International Journal of Shape Modeling, Vol. 6, No. 2 (2000) 131–143
© World Scientific Publishing Company

CONFIGURATIONS, CONSTRAINTS, AND CSG

ADRIAN BOWYER
Department of Mechanical Engineering, University of Bath
Bath BA2 7AY, United Kingdom

DAVID EISENTHAL
D-Cubed Ltd.
Cambridge, United Kingdom

DAN PIDCOCK
Otto Bock UK Ltd.
Egham, Surrey, United Kingdom

and

KEVIN WISE
Silverscreen Productions Ltd.
Wellington, New Zealand.

Received (July 21, 2000)
Revised (September 27, 2000)
Communicated by (M.-S. Kim and R. Martin)

This paper presents a multidimensional CSG geometric modeller. It then shows how this modeller can be used to produce a compact representation of the complete configuration-space map of a mechanism composed of polyhedral parts. The map is derived directly from the three-dimensional geometric model that describes the mechanism, and allows more degrees of freedom than any other current system. Methods of exploring the map to give the kinematics of the mechanism will also be described and examples will be given. We will then go on to show how geometric constraints can be reformulated as a multidimensional CSG expression describing the constraints problem in a space similar to the configuration space above. Geometric constraints representing mechanisms will be described and the resulting mechanisms shown. Finally we will show how the two techniques, which are both coded using the same multidimensional modeller, can be combined. This allows parts of a machine that are known (such as slides and hinges) to be described explicitly using constraints, and parts that are unknown (such as interferences between moving parts) to be coded as a configuration-space map. The entire mechanism can then be simulated and its behaviour examined.

Keywords: Configuration Space Maps, Constraint Resolution, CSG, Mechanisms, Geometric Modelling, SVLIS.

1. Introduction

This paper is about using a multidimensional CSG [10] modeller to build representations of configuration-space maps [12] and of systems of constraints. These can then be combined to give real-time kinematic predictions about the behaviour of machines, these predictions being completely derived from the machines' geometry.

A key fact about the CSG (or set-theoretic) representation underlies everything described here. It is that the complexity of a CSG tree does not change with the number of dimensions of the space containing the geometric model that it represents. The model's primitives at the leaves of the tree will (in general) become more complicated as the number of dimensions rises, but the tree itself is dimension-independent.

This fact has allowed us to write a multidimensional geometric modeller that is based on, but is also independent of, our conventional three-dimensional CSG modeller SvLis. This multidimensional modeller is called SvLis-m [4].

SvLis-m, which is written in C++, currently allows models to be generated in up to 32 dimensions*and we routinely work with models of between 8 and 12 dimensions. This would not be possible in a B-rep-type modeller that needed to keep track of the topological relationships between the entities that it was representing, as combinatorial problems would quickly become intractable as the number of dimensions increased.

2. SvLis-m and Configuration-space Maps

A single rigid object in space has six degrees of freedom—three translations and three rotations. By specifying six mutually independent values for these we can uniquely position and orient the object anywhere in space. A configuration space [12] is a space of degrees of freedom, with a dimension for each degree. Our single object would therefore have a six-dimensional configuration space.

Suppose we now have two objects in real-world space, one of which can move as before and which we shall consequently call a *nomad*, and the other of which is fixed and which we shall call the *obstacle*. The configuration space for this system is still six dimensional as only the nomad is free. But the nomad is not completely free—clearly it can only go to places where it is not obstructed by the obstacle.

We can now define a *configuration-space map*[7]: it is a division of the configuration space into two types of region—one in which every point represents a place and orientation where the nomad can go (called the *safe* region), and the complement of that, which is the region where the nomad interpenetrates the obstacle (called the *prohibited* region). Each region may be multiply connected and may be non-manifold, but, given the shapes of the two objects, the two regions are uniquely and completely defined. It is, of course, possible to add more nomads. This increases the dimensionality of the map by the number of degrees of freedom added.

However, it is one thing to define a configuration space map, and quite another to construct it. Making configuration space maps is a very hard problem which has received considerable attention. For reviews the reader is referred to the book by Latombe [5] or our own paper [12]. But making such a map is well worth the effort for many problems, because it completely encapsulates all the kinematic behavior of a group of rigid objects moving relative to one another. In particular, the problem

*Increasing this would just involve the rewriting of a small bit-manipulation class.

of path planning[†] [12] reduces to finding a track for a single point through the configuration space map that lies entirely in a safe region of the map.

A number of new and related algorithms for constructing configuration space maps have been devised by one of us and are described in detail in a recent thesis [13]. These algorithm descriptions fill some 90 pages, and so the two main ones are but summarized here; the thesis is available online.

- We have a general algorithm for non-polyhedral objects, but this is extremely memory-hungry as it relies on a recursive division of an *omnimodel* of the problem. An omnimodel is a combination of the problem's configuration space with the three[‡] real-world dimensions to give a model with a number of dimensions equal to three plus the number of degrees of freedom. The omnimodel effectively represents the nomad(s) as they sweep through all possible positions and orientations in space as a single CSG expression, along with the static obstacles. The configurations space map is a projection—parallel to the three real-world dimensions of the omnimodel—of the set-theoretic intersection of the obstacles and the swept nomads. This projection is done by recursively dividing the omnimodel into a bin-tree of boxes, and then projecting occupied boxes into the hyperplane defined by the configuration-space dimensions.
- We have a much more efficient algorithm for constructing configuration-space maps for polyhedral nomads moving among polyhedral obstacles. This works analytically, and depends on the facts that Minkowski difference is the same thing as a configuration-space map without rotations, and that the Minkowski difference of two unions of intersections is the union of the Minkowski differences of the intersections. In the case of polyhedra all intersections are obviously convex, and computing the Minkowski differences of two convex polyhedra is straightforward. To distribute unions over intersections in a CSG expression is to take the disjunctive form[§] We have extended the idea of Minkowski difference to include rotations, and all this allows us to create an analytical CSG expression for the configuration-space map of polyhedral nomads moving among polyhedral obstacles. Being a single expression, this is much more compact than the recursive hyperbox division required for our method for non-polyhedral objects.

Both these algorithms have been implemented in svLis-m.

To continue with polytopes, consider the simple two-dimensional situation shown in Figure 1. Here a square nomad is free to go everywhere but where it overlaps the

[†]That is, finding a path between the obstacles for the nomad or nomads from starting configurations to desired destinations; we return to this below.

[‡]Or sometimes two; see Figures 1 and 2.

[§]In the worst case, finding the disjunctive form is an exponentially-hard problem. But in practice, because of the way that it is natural for people to define objects by building them from unions of simpler shapes, most real models can be put in disjunctive form without too much computational effort.

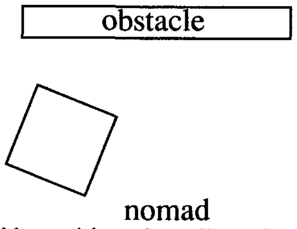

Fig. 1. A two-dimensional problem with a three-dimensional configuration-space map. The small square nomad is free to translate and to rotate everywhere but where it overlaps the fixed rectangular obstacle.

rectangular obstacle. The nomad has three degrees of freedom—two translations and a rotation—and so has a three-dimensional configuration space. Figure 2 shows the prohibited region of the configuration-space map computed by the analytical technique outlined above.

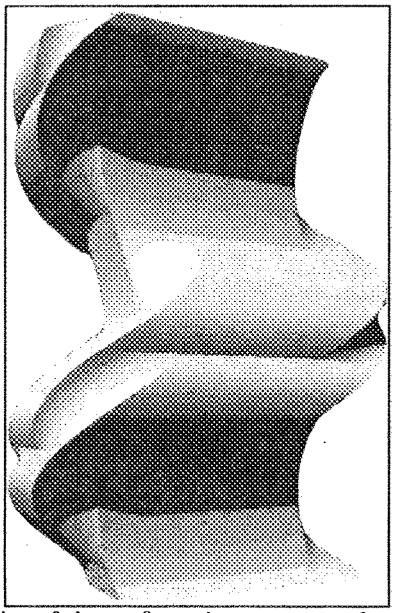

Fig. 2. The prohibited region of the configuration-space map for the problem in Figure 1 ray-traced by SVLIS. The translation dimensions are the horizontal plane, and the rotation dimension is the vertical axis.

Of course it is not possible to render configuration-space maps that are more than three dimensional, so to check them we use an animated display of the nomad(s) and the obstacle. In this the nomads can be moved by the computer's mouse or a 6 degrees-of-freedom input device such as a Spacemouse [6]. As the nomads move, the point representing their positions and orientations in their configuration space is continually membership-tested against the configuration-space map. When a movement would result in the configuration point straying into the ***prohibited*** region the movement may be prevented, or it may be permitted but with a visible flag being set in the graphics window to indicate the transgression.

Figure 3 shows two configurations of a nomad (the brown union of two tetrahedra) either side of a simple obstacle (the blue cube). If the configuration point

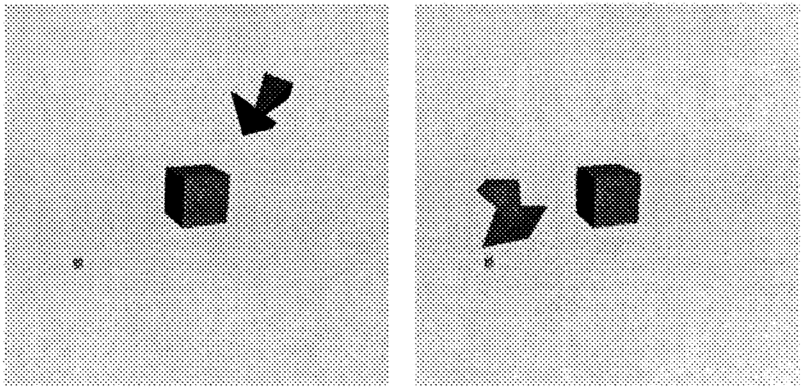
Fig. 3. Two configurations of a nomad (the brown union of two tetrahedra) either side of a simple obstacle (the blue cube).

is moved along the straight line in the configuration space that joins these two configurations the nomad enters the prohibited region, as shown in figure 4.

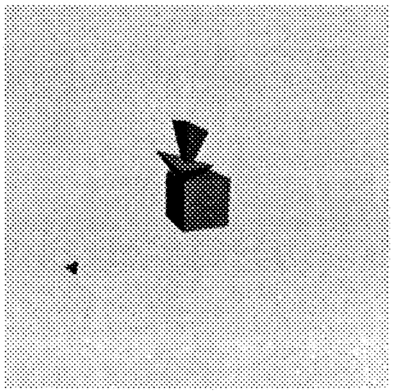
Fig. 4. A configuration about half way between those in Figure 3 where the configuration point has entered the prohibited region. The green box in Figure 3 is the flag to indicate that those configurations are safe. Here the flag has become a red pyramid to indicate that the configuration point is in the prohibited region.

In addition to membership testing configuration points against the configuration space map, we can also perform ray-tracing using the interval algorithm described in another of our papers [2]. Ray-tracing in a configuration-space map allows any possible collisions along a straight-line¶to be detected. The ray between the two configuration points in Figure 3 enters and leaves the prohibited configuration-space region at the configurations shown in Figure 5.

¶That is, a straight line in the configuration space, not in the real world, of course.

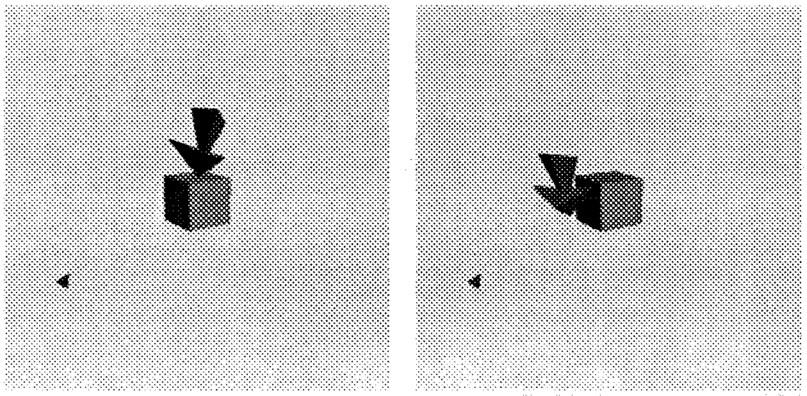
Fig. 5. The two configurations of the nomad from Figure 3 where a ray between those extreme configurations just enters and just leaves the prohibited region of the configuration space.

Given the configuration space ray-tracer, it is possible to implement a method due to one of us [9] that finds a path between two points in a CSG model that avoids obstacles; it also functions as a finder of connected components. The way this works is to cast a ray between the points. If this hits nothing, then the ray is the path. If it hits something, the mid point of the solid(s) [i.e. of the prohibited region(s)] along the ray is found, and a new point is generated in the hyperplane that perpendicularly bisects the ray at that point. This point must be in the safe region, and is found either by recursive division of that hyperplane or (and in practice this is often more efficient) by a Monte Carlo technique. The ray is then broken into two rays, one from the start point to the new point, the other from there to the end, making a polyline with two segments.

This breaking process is carried out recursively on the pairs of segments generated until a path is found, or the depth of the recursion is such that it is assumed that the polyline being generated is trapped.

This is a simple method, and it is not guaranteed to find a path should one exist. But it does run quickly (for the problem in Figure 3 a single path takes about two minutes on a 400MHz Pentium running Linux) so it is possible to run the search many times and look for an optimum (that is a minimum path length) among the resulting paths. Figure 6 shows the point on such a free path found by this method. It corresponds roughly with the nearest point on the free path to the blocked configuration shown in Figure 4. Of course, the shortest such path may not be desirable from an engineering standpoint: in general it will bring the moving nomads as close as possible without contact to the obstacles, so, for example, any inaccuracies in manufacture may cause glancing collisions.

The polyline generator is a very simple path planner, and is not intended to be either optimal, or even particularly good, though its speed does allow good paths to be found by multiple tries. But it can form the basis for the implementation of more sophisticated methods based on such things as potential fields and medial surfaces [11] (related, in this context, to Canny's road-maps, q.v. [3]) which we

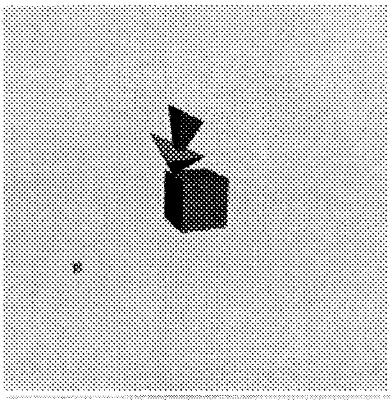

Fig. 6. A configuration about half way between those in Figure 3, but on the free polyline path generated by the method described in the text.

shall be investigating. The advantage of the ray-tracer is that it guarantees that any path found really does not cause the nomad to hit anything; it does not suffer from the possibility of failing to find collisions, as might, for example, a discreet and incremental method that took a series of isolated configuration points and membership tested them.

The ray-tracer also allows other problems to be solved. Figure 7 shows a spanner tightening a nut amongst some obstacles on a flat plate. The spanner-nut combination has three degrees of freedom—the centre of the nut can be placed anywhere in the plate and the two can rotate about the axis of the nut. Suppose, as designers, we need to know where the nut may be placed on the plate such that the spanner

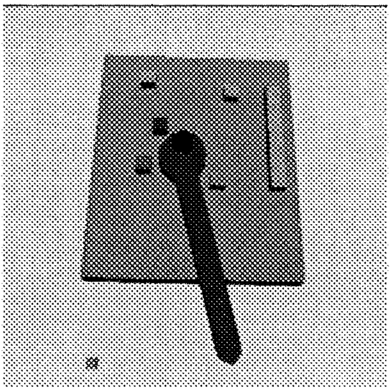

Fig. 7. A spanner tightening a nut amongst obstacles.

has enough rotational freedom to tighten it. If we turn the spanner over each time the nut is levered round then the spanner needs to be able to rotate through at least 30^o. If we don't turn the spanner over, then we need at least 60^o.

We computed the configuration-space map of the spanner-nut combination and the obstacles. Then, for a square matrix of 200×200 centre points for the nut on the plate, we cast a ray in the third dimension of the problem—the angle through which the spanner turns. For each ray we computed the longest continuous interval (representing an angle, of course) that was in the free region of the configuration space. Figure 8 shows the results. The black regions of the map are locations

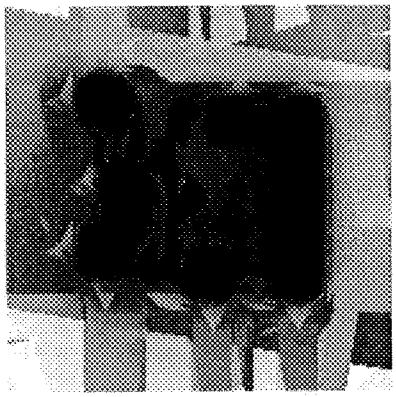

Fig. 8. A map of the problem in Figure 7 in the plane of the plate. The brighter the colour, the bigger the angle through which the spanner can turn.

where the nut cannot be placed at all. In the red region the spanner could only rotate through less than 30^o, so the nut could be placed here, but not tightened. In the blue region the rotational freedom is between 30^o and 60^o so the nut could be tightened here, but only by turning the spanner over. In the grey-to-white region, the spanner angle is bigger than 60^o, so the nut could be tightened anywhere here without turning the spanner over.

3. SvLis-m and Constraints

In this context, a constraint is a piece of geometric information that restricts degrees of freedom of movement; for example, a point is free to move anywhere in a plane; but if it is constrained to be a constant distance from a fixed point, it may then only move in a circle. Here we are concerned primarily with constraining the geometrical relationships between rigid solids that form the parts of a machine, but the methods described below could just as easily be used to define and constrain the shape of an individual object. Because a geometric constraint can always be represented as an algebraic and trigonometric expression, it is possible to treat that expression as a piece of geometry in svLis-m. As svLis-m is a CSG modeller, such constraints can easily be intersected together to define the region where they are all satisfied.

To do this, first solid models of the component parts of a machine to be constrained are read in from the conventional three-dimensional modeller svLis. Before constraints can be imposed on them, points of interest on each part are defined. These are known as *constraint points.* The constraint point has associated with it svLis-m sets that represent the possible x, y and z positions of the point. These sets are written in terms of the degrees of freedom of the part. As a two-dimensional example, consider a crankshaft, A, and a con-rod, B — parts of a reciprocating piston engine (Figure 9).

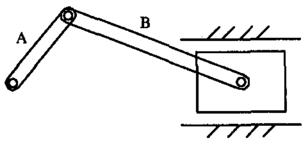

Fig. 9. Piston with a small amount of slap.

Ignoring the piston for the moment, this will have six degrees of freedom—two translations (x_A, y_A) and a rotation (θ_A) for the crankshaft and a corresponding (x_B, y_B, θ_B) for the con-rod. The process for identifying a constraint point on an individual component, A, on Figure 10 goes like this:

Express A1 in terms of the degrees of freedom of part A:

```
ConstraintPoint A1 = A.IdentifyPoint(XA1, YA1)
```

where the function `IdentifyPoint` *assigns:*

$$x_{A_1} \Leftarrow x_A + r_{A_1}cos(\theta_A) - r_{A_1}sin(\theta_A) \tag{1}$$

and

$$y_{A_1} \Leftarrow y_A + r_{A_1}cos(\theta_A) + r_{A_1}sin(\theta_A) \tag{2}$$

(where r_{A_1} is the distance of A_1 from an arbitrary origin defined on part A.)

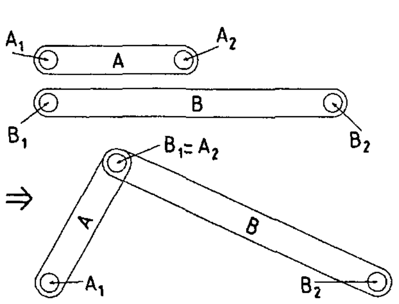

Fig. 10. Components of the piston example.

Fixing this part to pivot about A_1 at a particular point P $= (x_p, y_p)$, involves intersecting the constraint model with the x_{A_1} and y_{A_1} sets of constraint point A_1 equal to the (x_p, y_p) values of P. Since SVLIS-M uses closed implicit inequalities ('solid' is everywhere that an inequality is negative or zero), the absolute value function can be used to represent equality (taking the *abs* of an inequality forces it to be positive everywhere but where it is zero—it turns a solid into a sheet representing its surface). Initially (before any constraints are imposed), the constraint model, **C** is the universal set. The constraint model, now becomes:

$$\mathbf{C} \Leftarrow \mathbf{C} \ \cap \ |x_{A_1} - x_P| \ \cap \ |y_{A_1} - y_P| \tag{3}$$

Constraint points B_1 and B_2 can be identified on part B (the con-rod) in the same way as above. Fixing point B_1 to coincide with point A_2 on the crankshaft A corresponds to intersecting the constraint model as below:

$$\mathbf{C} \Leftarrow \mathbf{C} \ \cap \ |x_{A_2} - x_{B_1}| \ \cap \ |y_{A_2} - y_{B_1}| \tag{4}$$

Continuing with the piston example, the system can represent constraint points that are allowed to move between certain bounds, i.e. inequality constraints. This is not easy for some other systems. Point B_2 may be constrained to have a limited movement in the y direction to simulate a piston in a cylinder that has a certain amount of slap (for this example we will ignore the fact that in reality this floppiness would also allow the piston to rotate a small amount). If the y value of point B_2 is required to be $-0.1 \leq y \leq 0.1$ (see Figure 9) then the intersected sets are:

$$\mathbf{C} \Leftarrow \mathbf{C} \ \cap \ (y_{B_2} - 0.1) \ \cap \ (-0.1 - y_{B_2}) \tag{5}$$

Note there are no absolute value operators around the sets because they are representing inequalities.

By intersecting all these constraints, we have now constructed a constraint model in the six original degrees of freedom of the simple piston example, the piston itself being allowed some lateral movement within the cylinder. Any point that membership-tests solid against that model will indicate a valid configuration of the mechanism.

It is now possible to use a simple Newton-Raphson procedure (possibly combined with a recursive division of the multidimensional volume containing the constraint representation) to find and to move along/inside the multidimensional line, sheet, or volume that represents the constrained mechanism defined as above.

Figure 11 shows a more complicated example. It is a series of stills taken from an animation of a Peaucellier straight-line motion modelled using our constraints system.[||]

4. Combining configuration-space maps and constraints

So far we have shown sVLIS-M acting in two separate ways to solve configuration-space problems and constraints problems. But the fact that both sorts of problems are described using the same system allows us easily to combine solutions to them, giving an even more powerful method of representing the kinematics of machines.

[||]The full animations of all the examples in this paper can be seen at the project website [4].

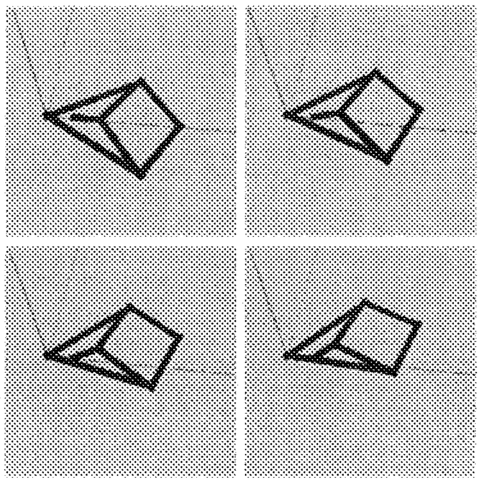

Fig. 11. The constraints system animating a Peaucellier motion. The red, green and blue lines are the coordinate axes.

If we wish to model the kinematics of a machine consisting of rigid moving parts, there are two sorts of information that we may have available. The first is the geometry of the parts—this will influence how they move relative to each other, as no two parts can be in the same place at once. The second is the types of connections between the parts—this too will influence how they move relative to each other, as, for example, a pin joint will allow different movements to a sliding joint. The first sort of information can be embodied in a configuration space map, and the second as a series of geometric constraints.

In principle just the configuration-space map of all the geometry would be enough to decide how the machine worked, but this would be a very inefficient way to solve the problem, especially as explicit constraint information about hinges, sliders, rollers and so on is almost always available too.

As the configuration and constraint approaches described in the second and third sections of this paper have both been implemented in the same multidimensional geometric modeller, we have been able to combine them into an integrated system that models the behaviour of machines.

First, geometric models of the component parts of the machine are created, then constraint points on them are defined and a constraints model is built. Then components that may interfere are modelled in a configuration space map. Finally, the constraints system animates the machine, checking each movement by ray-tracing in the configuration-space map for possible interference between the components, as in Figure 5.

Figure 12 shows a three-dimensional four-bar linkage; the two pink triangular prisms are free to rotate about the axes of their bases, and their apexes are joined by the long green rectangular link. The grey cube is an obstacle to movement. The constraints system represented by the linkage and the configuration-space map of

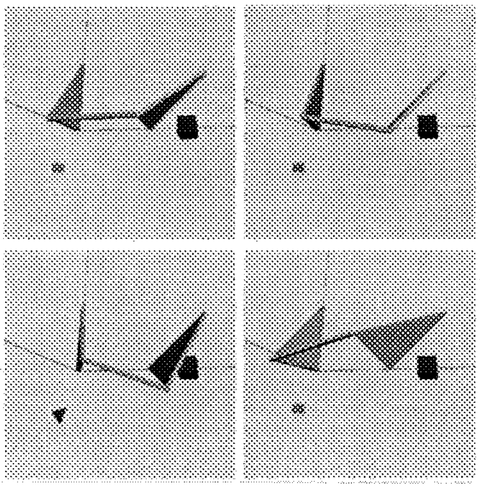

Fig. 12. A three-dimensional four-bar linkage with an obstacle modelled and animated using our constraint system and our configuration-space map-maker. As before, the red green and blue lines are the coordinate axes and the small green cube and red tetrahedron are not part of the problem but are simply a collision flag.

the linkage and the grey cube were computed. The constraints system was then used to animate the linkage, following its path in configuration space. This path ended when the right-hand pink triangle struck the grey cube, as at that point the configuration-space map turned from a safe region to a prohibited region. The system was programmed to bounce—reversing its direction of motion whenever an obstruction was found—and so the four-bar linkage then moved in the opposite direction.

5. Conclusions

The principal limitation to the system described in this paper is that the configuration space map-maker can, at the moment, only realistically deal with polyhedra, as other shapes use up too much memory. Having said that, we have produced complete polyhedral configuration-space maps in twelve dimensions [13], which is much more than any other system that we have found in the literature [12]. We have also started work on path planners for configuration-space maps. In addition, the spanner problem gives an example of a non-path-planning application for these maps.

The combination of a configuration-space map for parts of a machine with a constraints model describing the machine's workings has been made easier by the implementation of both in the same multidimensional geometric modeller. The constraint representation is, effectively, a complicated but low-volume region of the configuration space. An obvious next step is to try to exploit this by intersecting the constraint representation with the configuration-space map to reduce the complexity of the latter and to create a single representation of an entire machine. We intend to follow that idea up.

6. Acknowledgements

The authors would like to thank the UK's Engineering and Physical Sciences Research Council, Information Geometers Ltd, and Southco Ltd for supporting this work. They would also like to thank John Woodwark, whose original idea [8] led to the development of these techniques.

The authors can be contacted at `g_mod@bath.ac.uk`.

1. A. Bowyer, SVLIS: set-theoretic kernel modeller, (Information Geometers Ltd, 1995) and http://www.bath.ac.uk/~ensab/G_mod/Svlis/
2. A. Bowyer, J. Berchtold, D. Eisenthal, I. Voiculescu, and K. Wise: Interval Methods in Geometric Modeling, Proc. GMP2000, Hong Kong, April 2000, IEEE Publications, ISBN 0-7695-0562-7.
3. J. F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, Mass., 1988.
4. D. Eisenthal, K Wise and A. Bowyer, SVLIS-M - a multidimensional CSG modeller, http://www.bath.ac.uk/~ensab/G_mod/Svm/
5. J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publ., 1993.
6. LogiCAD GmbH. Spacemouse homepage at http://www.spacemouse.com/
7. T. Lozano-Pérez. Spatial planning: a configuration space approach. *IEEE Transactions on Computers*, 32(2):108–119, 1983.
8. S. J. Parry-Barwick and A. Bowyer: Multidimensional set-theoretic feature recognition. CAD Journal 27(1995)731.
9. D. Pidcock: Convex hull generation, connected component labelling and minimum distance calculation for set-theoretically defined solid models, University of Bath PhD Thesis, 144pp, 2000.
10. A. Requicha: Representations for rigid solids: theory, methods, and systems, *Computing Surveys* 12,4 (437-464), December 1980.
11. D. J. Sheehy, C. G. Armstrong & D. J. Robinson: Shape Description by Medial Surface Construction, IEEE Trans. Visualization & Computer Graphics, 2, pp 62-72. 1996.
12. K. Wise and A. Bowyer, A survey of global configuration-space mapping techniques for a single robot in a static environment, International Journal of Robotics Research, Vol 19, No 8, August 2000, pp 762-779.
13. K. Wise: Computing global configuration-space maps using multidimensional set-theoretic modelling. University of Bath PhD Thesis, 232pp, 1999. Available online from the project website [4].