Better and faster pictures from solid models

by John Woodwark
IBM UK Scientific Centre
and Adrian Bowyer
University of Bath

This paper describes a series of programs for producing pictures from, and performing other calculations on, set-theoretic solid models. The programs all use a technique of recursive division of the space in which the model resides. This ensures that their performance is *better than linear* against the complexity of the objects which they are being used to model.

Introduction

The rationale behind the development of solid modelling [1] techniques is their potential to produce complete and accurate representations for engineering analysis and, ultimately, synthesis. However, it is often admitted that most of the work on solid modelling so far has gone into the production of graphics. This is logical, for a picture of a model is usually necessary before one can be confident in attempting any further processing.

Despite the effort put into graphics, it is still generally thought that producing pictures of solid models is a time-consuming process, and that this is especially true when more sophisticated and realistic pictures are required. Many advanced methods of picture production and enhancement have been developed in pure computer graphics work, but these are mostly based on simple face models, and the techniques are seldom applied to solid models.

Why is this? One reason is that many engineers are clearly reluctant to depart from traditional standards, meaning monochrome line drawings. However, one of the prime purposes of traditional drawing practice is to overcome the restrictions of drawing reproduction techniques; but these historical limitations are currently disappearing, or becoming less of a problem as more on-line facilities become available. Electronics engineers have long since come not merely to approve of, but to rely on, coloured pictures as their technology has become more complicated, and one does not imagine that mechanical engineers are any less flexible. A more substantial reason that is often given for not using more sophisticated graphics is that realistic pictures take much longer to produce. Indeed solid modelling itself has been the subject of unfavourable comparisons with less advanced drafting technology on this very point, whatever sorts of pictures are being computed.

Why does it take so long to make pictures from solid models? In essence, time is spent overcoming complexity which comes from two sources. The first is inherent in the *geometry* of the shape elements used: the necessity to evaluate algebraic expressions in order to do anything useful with the model. The second comes from the *number* of elements used, and is time spent searching through geometric data. The latter is much more serious because it

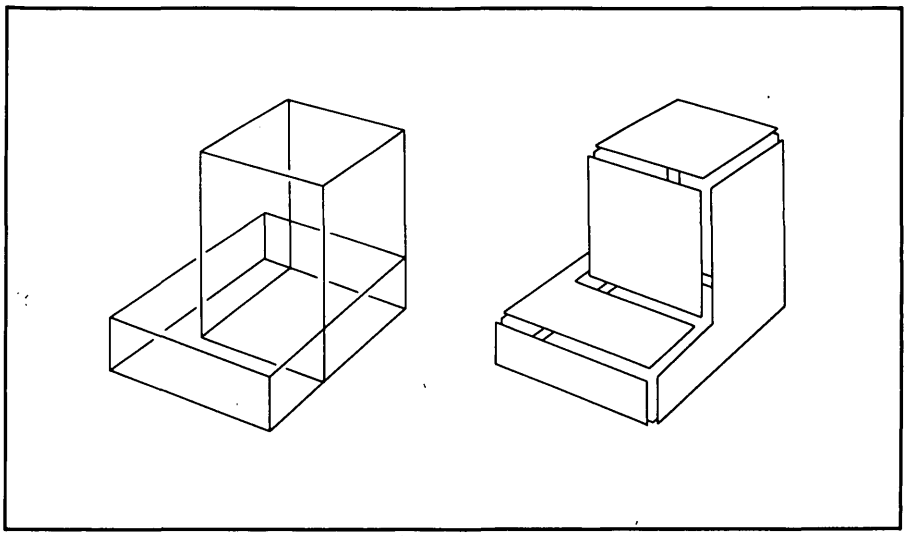


Fig. 1 Set-theoretic, face and boundary representation models
The left-hand diagram shows a set-theoretic model made of the *set union* of two rectangular blocks. That on the right-hand side shows the same object defined by a list of its faces, which may have a constraining logical structure imposed upon them to ensure the solidity of the object defined, thus making a B-rep model

The SID model building language

```
; Build the bolt's shaft
  z_point :=pt(0,0,1)
  axis :=ln(z_point,z_point)                ; ln returns a line through a point
                                            ; in a given direction
  radius :=diam*0.5
  blt_ shaft :=cylinder(axis, radius,n_ facets)
; Now the hex head
  root_ 3 :=sqrt(3.0)
  radius :=diam*0.5*root_ 3
  face_ point :=pt(0,radius,0)
  face :=space(face_ point,face_ point)     ; space returns a planar
                                            ; face through a point
                                            ; with a given surface
                                            ; normal
  head :=face
  FOR count :=1 TO 5 DO
  { angle :=count*3.1415926/3
    head :=head & spin(face,axis,angle)     ; & means intersection
  }
```

Part of the SID code that defined the hex-headed bolt that holds down the top of the air filter in Fig. 6.

affects the way in which a system performs as the complexity of the model with which it is concerned increases. Many current commercial systems exhibit a markedly worse than linear performance as models become more complicated; computation times for picture generation (and other things) increase faster than model complexity. This is the real reason stopping many techniques which produce very elegant pictures of teapots from being applied to models of entire gearboxes or car bodies.

This paper is an introduction to a suite of solid modelling programs which have been developed using algorithms especially designed to minimise the effect of complexity on performance (a number of programs which will be mentioned are the subject of more detailed forthcoming publications, and one has already been described elsewhere [2]). All the picture production techniques described in this paper work in *better than linear* time against model complexity.

Now, almost all commercial solid modelling systems use what is called a *boundary representation* (B-rep), which represents the *surface* of the object being modelled. However, the techniques mentioned in this paper use a *set-theoretic* — sometimes called Boolean or constructive solid geometry (CSG) — representation exclusively (Fig. 1). That is to say that objects and parts of objects are represented as a set-theoretic algebraic expression. The operands in this expression are elementary solids, and these are linked by operators taken from set theory [3]. Many systems use set operations as input techniques, but nevertheless build a boundary model.

The advantages of true set-theoretic modellers are that they make many processes easy to program, and that they are numerically stable. However, because there is no *localisation* of the effect of any of the primitive elements of which set-theoretic models are composed, the performance of algorithms deteriorates quickly with complicated models, and that is why set-theoretic models, as they stand, are little used.

This difficulty can be overcome by a technique which is the major feature of the programs reported in this paper: segmentation of the region of space occupied by the object being modelled. This *object space* is recursively divided up into a number of smaller volumes, or sub-spaces. In the case of all the programs mentioned in this paper, these are cuboids. During this division process the set-theoretic model itself is *pruned* [4] (Fig. 2), which removes the parts of the algebraic expression describing the model which are not relevant to a given sub-space. The result is a large number of simple set-theoretic models, each valid only in its own well defined volume. These sub-models are simple to process because of the numerical robustness of the set-theoretic model, which was mentioned above. This robustness is not compromised at all by the division process.

There is one limitation common to all the programs described in this paper. That is that all the models created are *facetted.* Curved surfaces must be approximated by a combination of planes. The systems have only one primitive: the infinite planar half-space. This restriction has been accepted in order to produce a broad spectrum of interesting and fast running software. However, as will be mentioned in the conclusions, this is not a limitation on the basis of the authors' techniques for the control of complexity, and they have extended the techniques described in this paper to models of components with complicated curved surfaces.

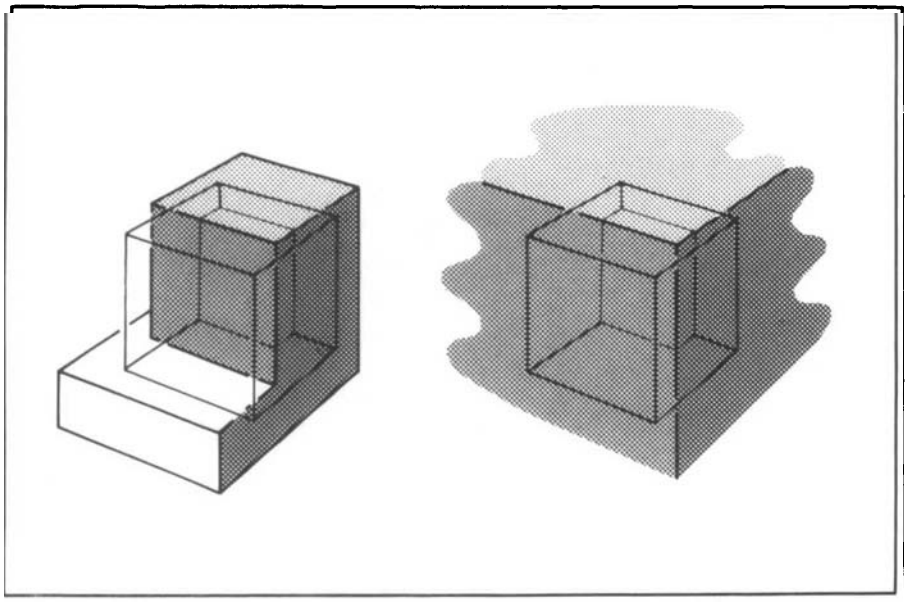


Fig. 2 Dividing the space in which the model resides
The left-hand diagram shows the blocks model from Fig. 1 being considered inside the cube indicated. The right-hand diagram shows the *pruned* model that is needed inside the cube; just three faces

Input

A common criticism of solid modelling systems is that construction of solid models is very time consuming, and

that the ideas needed for their construction are alien to many potential system users. While the set-theoretic concepts commonly used to construct solid models (whether set-theoretic or boundary in their internal representations) may be less natural than the better computer-aided drafting systems, they are a great improvement on the *ad hoc* techniques commonly used to construct face models for pure computer graphics work.

The authors are well aware of the desireability of graphical input to solid models [5], but the techniques presently available are limited, and a general facility can only be obtained from text input. This has the further advantage in software intended to have a good degree of standardisation and portability that it is not necessary to accommodate various input devices and protocols. All that is needed is a text editor.

The language that the authors have designed aims to compensate for the unnaturalness of textual input by allowing the greatest exploitation of the flexibility of a language form. The language (SID) is structured and Algol-like. SID shares the fundamental purpose of all solid model input languages [6] by allowing the user to construct part-models and to combine them using the set-theoretic operators.

However, SID provides two important groups of additional facilities. The first is a range of auxiliary structures and operations which allow the geometry of a model to be written in the language as a fairly direct translation of the geometric steps in its construction. For instance, a scalar product between two vectors can be written as such, and the resulting numerical value used, say, to set up a distance between two points. There is no need for the user to do such calculations himself, or for him to write out their algebra in full.

The second group of facilities consists of control structures which simplify repetition of parts of a model, either directly or with parameterisation. Loops and conditionals are available within program modules, and additional modules may be called with parameter values (which can be geometrical entities as well as scalars). In addition to assisting in the programming of particular models this allows the user to construct libraries of parameterised components such as, say, nuts and bolts, which can easily be instantiated in different sizes. The component models can be structured so that, in this example, the correct relationships between head size, thread length and diameter would be automatically maintained, with the nut or bolt being specified by its diameter only.

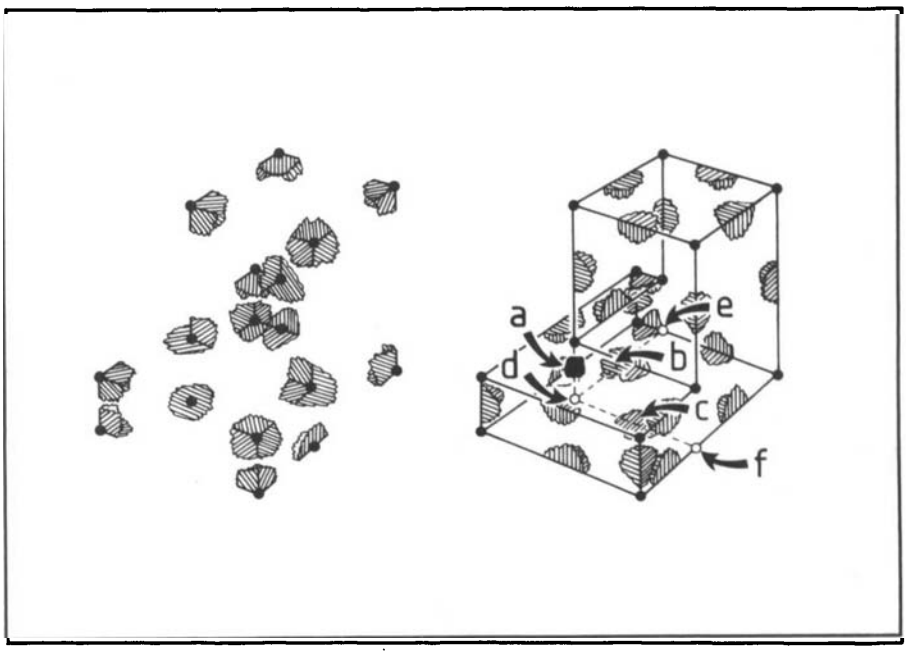


Fig. 3 Finding the edges of a model
The model is divided to find vertices (left-hand diagram). The edges between these are then found. The right-hand diagram shows an un-needed edge between a needed vertex and an un-needed one (*a*), un-needed edges between two un-needed vertices (*b* and *c*), and un-needed vertices on needed edges (*e* and *f*). MEG eliminates all these un-needed features, producing a simplest possible edge description

After a model has been defined it is compiled into a single set-theoretic expression relating a number of planar half-spaces. This is then available in a common form for any of the programs described in the rest of this paper.

Picture generation

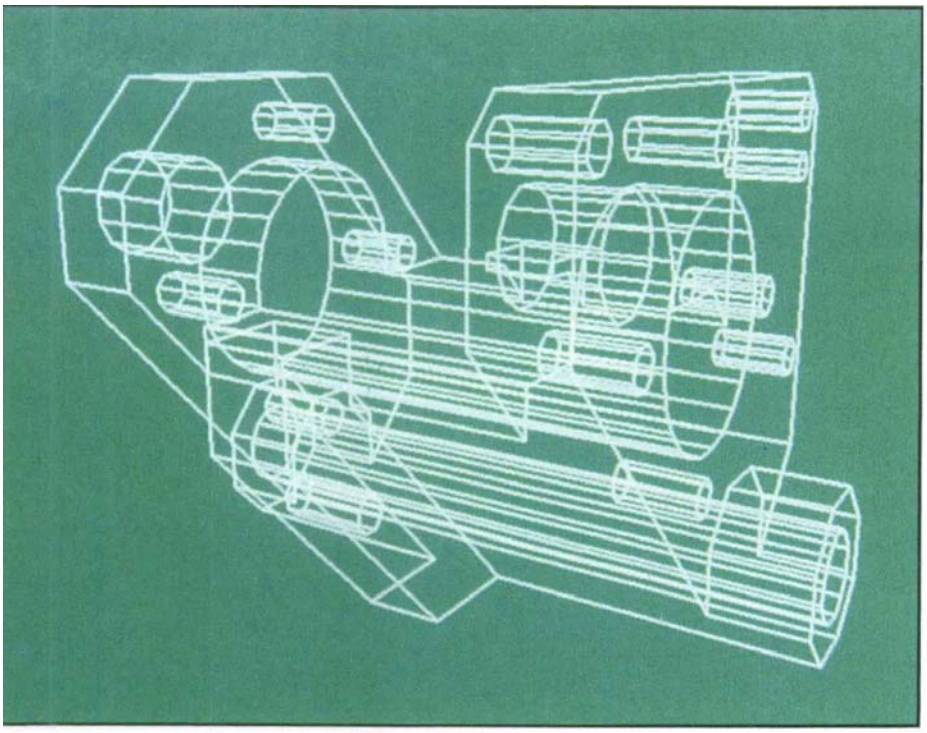
The spatial segmentation techniques outlined in the introduction have been applied to three different programs which produce pictures. They provide a spectrum of increasing realism at increasing cost (although this does not correspond to the chronological order in which they were written). The first two programs use the technique of spatial segmentation as a *strategy* to obtain a picture of the model. The segmentation process creates sub-volumes and their associated sub-models on the fly, processes them, and then discards them.

The first program (MEG) uses this technique to generate a wire-frame representation of the edges of the object. MEG can provide a quick view to verify a simple shape, and (by changing the viewing parameters and re-drawing the wire frame in perspective) it allows the user to assess various viewpoints quickly before committing himself to a more time-consuming process.

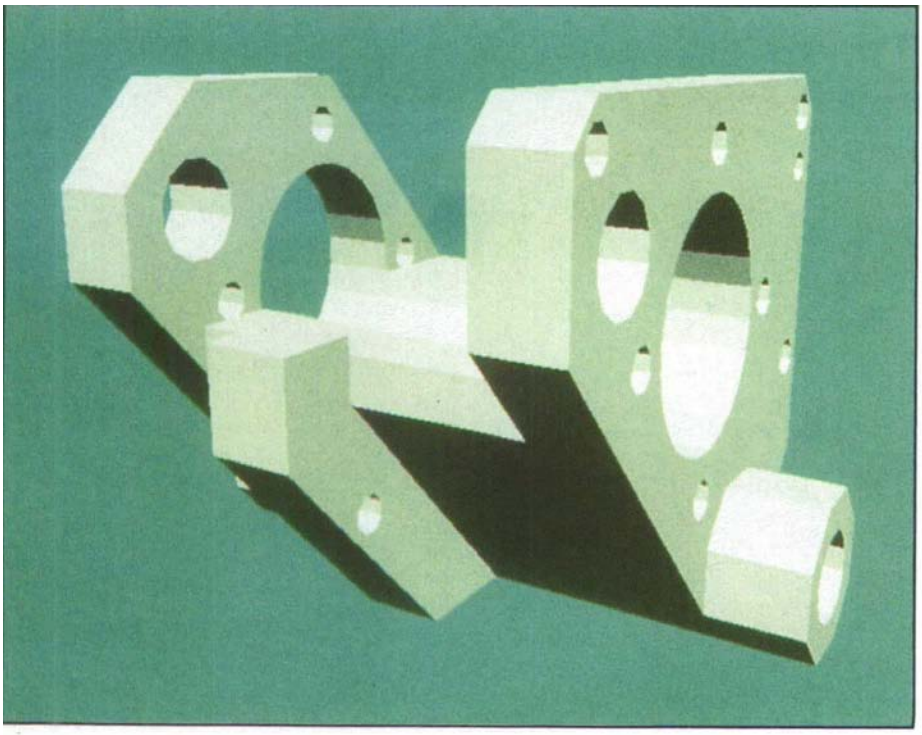
MEG starts by dividing up the object space. As in all these programs a point of particular concern is control of the division so that it neither generates excessively complicated sub-models, nor segments the model so much that the processing as a whole is slower because of the sheer number of sub-models to be processed. In MEG, division is controlled by a geometric analysis of the contents of a given sub-model. We know that further division of a sub-

The software

DORA	Divided Object Ray-casting Algorithm
SID	Set-theoretic Input to DORA
MEG	Model Edge Generator
VOLE	VOLume Evaluator
SAM	Surface Area and Mass
FIDO	Fractals In DORA



a



b

Fig. 4 Pictures generated by four of the programs
a Wire frame generated by MEG (this took about ten minutes on the authors' VAX 730; subsequent views of the wire frame would only take a few seconds)
b Shaded but not shadowed picture generated by VOLE (this took about 20 minutes)

space is futile when the sub-model it contains consists of half-spaces which all pass through a point. The algorithm calculates the point nearest to all the half-spaces, and its average distance from them. The decision on whether to divide further is made by comparing this average distance with the size of the sub-space.

Constructing a wire frame from a solid model is not a difficult process. The method is one of generate-and-test. Potential, or tentative, edges are formed from the intersections of primitives, and then the real edges among these are identified by membership tests [7]. In the current context, this process could easily and quickly be applied to each sub-model as it was generated, and the first implementation of recursive division [4] was for this purpose.

However, this direct approach leads to a wire frame which, while correct, consists of many more than the smallest possible number of line segments. Since the aim of MEG is to facilitate repeated projection and display of the wire frame, unnecessary wires would make for inefficiency. It might be possible to re-assemble the wires, but this would require a lot of processing and, possibly, involve numerical problems. Instead, MEG does not generate the edges of the model in each sub-space that it produces; it processes the sub-spaces to identify tentative *vertices* of the model. Further, by applying the pruning technique already mentioned, each vertex is associated with a sub-model representing the set-theoretic relationship of the half-spaces that pass through it. When division is complete, the vertices are examined to find pairs of non-coplanar half-spaces occurring at more than one vertex.

In this way lists of collinear vertices are created. Between each pair of vertices on such a list there may tentatively be an edge. Further pruning and membership testing determines whether each such edge actually exists, and those that do are aggregated into the longest possible wires. At the same time the colours associated with the half-spaces which generate an edge are averaged to produce a notional edge colour. This may be used to colour the wire frame to make it easier for the user to distinguish different parts of the model.

The second program in the package (VOLE) produces continuous-tone colour pictures, with shading and hidden-surface removal. VOLE's method, which is reported in detail elsewhere [2], is to deform the set-theoretic model into an object space aligned with the screen co-ordinate system in such a way that a picture of the deformed object, seen in *parallel* projection, corresponds to a picture of the original object seen in the perspective projection required. As in MEG, division is used as a picture production strategy, and the depth of division is conveniently controlled by screen resolution. In VOLE the divided model is also discarded. VOLE is unusual in the small amount of memory that it requires. It provides a performance between that of MEG and that of DORA (described below) except for very large models, for which DORA is faster.

DORA is based on the technique of ray-casting. A picture is produced by considering every pixel on the display device, and generating a *ray* into the scene corresponding to a line from the eye position through that pixel (Fig. 5). The ray is followed until a surface of the object is struck, when the colour of that surface is painted into the pixel. This

technique has previously been used for producing pictures of solid models [8], but with only a somewhat worse-than-linear performance against complexity.

DORA starts as the other programs do by dividing the object space. In DORA's case, however, the divided structure is retained, and can be used to produce a number of different pictures. Because of this, the division process is designed to perform a near-optimal division, with the idea that the time needed can be recouped by increasing the speed of picture production over a number of pictures.

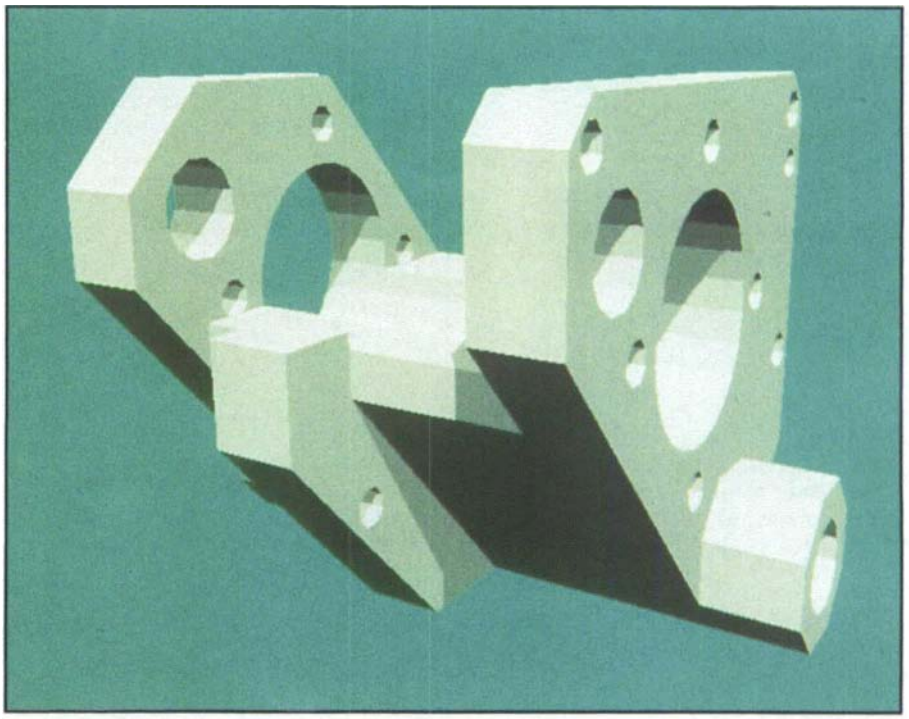
The division strategy is essentially one of previewing the results of the division of a given sub-space before actually performing it. This is not only applied to deciding whether to divide a given sub-space, but also to deciding the position of the plane splitting the sub-space into two smaller ones. When comparing alternative divisions, the program considers the likelihood of a sub-space's being struck by a ray, as well as the complexity of the sub-model that it contains. Constructing the segmented sub-model takes a time roughly proportional to model complexity, and the size of the segmented model, which is considerable, is also proportional to the size of the original model. The subsequent ray-casting process is, however, attractively insensitive to model complexity.

The ray-casting process in DORA (which is similar to parallel US work [9], although that does not use solid models) compares each ray with the division structure, and subsequently with the contents of the sub-spaces through which it passes. Both the traversal of the division and the comparison with the model are performed in the one-dimensional parameter space of the ray, and are consequently fast. Provided that the division has been performed reasonably efficiently, most of the path of an average ray will be through sub-spaces containing null sub-models (empty boxes, in other words). Only as the ray approaches the surface of the object must appreciable geometric work be done on parts of the model.

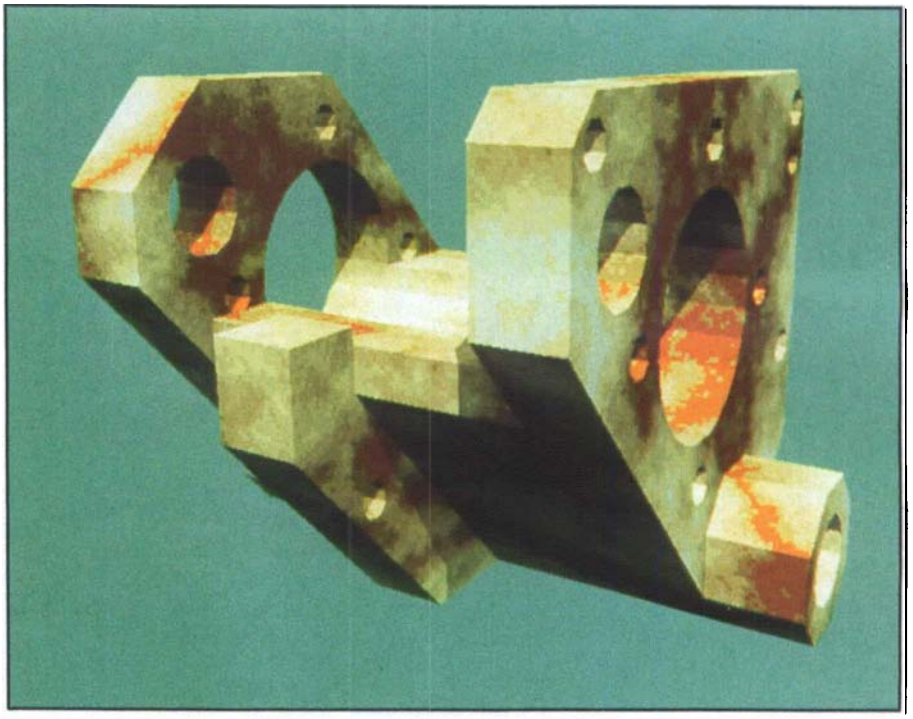
As models become more complicated, traversing the segmentation (which is organised as a tree) becomes only slightly more time consuming, and, since each ray only strikes the surface of the model once, there is little increase in computation time. As with VOLE, the performance of this algorithm is more dependent on the complexity of the view than on the complexity of the model.

Because (unlike VOLE) every pixel on the screen is always considered individually by DORA, it is a simple matter to model the effect of a number of light sources, with an increase only in the time spent on lighting intensity calculations. More sophisticated effects may be produced at a cost which is proportional to the number of rays needed.

The simplest effect is to generate a second ray from each point on the model struck by the viewing ray back to the light source. If this second ray strikes another part of the model, then the first point is in shadow; if not, it is directly illuminated. With appropriate lighting parameters, cast shadows can easily be simulated in this way, and this can be extended to shadows from multiple light sources. Additionally, reflective surfaces can be modelled by causing the viewing ray to bounce off mirror surfaces. Atmospheric effects, such as fog or mist, can also be simulated by attenuating colours propor-



c



d

Fig. 4

c Picture generated by DORA (it took about half an hour) with both shading and cast shadows

d Picture generated by FIDO, which was used to add a rusty texture (this took about 40 minutes)

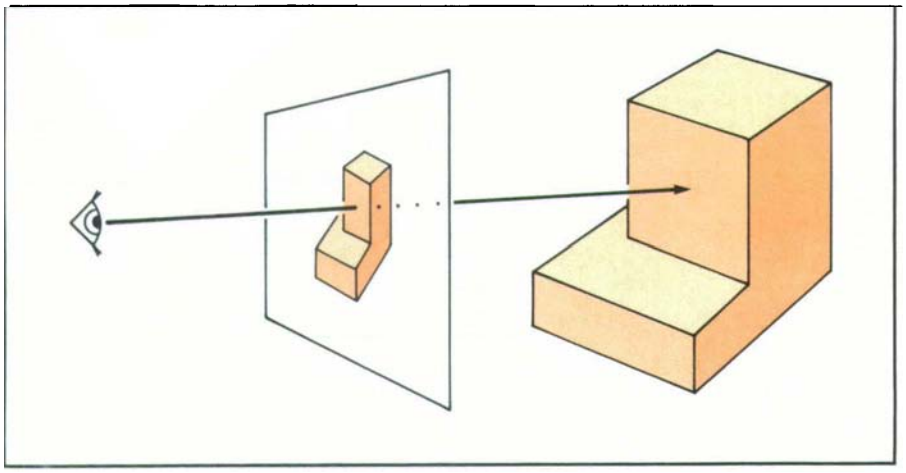
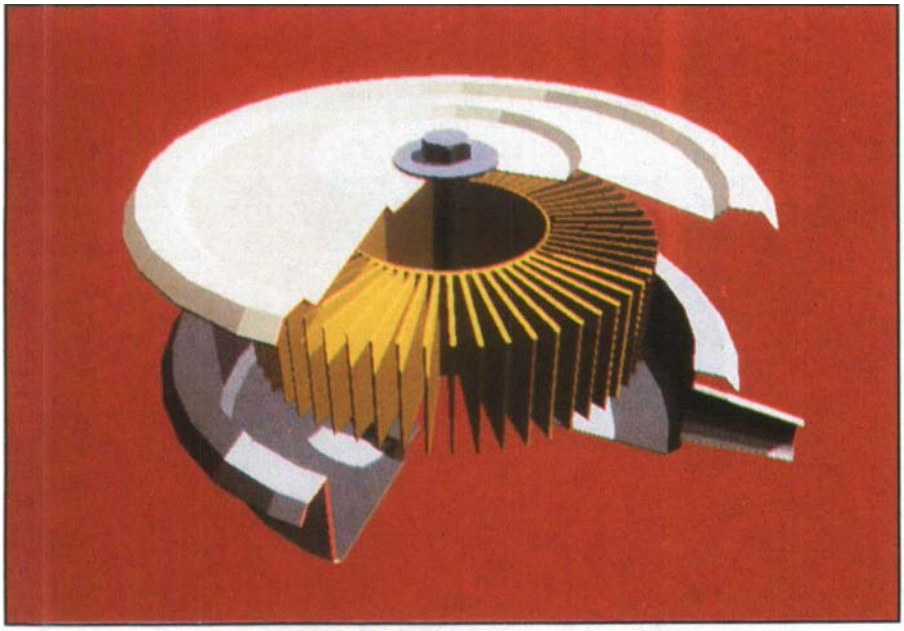
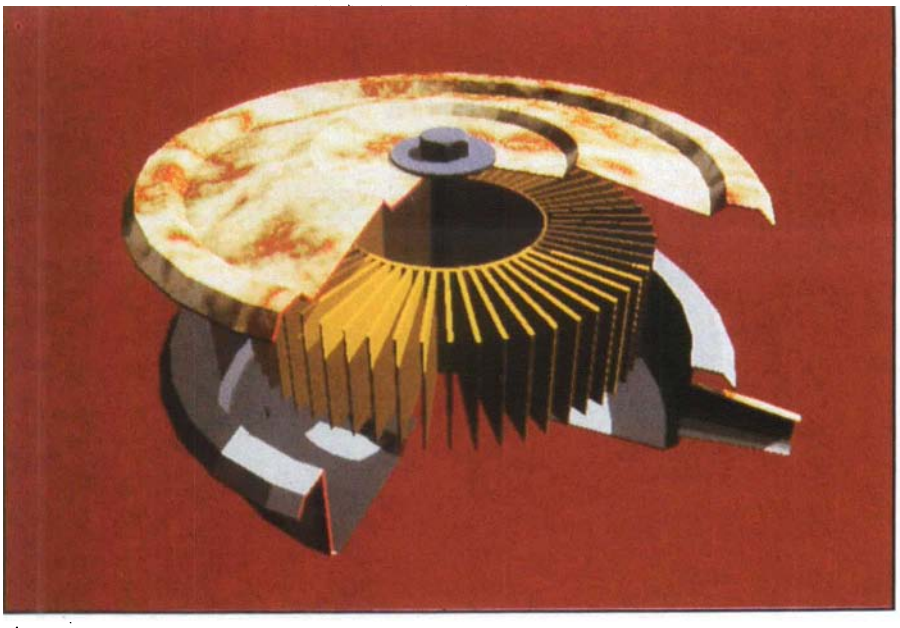


Fig. 5 Ray-casting to make a picture of a model
Through each pixel on the graphics screen a line from the eye is traced into the model. The first surface that it hits defines the colour that needs to be painted into the corresponding pixel



a



b

Fig. 6 A model of a car's air filter
a Model created by DORA
b Rust added to the model by FIDO
These pictures each took about 45 minutes to generate on the VAX 730. The fact that SID contains a loop instruction and parameterised objects made the filter very easy to model

tionally to the length of the viewing ray. Even more complicated effects, such as inter-reflections between objects, can be simulated (at some cost) by generating a fan of rays from points on the object. So far shadowing and reflections have been implemented in DORA, and experiments have been carried out with multiple light sources.

To a design engineer the synthesis of texture is probably a more interesting development than very complicated lighting. Recent texture simulation techniques have mainly relied on the mapping of two-dimensional texture patterns onto the faces of an object [10,11]. This is only suitable for topologically simple objects, because of the problems of joining the edges of the piece of texture correctly.

The authors have implemented a technique of solid texturing, based on three-dimensional fractals, which parallels the latest pure computer graphics work [12,13], although that is not fractal based. The pseudo-fractals, as they should properly be called, are defined within the entire object space, but only evaluated at the surface of the component.

This is achieved by another tree-structured division of space in a program called FIDO. The division is created dynamically as and when fractal values are required. Because of the coherence of the ray-casting process only a small part of the tree of fractal values needs to be generated to provide a value at a given surface point. So far the fractal values produced by FIDO have only been used to perturb colour values, which can yield a sand-blasted effect, or, more spectacularly (as in Fig. 6*b*), a rusty appearance. Other effects may be achieved by using the fractals to perturb the surface normal at each point.

Other processes

The main feature of the software discussed in this paper is its ability to produce high-quality graphics at reasonable cost, but it is important to be able to exploit the properties of solid models as far as possible, and the calculation of mass and area properties is a facility that is extremely useful, and not too difficult to implement.

In calculating these properties (which is done by a program called SAM) the division technique is again employed. This is similar to the approach taken by others [14], except that they do not use a pruning process. As SAM divides the object space, sub-spaces are classified as being outside the object, inside the object, or possibly containing some of the surface of the object. In the case of

sub-spaces of the last type, division continues to a given resolution which determines the accuracy of the analysis. Within each undecided sub-space a point is generated pseudo-randomly from the uniform distribution over the sub-space and a membership test is performed. If the point is inside the object, the volume of the sub-space is added to the object volume; if not, then the sub-space volume is not added. Other mass properties are obtained by appropriate summations of sub-space masses.

Area calculations are more of a problem with set-theoretic models, unlike boundary models where areas can be computed directly. Sarraga [15] suggests a segmentation in the parametric space of the faces, but the authors prefer to use the volume segmentation again, which has to be computed in any case, and they believe it to be more efficient.

Essentially, for each sub-space, each half-space is considered in turn. A bounded area of it is generated, by intersecting it with a convenient projection of the sub-space. Now, some of this area may lie outside the sub-space, but SAM does not need to know how much. SAM again generates a random point from the uniform distribution over the bounded area of the half-space, and tests whether it is inside *both* the sub-space *and* on the surface of the sub-model. If it is, the area of the bounded piece of half-space is added onto the appropriate totals (Fig. 7). It is, of course, necessary to be careful with the membership testing to ensure that areas of coincident half-spaces which all lie on the surface of the model do not cause an erroneous multiplication of the area.

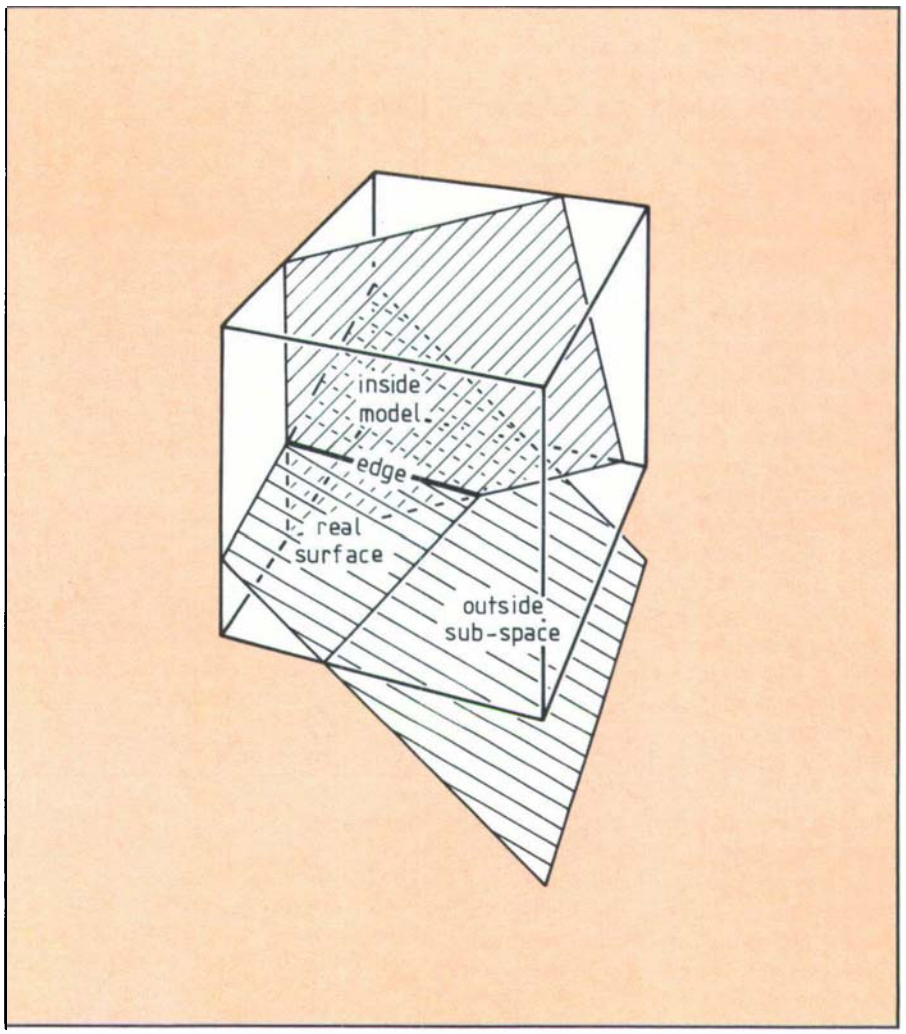


Fig. 7 Finding surface areas
The sub-space cuboid (a leaf of the division tree) is projected onto the surface of interest to form a quadrilateral in it. Part of that quadrilateral lies inside the model, part on its surface, and part outside the sub-space. A membership test serves to discriminate between these possibilities for a random point in the quadrilateral, which decides if its area should, or should not, be added to the accumulating total for the model's whole surface area

Implementation

Excepting VOLE, which was originally run on a DEC PDP-11, all the programs described in this paper were developed on a DEC VAX 11/730. They are all written in standard Fortran 77 with the aim of maximum portability. Some of the programs are already running on Prime equipment, and are being transferred to Apollo workstations and a Honeywell MULTICS system.

The requirements for graphics devices are simple. MEG just needs a line drawing function, VOLE needs a display which can receive blocks of a picture, and DORA generates pictures a scan line at a time. SID and DORA are by far the largest of the programs, but their use of memory is highly coherent, so they are very suitable for virtual memory systems. The authors' VAX has only 1 Mbyte of physical memory, and any one program is only permitted to use

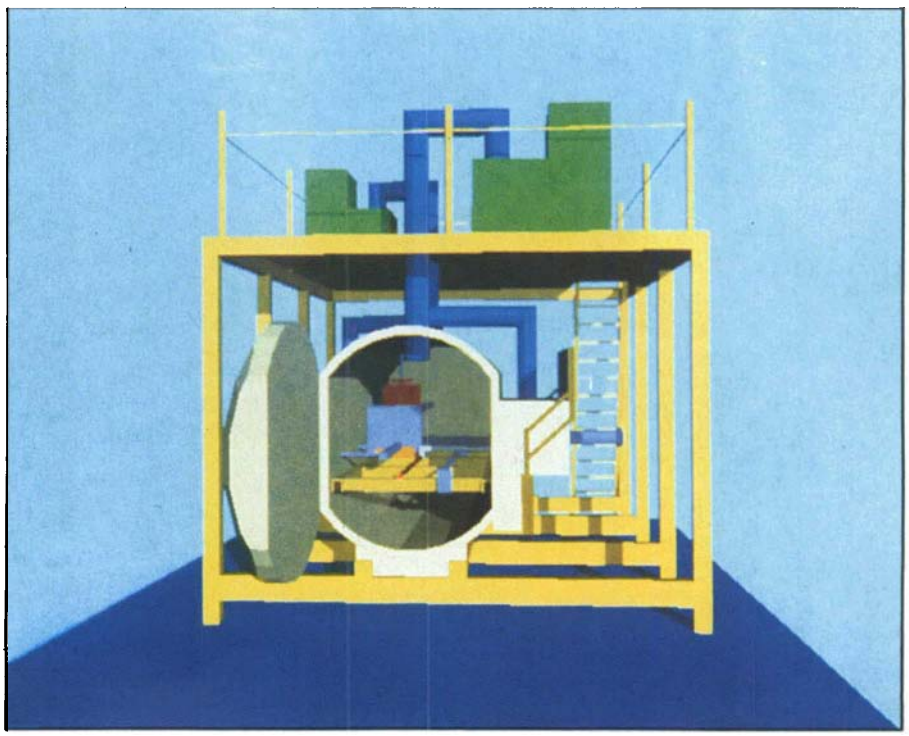


Fig. 8 A vacuum fabrication plant
This was designed by final-year engineering students at Bath University, who also created the model using SID (this picture took about 45 minutes to generate on the VAX 730)

256 Kbytes. Nevertheless, models with 5000 half-spaces can be handled without any noticeable deterioration in performance through paging. Above this size page fault statistics indicate considerable activity, but computation times do not seem greatly affected.

Conclusions

The authors have presented an outline of a number of complementary programs designed to extend the degree of realism associated with solid models from the level currently thought economical. The software can alternatively be considered as a vehicle for pure computer graphics work in which the advantages of solid modelling are offered as incidentals.

There are a number of possible developments of this work. The simplest would be to extend the range of visual effects which can be produced by ray-casting. This is straightforward, given the efficient method for casting a single ray. A more profound enhancement to the programs would be the extension of the types of primitive available. The authors have leap-frogged the obvious next step of introducing the usual solid modelling primitives (quadrics and the torus), and have been developing techniques (broadly similar to those in Ref. 16) based on a more general blended surface formulation (Fig. 9), of which the classical solid modelling primitives are special cases. Spatial segmentation and pruning are again used to give reasonable system performances on what are algebraically very complicated models.

Algorithms with poor performance are often defended on the basis that the development of parallel architectures will increase their feasibility with the passage of time. As well as giving very good performance on a conventional von Neumann machine, the techniques of spatial segmentation outlined in this paper are also particularly suitable for parallel implementation, and a detailed simulation has been performed on a parallel-processor version of VOLE [17] with good results.

Acknowledgments

The authors would like to acknowledge the help of their colleagues Andrew Wallis and Kevin Quinlan, who created some of the models and code, Normalair-Garrett plc (Fig. 4. is one of their products), undergraduate students Steven Plain, Richard Spink and Paul Wells, who created the model in Fig. 8, and the UK Science and Engineering Research Council, who funded some of this work.

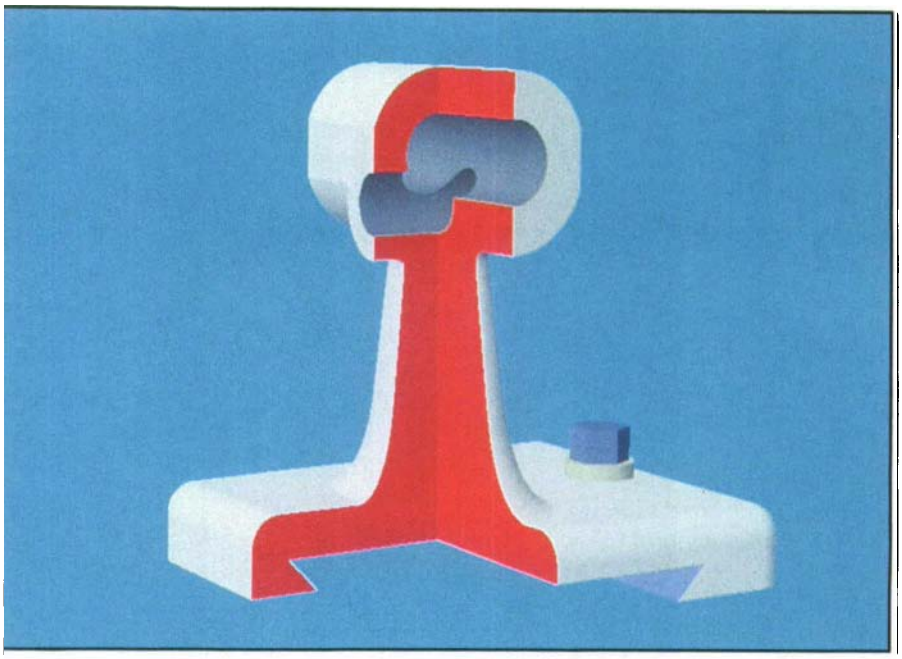


Fig. 9 A picture from the experimental blended surface modeller
The object is a casting from a gear testing machine, which has been modelled using smoothly curving surfaces

References

1 REQUICHA, A. A. G.: 'Representations for rigid models: theory, methods, and systems', *ACM Computing Surveys*, 1980, **12**, (4), pp. 437–464
2 WOODWARK, J. R., and QUINLAN, K. M.: 'Reducing the effect of complexity on volume model evaluation', *Computer-Aided Design*, 1982, **14**, (2), pp. 89–95
3 REQUICHA, A. A. G., and VOELCKER, H. B.: 'Constructive solid geometry'. Production Automation Project Technical Memo 25, University of Rochester, Rochester, NY, USA, Nov. 1977
4 WOODWARK, J. R., and QUINLAN, K. M.: 'The derivation of graphics from volume models by recursive division of the object space'. Proceedings of Computer Graphics 80 Conference, London, England, Aug. 1980, pp. 335–343
5 WOODWARK, J. R., and WALLIS, A. F.: 'Graphical input to a Boolean solid modeller'. Proceedings of CAD 82, Fifth International Conference on Computers in Design Engineering, Brighton, England, March 1982
6 REQUICHA, A. A. G.: 'Part and assembly description languages'. Production Automation Project Technical Memo 28, University of Rochester, Rochester, NY, USA, Nov. 1977
7 TILOVE, R. B.: 'Set membership classification: a unified approach to geometric intersection problems', *IEEE Transactions on Computers*, 1980, **C-29**, (10), pp. 874–883
8 ROTH, S. D.: 'Ray casting for modelling solids', *Computer Graphics & Image Processing*, 1982, **18**, (2), pp. 109–144
9 GLASSNER, A. S.: 'Space subdivision for fast ray tracing', *IEEE Computer Graphics & Applications*, 1984, **4**, Oct., pp. 15–22
10 BLINN, J. F.: 'Simulation of wrinkled surfaces', *Computer Graphics*, 1978, **12**, (3), pp. 286–292
11 BLINN, J. F., and NEWELL, M. E.: 'Texture and reflection in computer generated images', *Communications of ACM*, 1976, **19**, (12), pp. 542–547
12 PEACHY, D. R.: 'Solid texturing of complex surfaces'. Proceedings of SIGGRAPH 85, San Francisco, CA, USA, July 1985, pp. 279–286
13 PERLIN, K.: 'An image synthesiser'. Proceedings of SIGGRAPH 85, San Francisco, CA, USA, July 1985, pp. 287–296
14 LEE, Y. T., and REQUICHA, A. A. G.: 'Algorithms for computing the volume and other integral properties of solids', *Communications of ACM*, 1982, **25**, (9), pp. 635–641 (Part 1), pp. 642–650 (Part 2)
15 SARRAGA, R. F.: 'Computation of surface area in GMSolid', *IEEE Computer Graphics & Applications*, 1982, **2**, Sept., pp. 65–77
16 MIDDLEDITCH, A. E., and SEARS, K. E.: 'Blend surfaces for set-theoretic volume modelling systems'. Proceedings of SIGGRAPH 85, San Francisco, CA, USA, July 1985, pp. 161–170
17 WOODWARK, J. R.: 'A multiprocessor architecture for viewing solid models', *Displays Journal*, 1984, April, pp. 97–103

Dr. J. R. Woodwark is with IBM UK Scientific Centre, Athelstan House, St. Clement Street, Winchester, Hants. SO23 9DR, England. Dr. A. Bowyer is with, and Dr. J. R. Woodwark was formerly with, the School of Engineering, University of Bath, Claverton Down, Bath BA2 7AY, England.