a_i,b_i]$. Geometrically this is an n-dimensional box. Note that the real line is embedded in the collection of real intervals. For instance 0=[0,0] and 5=[5,5].

Let $X_1 = [a_1,b_1]$ and $X_2 = [a_2,b_2]$ be two intervals. The binary operations +, -, ., / are defined as:

$X_1 + X_2 = [a_1 + b_1 , a_2 + b_2]$
$X_1 - X_2 = [a_1 - b_2 , b_1 - a_2]$
$X_1 . X_2 = [\min(a_1.a_2 , a_1.b_2 , b_1.a_2 , b_1.b_2), \max(a_1.a_2 , a_1.b_2 , b_1.a_2 , b_1.b_2)]$
$1 / X_1 = [1/b_1 , 1/a_1]$ if $0 \notin X_1$
if $0 \in X_1$
$1 / X_1 = [-\infty , 1/a_1]$ if $b_1 = 0$
$1 / X_1 = [-\infty , 1/a_1] \cup [1/b_1 , +\infty]$ if $a_1 < 0$ and $b_1 > 0$
$1 / X_1 = [1/b_1 , +\infty]$ if $a_1 = 0$

Now, suppose X=[a,b], we define the midpoint of X as m(X)=(a+b)/2 and the width of X as w(X)=b-a.

Similarly, for interval vector $X=(X_1, X_2, \dots, X_n)$, we define the midpoint $m(X)=(m(X_1),m(X_2), \dots ,m(X_n))$ and the width $w(X)=(w(X_1),w(X_2), \dots,w(X_n))$.

If f is a vector valued function of the n real variables $x_1,x_2, \dots,x_n$, then we define the interval extension of f as an interval vector valued function F with interval arguments, for which $F(x_1,x_2, \dots,x_n)=f(x_1,x_2, \dots,x_n)$ for real arguments. That is, as the interval arguments decrease in width, F converges to f.

Suppose that the interval vector $X=(X_1, X_2, \dots , X_n)$ where $X_i=[a_i, b_i]$. We define $|X_i|=\max(|a_i|,|b_i|)$ and the interval vector norm as $||X||= \max_i(|X_i|)$. For interval matrix A (a matrix with interval entries) we define the matrix norm $||A|| = \max_i(\sum_j|A_{ij}|)$.

3.2. The Krawczyk-Moore test

In this section, we consider the following general problem:

Find, with certainty, approximations to all solutions of the non-linear system S:

$$f_i(x_1,x_2, \dots ,x_n) = 0 \ ; \ 1 \le i \le n$$

where bounds a_i and b_i are known such that :

$$a_i \le x_i \le b_i \quad \text{for } 1 \le i \le n$$

The generalized bisection algorithm used here consists of a geometrical bisection process, a root inclusion test and a search algorithm. The geometrical bisection process is similar to that given in [11]. To each box

$$B = (X_1, X_2, \dots , X_n) \text{ where } X_i = [a_i, b_i]$$

we find k such that

$$w(X_k) = \max (w(X_i)) \ ; \ 1 \le i \le n$$

then we form two boxes B1 and B2 such that :

$$B_1 = ([a_1,b_1],[a_2,b_2], \dots ,[a_k,m(X_k)], \dots ,[a_n,b_n])$$
$$B_2 = ([a_1,b_1],[a_2,b_2], \dots ,[m(X_k),b_k], \dots ,[a_n,b_n])$$

That is, we cut the box B into two boxes B_1 and B_2 by cutting B in the co-ordinate direction in which B is longest.

The root inclusion test is a function T_F that associates to each box the values "true", "false" or "unknown". Defined formally in [11], it has the following properties:

i) $T_F(B)$ = "true" implies that there is a unique solution of the system S within B, and Newton’s method or one of its variants will converge to that solution from any starting point in B.

ii) $T_F(B)$ = "false" implies that there are no solutions of the system S within B.

iii) $T_F(B)$ = "unknown" ; the box B must be bisected further.

3.3. Bisection process

The bisection process and the root inclusion test define a binary search algorithm naturally. The Krawczyk-Moore algorithm to find all the solutions of a system of equations within a defined box is given below (see [7,8,10,11,12]). The initial box X is defined by bounds on each individual co-ordinate of the geometric objects. The equations $f=(f_1,f_2, \dots ,f_n)$ of the system correspond to the geometric constraints. Finding the solutions of the system of equations is summarized by the algorithm given below.

Algorithm

```
TEST (X)
Begin
Compute K(X); /* interval computation */
if (K(X) ∩ X) = ∅
then there are no solutions of the system within X
else   /* (K(X) ∩ X) ≠ ∅ */
    r =||I - M.J(X)||;/* I is the identity matrix */
    if (K(X) ⊂ X)
    then if r < 1
        then the simple Newton sequence
             x_{n+1}=p(x_n) will converge to a unique
             solution x* in X from any starting
             point x_0 in X
        else begin BISECT (K(X), X_1, X_2);
                 TEST(X_1); TEST(X_2);
             end
    else
        a=min(w(X))/2;
        if r < 1 and ||m(X) -p(m(X))||< (1-r).a
        then the simple Newton sequence
             x_{n+1}=p(x_n) will converge to a unique
             solution x* in X from any starting
             point x_0 in B(m(X),a).
        else begin BISECT ((K(X) ∩ X), X_1, X_2 );
                 TEST(X_1); TEST(X_2);
             end
End.
```

Where:

$K(X) = m(X) - M.f(m(X)) + [I - M.J(X)].[X - m(X)]$;

J(X) is the jacobian interval matrix;

$M = J(m(X))^{-1}$; $p(x) = x - M.f(x)$.

$B(m(X),a) = \{x \in X : || x - m(X) || \le a \}$;

A box X is said to be "safe" if the root inclusion test is true. Newton’s iteration is performed and a solution is computed. A box X is excluded if there are no solutions in it. Otherwise the box $(K(X) \cap X)$ is subdivided to yield two new boxes X_1 and X_2. The "safety" of these two boxes is then tested.

4. Constraints decomposition

Before applying the bisection process, we must isolate the well-constrained part of the system of equations. For this purpose, we consider the natural bipartite graph associated with systems of equations. This bipartite graph has one vertex per equation, one vertex per unknown, and an edge between an unknown *x* and an equation *f* if and only if *x* appears in equation *f*. This type of graphs has been already used, for example by Serrano in [16]. Let $G=(V, E)$ be this bipartite graph with edges *E* and with vertices *V*. Then $V = F \cup X$ and $F \cap X = \varnothing$. *F* is the set of equations; *X* is the set of unknowns.

Example 2:

The bipartite graph corresponding to the system of equations $\{F_1: x_1 + x_2 - 1 = 0, F_2: x_1 + 2x_2 - 3 = 0\}$ is shown in figure 4.

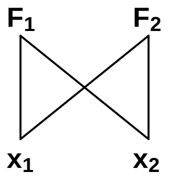


Figure 4. Bipartite graph.

Any bipartite graph can be canonically decomposed in three parts (see [13]): well-constrained, over-constrained and under-constrained. An over-constrained system has more equations than unknowns; its equations are either redundant, or contradictory and thus yield no solution. An under-constrained system has more unknowns than equations and has an infinite and not numerable set of solutions. Both under and over constrained systems cause numerical difficulties if solved by classical methods. The following theorem (see [9] for the proof) describes the decomposition process.

Theorem

Let G = (V={F,X},E) be any bipartite graph. V can be partitioned into three sets :
D, A, C where :

- D is the set of all vertices in G not covered by at least one maximum matching,
- A is the set of all vertices in V - D adjacent to at least one vertex in D,
- C = V - A – D.

These subsets are unique and yield a unique decomposition of G into three sub-graphs G1, G2 and G3 defined by :

- G1 =({C1,C2}, E1) where C1 = C $\cap$ F, C2 = C $\cap$ X and E1 is the set of induced edges of G1,
- G2 =({D1,A2}, E2) where D1 = D $\cap$ F, A2 = A $\cap$ X and E2 is the set of induced edges of G2,
- G3 =({A1,D2}, E2) where A1 = A $\cap$ F, D2 = D $\cap$ X and E3 is the set of induced edges of G3.

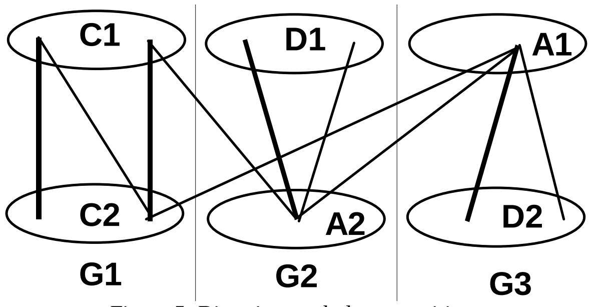


Figure 5: Bipartite graph decomposition.

This general bipartite graph decomposition is shown in figure 5 (equation vertices are drawn above unknown vertices). G1 corresponds to the well-constrained part, G2 to the over-constrained part and G3 to the under-constrained part. Note that G1, G2 or G3 can be empty.

The sub-graphs G1, G2 and G3 can be obtained by the following algorithm (see [1]).

Algorithm:

- **i)** Find a maximum matching M of G.
- **ii)** Build the directed graph G' from G by replacing each edge xf in M by two arcs xf and fx, and by orienting all other edges of the bipartite graph from F to X.
- **iii)** The graph G2 is the set of all descendants of source of G'.
- **iv)** The graph G3 is the set of all ancestors of sinks of G'.
- **v)** G1 = G – G2 – G3.

The steps 2, 3, 4 and 5 can be computed in O(n+m), and the whole algorithm runs in $O(mn^{1/2})$ where n=|V| and m=|E|.

5. Speeding-up the resolution process

After the decomposition of the graph in well- over- an under-constrained parts, we focus on the resolution of the well constrained part. Before the effective numerical solving, we try to divide the resolution process of the whole system of equations into "small" sub-systems. The following algorithm gives the unique decomposition of the well-constrained bipartite graph into its irreducible components and an order of resolution between them.

Algorithm

1. Find a perfect matching M of G.
2. Build the directed graph G' from G by replacing each edge xy in M by two arcs xy and yx, and by orienting all other edges from Y to X.
3. Compute the strongly connected components of G'. Each of these strongly connected components is irreducible.
4. To compute the dependencies between these irreducible subgraphs, build the reduced graph R from G' by contracting each strongly connected

component in a vertex. Each arc of R, say from s_1 to s_2, means : solve subsystem s_2 before s_1 . A compatible total order between subsystems can be obtained by any topological sorting of R.

Steps 2, 3 and 4 can be computed in O(n+m) where n=|V| and m=|E|. The whole algorithm runs in $O(m\ n^{1/2})$. Running time of the whole algorithm including step 1 is bounded by the cost of the search of a perfect matching. The decomposition is independent of the perfect matching *M*.

We present here an illustrative example of a system of constraints and give the corresponding decomposition. A “dimensioning scheme” is shown in figure 6.

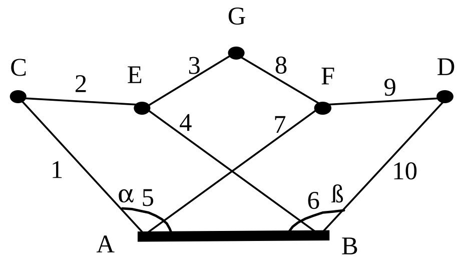


Figure 6: A well constrained scheme.

The points *A* and *B* are initially fixed. The labelled edges correspond to the distance constraints (quadratic equations). α and β are the arguments of the angle constraints.

The system of equations corresponding to this scheme is well constrained. The perfect matching $\{eq1x_C, eq5y_C, eq6x_D, eq10y_D, eq2x_E, eq4y_E, eq7x_F, eq9y_F, eq3x_G, eq8y_G\}$ and the decomposition of *G* into irreducible components are shown in figure 7. The eqi corresponds to the constraint i (for example eq2 is the distance equation between points C and E). The resolution order of these irreducible parts (the strongly connected components) given by step 4 is the following :

$$G_1, G_2, G_3, G_4 \text{ and } G_5$$

We first compute the co-ordinates of the point *C* using the two equations eq1 and eq5, then the co-ordinates of points *D, E, F* and *G* in this order.

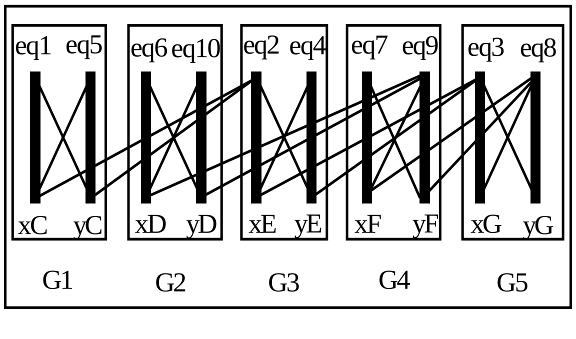


Figure 7: Perfect matching and decomposition

The resolution time of figure 6 is divided by 20 when using this decomposition instead of solving simultaneously the set of equations. The gain can be more important for "huge" reducible systems. For irreducible systems, the decomposition cannot speed up the resolution but the decomposition time is anyway negligible compared to the whole resolution time.

6. Distributed computing

Nevertheless, despite the use of the speeding process, the time needed for the resolution of a system of constraint can remain "important". For this reason, we have decided to "parallelize" the bisection algorithm with system decomposition.

We have managed our tests on 4 personal computers interconnected by an Ethernet LAN. One PC act as a Master Server while the three others act as Slave Servers.

The computing task is distributed as follows: the Master Server divides the initial box, where all the solutions must lie, into four sets of boxes (four loads). It assigns one load to each Slave Server and begins managing the forth load. Each Server is now solving the system of equations with specific boxes. The tree of boxes is generally not balanced (see figure 9) so we do not have any prior information on the time taken to manage a load.

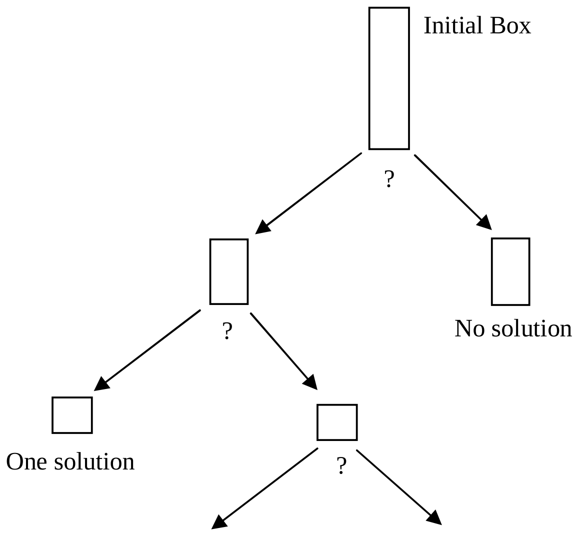


Figure 9: Tree of boxes.

One Server can terminate its task and becomes idle. When a Slave Server terminates, it sends the solutions founds to the Master Server and asks for more boxes if any (figure 10). When receiving this request, the Master Server examines his load. If this load is beyond a predefined threshold (number of boxes exceeding a predefined limit), the Master Slave selects some of its own boxes and sends them to the idle Server for processing. If the Master's load is below the predefined threshold or it was the request initiator Server, then it broadcasts a request of boxes on the network (figure 11). Only heavily loaded Servers respond to that request. The Master Server selects then the busiest Server and transfers some of this Server's boxes to the request initiator Server (figure 12 and figure 13). The processing terminates when all Servers become idle. The Master Server displays then all the solutions.

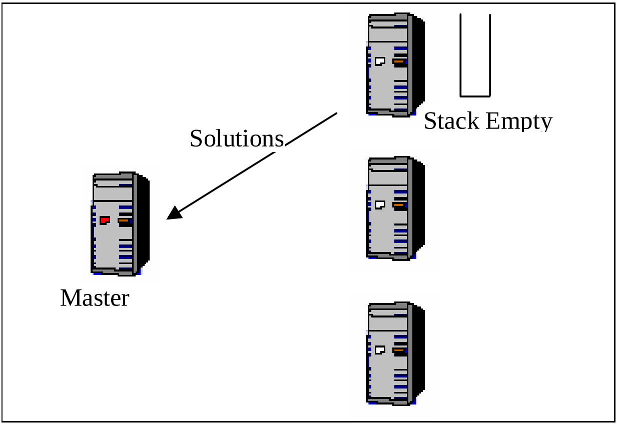


Figure 10: Solutions transfert.

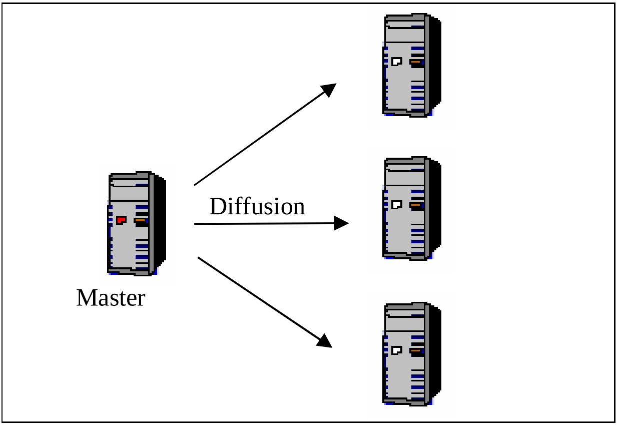


Figure 11: Request diffusion.

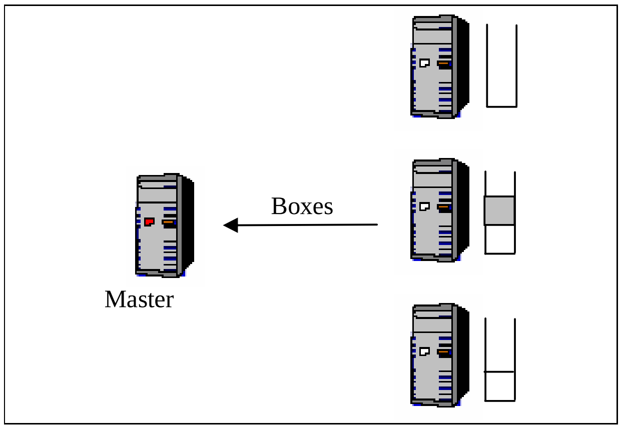


Figure 12: Boxes Transfer.

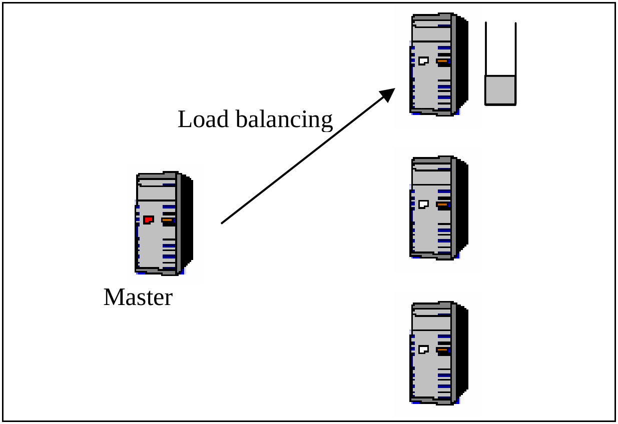


Figure 13: Load balancing.

7. Results

We have implemented a 2D constrained based modeler in C++. The user interacts with the modeler on the Master Server. The user creates points, lines, and circles and specifies constraints in an interactive way. Predefined constraints are: distance between two points or between a point and a line, angle between two lines, tangency between a line and a circle or between two circles, incidence between a point and a line or a circle. The user can specify (with an algebraic formula) any algebraic equation. He can also fix co-ordinates or declare them 'moveable'. Figure 14 shows a screen dump of the user interface.

After constructing and decomposing the bipartite graph, the distributed bisection method is applied for the well-constrained part of the system of equations. For the under-constrained part the user must add constraints. For the over-constrained part the user must remove constraints. The PCs used are IBM PC compatible machine (CPU Pentium IV, 1,7 GHz and 128 MO of RAM). The PCs communicate over a 10 Base-T Ethernet Hub.

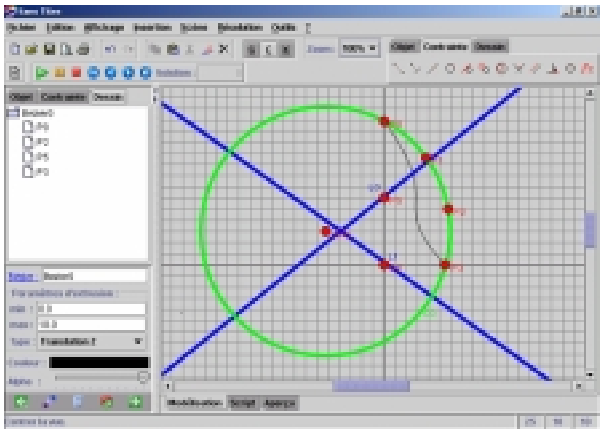
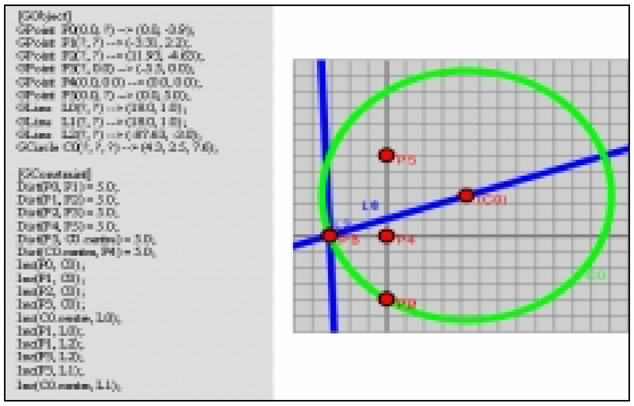


Figure 14: Screen dump of the user interface.

We have implemented two variants of the Krawczyk -Moore algorithm to do the bisection process. The first one is due to Moore and Jones [11] (this method will be named MOJO in the results below) and the second is due to Moore and Qi[12] (named MOQI below). Tables 1 and 2 give the times taken to solve scene1 and scene2 (sketches shown in figure 15 and 16 respectively) on one, two and four PCs. In the initial box for both cases each co-ordinate must be within the interval [-100, +100}. A 16*16 system of equations correspond to scene1. The speeded algorithm decomposes this system in 6 sub-systems of size 1*1, 2*2, 7*7, 2*2, 2*2, 2*2. Forty-eight solutions were found. A 18*18 system of equations correspond to scene2. The speeded algorithm decomposes this system in 8 sub-systems of size 1*1, 1*1, 1*1, 2*2, 1*1, 2*2, 2*2, 8*8. Forty-eight solutions were found.

Table 1. Times to solve Scene 1.

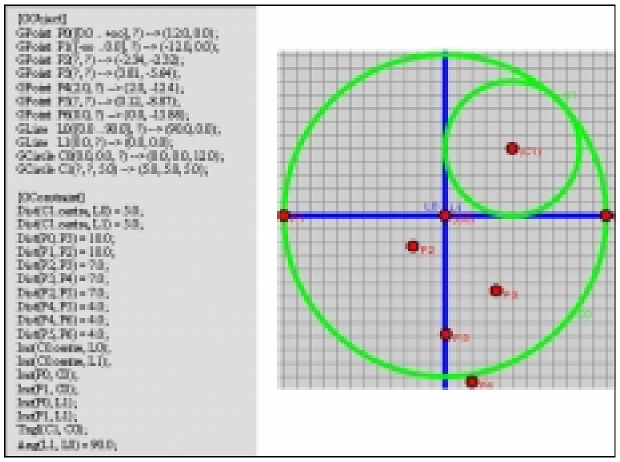
	1 PC	2PC	4PC
MOJO	3m 09s 180ms	1m 39s 183ms	52s 867ms
MOQI	3m 10s 800ms	1m 40s 898ms	54s 004ms



```
[GObject]
GPoint P0(0.0, ?) --> (0.0, -3.9);
GPoint P1(?, ?) --> (-3.30, 2.2);
GPoint P2(?, ?) --> (11.93, -4.63);
GPoint P3(?, 0.0) --> (-3.5, 0.0);
GPoint P4(0.0,0.0) --> (0.0, 0.0);
GPoint P5(0.0, ?) --> (0.0, 5.0);
GLine  L0(?, ?) --> (18.0, 1.0);
GLine  L1(?, ?) --> (18.0, 1.0);
GLine  L2(?, ?) --> (-87.64, -2.0);
GCircle C0(?, ?, ?) --> (4.3, 2.5, 7.0);

[GConstraint]
Dist(P0, P1) = 5.0;
Dist(P1, P2) = 5.0;
Dist(P2, P3) = 5.0;
Dist(P4, P5) = 5.0;
Dist(P3, C0.centre) = 5.0;
Dist(C0.centre, P4) = 5.0;
Inc(P0, C0);
Inc(P1, C0);
Inc(P2, C0);
Inc(P3, C0);
Inc(C0.centre, L0);
Inc(P1, L0);
Inc(P1, L2);
Inc(P0, L2);
Inc(P3, L1);
Inc(C0.centre, L1);
```

Figure 15. Sketch of scene1.



```
[GObject]
GPoint P0([0.0 .. +oo], ?) --> (12.0, 0.0);
GPoint P1([-oo .. 0.0], ?) --> (-12.0, 0.0);
GPoint P2(?, ?) --> (-2.34, -2.32);
GPoint P3(?, ?) --> (2.81, -5.64);
GPoint P4(2.0, ?) --> (2.0, -12.4);
GPoint P5(?, ?) --> (0.13, -8.87);
GPoint P6(0.0, ?) --> (0.0, -13.86);
GLine  L0([0.0 .. 90.0], ?) --> (90.0, 0.0);
GLine  L1(0.0, ?) --> (0.0, 0.0);
GCircle C0(0.0, 0.0, ?) --> (0.0, 0.0, 12.0);
GCircle C1(?, ?, 5.0) --> (5.0, 5.0, 5.0);

[GConstraint]
Dist(C1.centre, L0) = 5.0;
Dist(C1.centre, L1) = 5.0;
Dist(P0, P2) = 18.0;
Dist(P1, P2) = 18.0;
Dist(P2, P3) = 7.0;
Dist(P3, P4) = 7.0;
Dist(P3, P5) = 7.0;
Dist(P4, P5) = 4.0;
Dist(P4, P6) = 4.0;
Dist(P5, P6) = 4.0;
Inc(C0.centre, L0);
Inc(C0.centre, L1);
Inc(P0, C0);
Inc(P1, C0);
Inc(P0, L1);
Inc(P1, L1);
Tang(C1, C0);
Ang(L1, L0) = 90.0;
```

Figure 16: Sketch of scene2.

Table 2. Times to solve Scene 2.

	1 PC	2PC	4PC
MOJO	8m 38s 279ms	4m 34s 230ms	2m 21s 900ms
MOQI	8m 45s 666ms	4m 40s 450ms	2m 25s 212ms

8. Conclusion

We have implemented a 2D constraint-based modeler using a distributed bisection. We can note that all the techniques used here can be used to solve systems of 3D constraints. This is because bisection also applies for 3D geometric constraints. The "parallelization" technique allows us to exploit the available computing distributed resources. The results obtained were promising. But we must undertake several tests on a larger network to study the effects of the overheads on the whole processing time. The technique of load balancing used here must be compared with others techniques to select the "best" method. We think that the distributed bisection will have a major place among numerical methods for solving geometric constraints.

References

[1] Ait-Aoudia S., Jegou R. and Michelucci D., "Reduction of Constraint Systems", *In Proceedings of Compugraphics*, (Alvor, Portugal), 83-92, 1993.

[2] Ait-Aoudia S., Brahim H., Moussaoui A. and Saadi T., "Solving Geometric Constraints by a Graph Constructive Approach", *IEEE International Conf. on Information Visualization*, (London, England), pp. 250-255, July 99.

[3] Fudos I. and Hoffman C.M., "A Graph-Constructive Approach to Solving Systems of Geometric Constraints", *ACM Trans. on Graphics*, Vol. 16, No. 2, April 1997, 179-216.

[4] Gao X.S. and Chou S.C., "Solving Geometric Constraint Systems : a symbolic approach and decision of Rc-constructibility", *Computer Aided Design*, pp. 115-122. vol. 30, n°. 3, 1998.

[5] Hoffman C., Lomonosov A. and Sitharam M., "Geometric constraint decomposotion". *In B. Bruderlin and D. Roller editors*, Geometric constraint solving and applications, pages 170-195, Springer 1998.

[6] Jermann C., Trombettoni G., Neveu B. and Rueher M., "A constraint programming approach for solving rigid geometric system", *6th Conf. on Principles and Practice of*

constraint programming, pp 233-248, Singapore sep. 2000.

[7] Kearfott R.B., "Some tests of Generalized Bisection", *ACM Transactions on Mathematical Software*, Vol. 13, No. 3, 197-220, 1987.

[8] Krawczyk R., "Newton algorithmen zur bestimmung *von* nullstellen mit fehlerschranken", *Computing,* 4. pp 187-201, 1969.

[9] Lovasz L. *and* Plummer M.D., *Matching Theory,* North-Holland, 1986.

[10] Moore R.E., *Interval Analysis*, *Prentice Hall,* Englewood Cliffs, N.J, 1966.

[11] Moore R.E. and Jones S.T., "Safe starting regions for *iterative* methods", *SIAM J. Numerical Analysis,* vol. 14, num. 6, pp 1051-1065, Dec. 1977.

[12] Moore R.E. and QI, L., "A Successive Interval for *Nonlinear* Systems", *SIAM J. Numerical Analysis*, Vol. 19, No. 4, 845-850, 1982.

[13] Murota K., *Systems Analysis by Graphs and Matroids, Solvability and Controllability,* Springer Verlag 1987.

[14] Owen J.C., "Algebraic Solution for Geometry from Dimensional Constraints", *Symp. on Solid Modeling Foundations and CAD/CAM Applications*, 397-407, 1991.

[15] Perez A. and Serrano D., "Constraint base analysis tools for design", *2nd Symp. on Solid Modeling Foundations and CAD/CAM Applications*, Montreal Canada, pp. 281-290, Mai 1993.

[16] Serrano D., "Automatic Dimensioning in Design for Manufacturing", *Symp. on Solid Modeling Foundations and CAD/CAM Applications,* 379-386, 1991.

[17] Snyder J.M., "Interval Analysis For Computer Graphics", *Computer Graphics*, Vol. 26, No. 2, 121-130, 1992.

[18] Sunde G., "Specification of shape by dimensions and other geometric constraints", *Geometric modeling for CAD applications*, pp. 199-213. North-Holland, IFIP, 1988.

[19] Suzuki H., Ando H. and Kimura F., "Variation of geometries based on a geometric-reasoning method", *Computer and Graphics*, 14(2), pp. 211-224. 1990.

[20] Verroust A., Schonek F. and Roller D., "Rule-oriented method for parametrized computer-aided design", *Computer Aided Design*, 24 (3), pp. 531-540, Oct. 1992.

Samy Ait-Aoudia received a DEA "Diplôme d'Etudes Approfondies" in image processing from Saint-Etienne University, France, in 1990. He holds a Ph.D. degree in computer science from the Ecole des Mines, Saint-Etienne, France, in 1994. He is currently "Maître de Conférences" at the Computer Science Institute in Algeria. He teaches different modules at both BSc and MSc levels in computer science and software engineering. His areas of research include CAD/CAM, constraint management, solid Modelling and parallel algorithms.

Imad Manaa received a Bachelor's degree in computer science from the Computer Science Institute - Algeria, in 2003. He is currently pursuing a Master degree in computer science. He has worked on a project involving the developement of a constraint based CAD software. His current research interest includes distributed systems, parallel algorithms, and web applications.