SOLVING GEOMETRIC CONSTRAINTS IN A PARALLEL NETWORK

Robert B. Fisher
Mark J. L. Orr

Department of Artificial Intelligence
Edinburgh University

ABSTRACT

We describe a network implementation of the SUP-INF method of solving sets of inequalities that has advantages over previous implementations. The cost of symbolic manipulation is transferred to compile-time allowing speed up at run-time due to parallel evaluation. Further, allowing iteration in the network improves the competence of the method when working with non-linear expressions. We use the network to implement a geometric reasoner for a computer vision program and show that it meets the general requirements for such a system.

1. INTRODUCTION

In a previous paper (Orr 1987a) we investigated the general nature of a geometric reasoner for computer vision, concerned more with specification than implementation. Amongst the conclusions we made, was that algebraic inequalities were a suitable representation for constraints, because it was a uniform mechanism for a variety of model-data relationships, it represents both *a priori* and observed relationships and easily allows model variation. Constraints generally involve unknown quantities not measured from the image or specified in the models and for which estimates are sought. To find estimates it is necessary to combine related constraints and this involves solving sets of algebraic inequalities.

This approach has already been demonstrated by Brooks (Brooks 1981). A crucial part of his constraint solving mechanism was a symbolic manipulator for algebraic expressions. This had drawbacks of a high cost for symbolic processing and an inability to properly handle non-linear constraints.

Acknowledgements
This work was performed under Alvey Grant GR/D/1740.3.

Section 2 describes an implementation of the SUP-INF method that does away with the need for symbolic manipulation at run-time, has a natural parallel structure and copes better with non-linear constraints. In section 3 we show how this implementation can be used for geometric reasoning and how it meets the specifications outlined in our previous paper (Orr 1987a).

2. SOLVING CONSTRAINTS IN PARALLEL

The basic constraint solving method we use is Bledsoe's SUP-INF algorithm (Bledsoe 1975), later refined by Shostak (Shostak 1977) and Brooks (Brooks 1981). Constraints are expressed in the form:

$$x_i \leqslant f_i$$

or

$$x_i \geqslant g_i$$

where the x_i are members of a set $\{x_1, x_2, \ldots x_n\}$ of variables and f_i and g_i are values or expressions involving some or all of the x_i. A solution of the constraints would be a substitution of real values for the variables that maintained the truth of each inequality. The goal of the algorithm, for a given set of constraints, is:

(1) to decide whether the set of possible solutions is empty,

(2) to find bounds on the value that a given expression (involving some or all of the x_i) can attain over a non-empty set of solutions.

The algorithm is based on the recursive application of the functions SUP and INF on the expression to be bound and its sub-expressions. SUP returns an upper bound (supremum) and INF a lower bound (infimum). In Brooks' (Brooks 1981) program the simplification of constraints and the application of SUP and INF was handled by symbolic manipulation at run time. We present a new implementation of the SUP-INF method that has the following advantages:

AVC 1987 doi:10.5244/C.1.12

(1) it transfers the cost of symbolic manipulation from run-time to compile-time,

(2) it improves the performance of the algorithm for non-linear constraints,

(3) it has a natural parallel structure.

The implementation has the structure of a network with nodes and connections as described below.

Structure of the network

The network consists of two types of nodes: value nodes and operation nodes. The value nodes acquire numerical SUP and INF bounds on their associated algebraic variable or expression. The bounds are computed from connections with other value nodes or with operation nodes that receive inputs from other value or operation nodes. Each time new bounds are computed the change propagates over the network causing other nodes to acquire new bounds. The changes become smaller as the bounds get closer and the network converges asymptotically to a stable state when the desired bounds on variables or expressions of interest can be extracted from the associated value nodes.

Network Creation

A network is constructed by linking together several network fragments or modules. Each module represents a particular instance of a common constraint type and there may be more than one module of the same type in the network. The structure of modules is defined by an off-line compilation process. Consequently, the on-line program which uses the network, such as the geometric reasoner described in section 3, only has to connect instances of the appropriate modules to solve the problem at hand.

A module is compiled from a list of algebraic inequalities such as:

$$x \leqslant y + z$$

The inequalities are written by a human programmer after due consideration of the 'problem' that the module 'solves'. An example from geometric reasoning (section 3) is the problem of relating two pairs of direction vectors with a rotation when the relation between the paired vectors is that one is the rotation of the other. The relations between all the (scalar) variables occurring in the problem are expressed as inequalities. These give both exact relationships and 'heuristic' bounds. If an equality is encountered then it is split into two inequalities:

$x = expr$ becomes:

$$x \leqslant expr \;\&\; x \geqslant expr$$

If a product is encountered, then it is split into four inequalities involving the signed reciprocal ('srecip') function:

$x * y \leqslant z$ becomes:

$$x \leqslant z * srecip(y)$$

$$y \leqslant z * srecip(x)$$

$$x \geqslant -z * srecip(-y)$$

$$y \geqslant -z * srecip(-x)$$

This function has the definition:

$srecip(x) =$ if $x > 0$ then $1/x$
else 'undefined'

and consequently has the effect of turning off and on constraints according to the sign of its argument.

Recursive constraints are allowed such as:

$$x^2 \geqslant 1 - y^2$$

which becomes:

$$x \geqslant (1 - y^2) * srecip(x)$$

$$x \leqslant (y^2 - 1) * srecip(-x)$$

but are treated differently by bound simplification (see below).

Symbolic manipulation

Before compiling the network, the list of inequalities is checked for correct syntax, simplified and processed by the functions SUP and INF. In general this is a hard problem but the constraint manipulation system (CMS) of Brooks' program ACRONYM (Brooks 1981) at least provides some competence. We have extended this CMS to cope with square roots, powers of variables, symbolic rather than numeric bounds on products where appropriate, the unsigned reciprocal function and the undefined value (Fisher 1987b).

Simplification is only applied to non-recursive constraints where the variable on the left hand side of the inequality does not appear anywhere in the right hand side. Recursive constraints are difficult to handle and usually get

reduced to the trivial:

$$-\text{infinity} \leqslant x \leqslant +\text{infinity}$$

The CMS could be used directly (as in ACRONYM) by the on-line program. Measurements made by the program would add new constraints providing more scope for simplification and eventually to bounds on variables and expressions that are not measured directly. However, symbolic reasoning is computationally expensive and not suited to wide scale parallelism.

A more compelling reason for using a network is that it can iterate to better bounds over non-linear constraints than the single pass method of the CMS. Consider the following example.

$$x \leqslant 1 + 1/y$$

$$y \geqslant 1 + 1/x$$

$$0.1 \leqslant x \leqslant 10$$

$$0.1 \leqslant x \leqslant 10$$

The CMS (somewhat simplified) finds:

$$\text{SUP}(x) = 1 + 1/\text{INF}(y)$$
$$= 1 + 1/(1 + 1/\text{SUP}(x))$$

When it gets to the embedded SUP(x) it uses the numerical bound 10 to produce:

$$\text{SUP}(x) = 1 + 1/(1 + 1/10)$$
$$= 1.91$$

However the network computation iterates to the (analytically) best bound:

$$\text{SUP}(x) = 1.62$$
$$= (1 + \sqrt{5})/2$$

Network Compilation

Value nodes are created for all variables occurring in the constraint list. These are connected by various operator nodes that extract values from value nodes or other operators. The connections are determined by the expressions found in the constraints. The following is a list of the actions taken by the compiler when it encounters the specified expression type:

constant:
An operation node (with no inputs) is created that supplies the given constant.

variable:
An operation node is created that extracts the SUP (or INF) of the associated value node.

plus: An operation node is created that adds the results of the recursively compiled subexpressions.

max (or min):
SUP(max(list)) is compiled to be max(SUP(list)) (analogously for INF and 'min'). Thus subfragments for each subexpression in the list are created and linked to a series of connected binary 'max' (or 'min') nodes. Network evaluation is different for max (or min) nodes created from SUP or INF in their use of defaults when not all arguments are evaluated (which may arise from timing delays or alternative expressions being undefined). The INF max function returns a value if at least one argument is evaluated; the SUP max function only returns a value when all arguments are evaluated.

times:
SUP(A*B) is expanded to:

```
max(INF(A)*INF(B),
    INF(A)*SUP(B),
    SUP(A)*INF(B),
    SUP(A)*SUP(B))
```

and then compiled. The same for INF(A*B) except 'max' is replaced by 'min'.

recip(E) (where E is an expression):
A test-case node is required for the reciprocal function. Test-case nodes select their output according to a test defined at compile-time and carried out at run-time. If SUP is the desired bound, the test-case construction is:

```
if INF(E)>0 or SUP(E)<0

then 1/INF(E)

else plus_infinity
```

If INF is the desired bound then:

```
if INF(E)>0 or SUP(E)<0

then 1/SUP(E)

else minus_infinity
```

srecip(E) (where E is an expression):
This is the signed reciprocal function where:

srecip(x) = if x > 0 then 1/x
else 'undefined'

If SUP is the desired bound, a test-case node is created selecting:

if INF(E)>0

then 1/INF(E)

else 'undefined'

If INF is the desired bound then the test-case construction is:

if INF(E) > 0

then 1/SUP(E)

else 'undefined'

v^n (where v is a variable and n is odd):
A sequence of 'times' operation nodes are created and linked to the SUP (or INF) of the variable. The output of each 'times' operation becomes the input to the next.

v^n (where v is a variable and n is even):
If SUP is the desired bound then sequences of 'times' nodes are created and linked to both the INF and SUP of the variable and a final 'max' node linked to the output of each sequence. If INF is the desired bound then a 'test-case' node is created selecting:

if SUP(v) < 0

then $[SUP(v)]^n$

else if INF(v) > 0

then $[INF(v)]^n$

else 0

square_root(E) (where E is an expression):
The positive square root is assumed. If SUP is the desired bound then:

if SUP(E) $\geqslant$ 0

then sqrt(SUP(E))

else 'undefined'

If INF is the desired bound:

if INF(E) $\geqslant$ 0

then sqrt(INF(E))

else 'undefined'

As the same expressions may be used more than once in different constraints in the same module, the recursive compiler uses a previous compilation for a expression if one exists, thus avoiding duplication. Another simplification is the reduction of multiple constraints to a single 'min' or 'max' function:

$v \leqslant E_1$, $v \leqslant E_2$, ... becomes:

$v \leqslant \min(E_1, E_2, ...)$

A similar simplification is performed for lower bounds using the 'max' function.

To illustrate the creation of a network module, suppose we are interested in the 'problem':

$$A \leqslant B - C$$

which entails the further constraints:

$$B \geqslant A + C$$

$$C \leqslant B - A$$

This list of constraints would be the input to the CMS (Fisher 1987b), that would have little to simplify but would recursively apply the SUP and INF functions symbolically to find:

SUP(A) = SUP(B) - INF(C)

INF(B) = INF(A) + INF(C)

SUP(C) = SUP(B) - INF(A)

The compiler then produces the network shown in figure 1. This is a trivial example that even fails to compute both bounds on the parameters involved. In practice (see section 3) modules are larger and more complicated.

Modularisation

The run-time program constructs and evaluates its own networks according to the problems it is presented with. We assume that problems can be broken down into several parts each of which can be managed by an instance of some previously compiled module. Suppose we have the following two constraints:

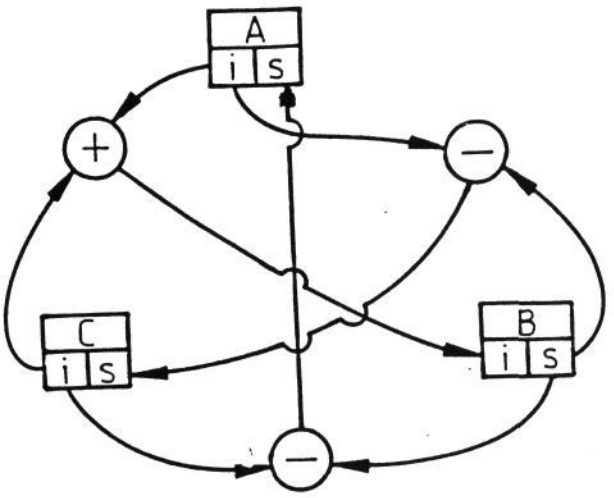

Figure 1: the network for $A \leqslant B - C$.

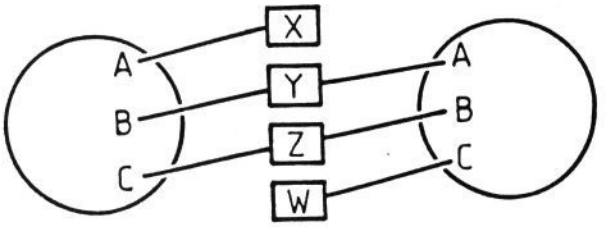

Figure 2: two connected modules.

$$x \leqslant y - z$$

$$y \leqslant z - w$$

A network for this problem would be constructed out of two instances of the module defined above for the constraint type:

$$A \leqslant B - C$$

and connected as shown in figure 2. The modules can be thought of as black boxes with connections to the outside world. For the first constraint the connections A->x, B->y and C->z are made, while for the second constraint A->y, B->z and C->w.

Network Evaluation

The values at each node are computed using the values at the connecting nodes. The SUP (INF) computation chooses the minimum (maximum) of each of its current bounds and its current value. Including the current value in the calculation ensures that bounds can only get tighter. Thus if:

$$SUP(A) \leqslant a_1,\ SUP(A) \leqslant a_2,\ \ldots$$

then:

$$SUP(A_{t+1}) = \min(SUP(A_t),\ a_1,\ a_2,\ \ldots)$$

is the updating function for the supremum of A from time t to time t+1.

The networks of modules are designed to be evaluated in parallel. The whole network could be evaluated synchronously or asynchronously in a MIMD processor with non-local connectivity. Ideally, each node would be stored in a separate processor, continually polling its inputs and updating its output if appropriate.

So far we only simulate the network serially. When the change at a node drops below a preset threshold its dependent nodes no longer require re-evaluation. When no more nodes need to be evaluated, the network has reached a stable state and processing can stop. The other way of stopping the network is the detection of an inconsistency signaled by a pair of bounds crossing over (the SUP of some value node becomes lower than its INF).

3. GEOMETRIC REASONING FOR COMPUTER VISION

We have previously (Orr 1987a) characterised the tasks carried out by a geometric reasoner for computer vision as a small set of functions that operate on and return certain types of objects, amongst which was the type 'position'. The five primitive functions needed are:

LOCATE:
deduces position constraints from pairings of model features to image features.

MERGE:
integrates separate position estimates.

TRANSFORM:
transforms one position by another (ie. given B relative to A and C relative to B find

C relative to A).

INVERSE:

transforms the position of A relative to B into B relative to A.

PREDICT:

deduces what data should be present in the image, given a position and a model feature.

We also noted that the position data type was required to represent uncertainty because of the inherent errors of image measurements.

That paper only addressed the question of what must a geometric reasoner do, leaving open the question of how it does it. We described above an implementation based on networks of algebraic constraints. In this section we will describe specific networks that can be used for geometric reasoning and show how the they meet the requirements of our previous paper.

Underlying our implementation is the use of algebraic constraints to represent knowledge about the world. For example, to represent a position requires specifying upper and lower bounds on each of the six degrees of freedom. This satisfies the requirement of representing uncertainty because the true value of each parameter can lie in a range.

The TRANSFORM function is implemented as a network module. Looked at as a black box, the TRANSFORM module has three sets of ports to the outside world representing three positions (18 parameters in total): the position being transformed, the transforming position and the resulting position. When operating in the context of an evaluating network, if any two of the sets of ports receive bounds from outside, the module will reflect the new situation by setting new bounds on the third set of ports. The INVERSE function is implicit in the network through the bi-directionality of all modules involving positions.

The MERGE function is carried out at the nodes linking the ports from different modules. Each port is 'saying something' about the bounds on some variable and if two or more ports are linked then they either agree (the bounds intersect) or disagree. In the latter case, an inconsistency has been detected - precisely what the MERGE function was designed to do. Further, the intersection of the bounds also improves the estimates of the variables.

The function LOCATE is implemented by a series of network modules, one for each general type of constraint derivable from the possible pairings between model features and image features. These features vary from one vision system to the next and the corresponding constraints will also differ. In Brooks' ACRONYM (Brooks 1981) the models were 3D volumetric primitives while the image features were 2D. Our own system, IMAGINE (Fisher 1986), is based on 3D models augmented by 2D viewpoint information (Fisher 1987a) and 3D stereo images.

In our case, most pairings between a model and an image feature can be reduced to a set of pairings between 3D direction and location vectors. The general constraint types are then distinguished by the numbers of location and direction pairings involved. Suppose we have the two pairings:

(1) a straight model line with an image edge, and

(2) a conical model surface with a similar image surface

Each of these can be broken down into the pairing of a direction vector and a point location in the model with a direction and location in the image. Even though the features involved are completely different, they both present the same general type of constraint and therefore can be 'LOCATEd' by the same type of network module.

Part of our vision system is a catalogue of constraints (Orr 1987b) available from pairings between feature types in our modeling scheme and their corresponding image feature types. Whenever the geometric reasoner encounters a new hypothesis (feature pairing) it can look up the type of pairing in the catalogue and find out:

(1) which data vectors to pair with which model vectors, and

(2) which network modules to use.

This enables it to create new instances of the modules and link them into the already existing and evaluated network. This network reflects constraints already integrated and may contain many modules created from several previous hypotheses. If the network was previously in a consistent state but, after the introduction of the new module, becomes inconsistent then the reasoner can deduce that something was amiss with the new hypothesis. Otherwise, the new module may help to decrease the uncertainty in the network by pushing some bounds closer together. Thus the geometric reasoner fulfills its role of aiding image interpretation.

Every module implementing a LOCATE function (where the input is from the image and models and the output is a position constraint) also implements a PREDICT function (where the input is from a position constraint and the models and the output is a constraint on something in the image). There is a symmetry between the two functions because the network module does not distinguish which of its ports are for input and which are for output. It simply maintains a set of relations between the values current at each of its ports.

Analysis of the geometric constraints used so far has shown repeated patterns in their algebraic structure and hence that the constraints can be constructed via the composition of six primitive structures:

(1) $| s_1 - s_2 | \leq$ threshold — two scalers are close

(2) $\| \underline{v}_1 - \underline{v}_2 \| \leq$ threshold — two direction vectors are close

(3) $\| \underline{l}_1 - \underline{l}_2 \| \leq$ threshold — two points are close

(4) $P_1(P_2) = P_3$ — a position transformed by a position gives a position

(5) $P(\underline{v}_1) = \underline{v}_2$ and $P(\underline{v}_3) = \underline{v}_4$ — a pair of direction vectors transformed by a position gives a pair of direction vectors (both relations need not be used, but are necessary for completely deducing the position from the paired vectors)

(6) $P(\underline{l}_1) = \underline{l}_2$, $P(\underline{l}_3) = \underline{l}_4$ and $P(\underline{l}_5) = \underline{l}_6$ — three points transformed by a position give three points (again, only one or two points may be used)

Thus, the constraint that says a transformed model direction vector ($\underline{m}$) must lie within a distance (t) of an observed data vector ($\underline{d}$)

$$| P(\underline{m}) - \underline{d} | \leq t$$

can be decomposed into instances of modules 2 and 5 connected.

The network modules are defined in terms of the algebraic relationships between the interface variables. When compiled, they mainly consist of operation nodes. The sizes of the six modules (in operation nodes) are:

module 1: 26 nodes	module 4: 2381 nodes
module 2: 826 nodes	module 5: 1589 nodes
module 3: 297 nodes	module 6: 1225 nodes

We conclude this section with an example of estimating an object's 3D orientation. Assume the following model direction vectors

$$\underline{m}_1 = (-0.51, 0.83, 0.22)$$
$$\underline{m}_2 = (0.68, -0.23, 0.69)$$

are rotated rigidly by position P to give the vectors $P(\underline{m}_1)$ and $P(\underline{m}_2)$. Then, assume we observe two data vectors

$$\underline{d}_1 = (-0.40, 0.91, 0.04)$$
$$\underline{d}_2 = (-0.52, -0.67, 0.51)$$

that are paired with $\underline{m}_1$ and $\underline{m}_2$ respectively. (This pairing results from the model-based scene analysis. An example pairing might arise from using a 3D orientation discontinuity and the normal of a planar surface patch.)

The pairings are represented using an instance of module 5 from above, linked to the $\underline{m}_i$ and $\underline{d}_i$ vectors. Evaluating this simple network, the following bounds are achieved on the rotation (represented as a quaternion):

Low (0.729, 0.249, -0.622, -0.141)
High (0.731, 0.252, -0.618, -0.139)

where the true value is

(0.73, 0.25, -0.62, -0.14)

The result required 46 network update cycles with 3919 operation node evaluations, where all evaluations in each cycle could be executed in parallel. This gives the time necessary for values to completely propagate through the layers of simple function units several times before convergence.

Now, suppose instead we anticipate that the vectors $\underline{d}_i$ are displaced from their true position by some observational error of maximum magnitude E. We then know

$$\| P(\underline{m}_i) - \underline{d}_i \| \leq E$$

so we can now use two instances of module two.

Evaluating this network with E = 0.05 produces the new bounds on the estimated rotation:

Low (0.615, 0.149, -0.810, -0.220)
High (0.804, 0.370, -0.438, -0.076)

which contains the correct rotation given above (63 network update cycles involving 7591 node evaluations).

An alternative to using module two entails setting the bounds on the $\underline{d}_i$ directly, which allows for non-isotropic errors. More exact error relationships could be represented algebraically and linked by other modules.

If we make a third model-to-data vector pairing, then we connect new instances of modules two and five. This final network is shown in figure 3. Suppose this new vector is observed with E = 0.02. Then, the new estimated rotation is:

Low (0.652, 0.173, -0.805, -0.194)

High (0.799,0.341,-0.464,-0.097)

which has reduced variance (52 network update cycles involving 12360 node evaluations). The average estimated parameter value is:

(0.725,0.257,-0.634,-0.145)

which compares favourably with the true value. Note that the bounds give the full range of allowable variation, instead of a statistical estimate.

This example demonstrates combining solutions to several constraint problems to fully constrain a larger problem. In practise, as more evidence is obtained, more network modules can be connected to further refine parameter estimates, especially as the pre-defined modules are usable for a variety of geometric relationships.

4. RELATED WORK

The use of algebraic inequalities to represent geometric constraints derives from Brooks' ACRONYM (Brooks 1981), as does the symbolic constraint manipulation methods. The network computation is similar to the many relaxation or constraint satisfaction algorithms that are suitable for parallel processing. However, it differs from the relaxation algorithms in that it is not a probabilistic labelling computation and from constraint satisfaction in that there is reduction of an infinite continuous range of values rather than selection from a finite set of discrete values. While the network relies on connections between units, the computation is not in the distributed connectionist form where the results are expressed as states of the network. Instead, the results are the values current at selected processors.

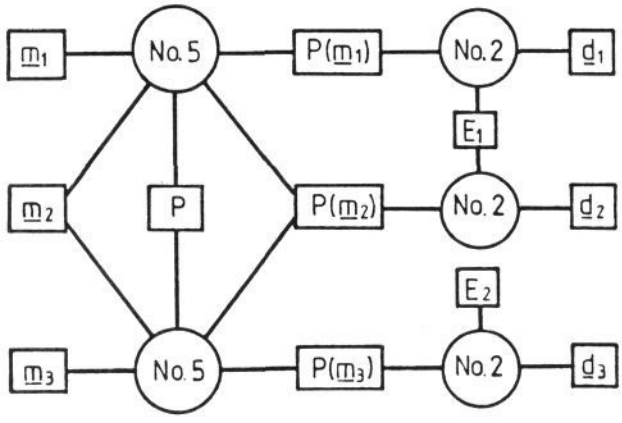

Figure 3: object orientation from three pairs of direction vectors

The work presented here differs significantly from two other network based geometric reasoning systems. Hinton and Lang (1985) learned and deduced positions of 2D patterns using a distributed connectionist network, whose intermediate nodes represented object position and gated connections between iconic image and model representations. Ballard and Tanaka (1985) demonstrated a 3D reasoning network whose nodes represent instances of parameter values and whose connections represent consistency according to model-determined algebraic relationships. In both cases, patterns of network activity result, with the dominant pattern accepted as the answer (unlike here, where the result is explicit). Both systems also simultaneously select a model, which is treated separately in our analysis.

5. SUMMARY

The methodology we have investigated is summarised here. We start with sets of algebraic constraints associated with particular geometric relationships (grouped into 6 primitive classes of relationships). Image observables are represented by variables at this stage. These constraints are then processed by a CMS to produce symbolic bounds on each variable. The bounds are compiled into a network where the structure of the network reflects the structure of the expressions for the bounds. When observable variables get bound to measured values the other variables (position or model parameters) are forced into consistency by evaluating the network, which is defined in a form suitable for parallel evaluation. Defining the constraints and applying the CMS can be done in advance. Then, at run time, networks solving particular problems can be built by connecting instances of the compiled constraint modules, according to the structure of the problem.

ACRONYM's CMS was optimal when producing numerical bounds on single variables over sets of linear constraints. Since we reproduce the symbolic reasoning in the network, only substituting data values later, the network must have the same performance over linear constraint sets. Here, as the geometric constraints are mostly non-linear, we cannot expect optimality, but our extensions to the CMS and iterative evaluation in

the network improve the performance.

While the network structure is capable of implementing many computations, we have used it for geometric reasoning, showing how the standard reasoning functions and the use of 3D models and 3D images require the definition of various network modules. We demonstrated use of the network to support geometric reasoning with an example of estimating an object's 3D orientation.

REFERENCES

1 Ballard, D. and Tanaka, H., "Transformational Form Perception in 3D: Constraints, Algorithms and Implementation", Proc. 9th Int. Joint Conf. on Artif. Intel., pp 964-968, 1985.

2 Bledsoe, W.W., "A new method for proving certain Presburger formulas", Proc. 4th Int. Joint Conf. on Artif. Intel., pp15-21, 1975.

3 Brooks, R.A., "Symbolic reasoning among 3-D models and 2-D images", Artif. Intel., 17, pp285-348, 1981.

4 Fisher, R.B., "From surfaces to objects: recognising objects using surface information and object models", PhD thesis, University of Edinburgh, 1986.

5 Fisher, R.B., "SMS: a suggestive modeling system for object recognition", Image and Vision Comp., 5, pp98-104, 1987a.

6 Fisher, R.B., "A PROLOG version of ACRONYM's CMS", Dept. of Artif. Intel., working paper ***, Edinburgh University, 1987b.

7 Hinton, G. and Lang, K., "Shape Recognition and Illusory Conjunctions", Proc. 9th Int. Joint Conf. on Artif. Intel., pp 252-259, 1985.

8 Orr, M.J.L., "Geometric reasoning for computer vision", to appear in Image and Vision Comp., August, 1987a.

9 Orr, M.J.L., "Geometric constraints in 3D computer vision", Dept. of Artif. Intel., working paper ***, Edinburgh University, 1987b.

10 Shostak, R.E., "On the SUP-INF method for proving Presburger formulas", J. Assoc. Comp. Mach., 24, pp529-543, 1977.