A multiprocessor architecture for solving spatial problems

Adrian Bowyer†, Philip J. Willis†, and John R. Woodwark‡

†School of Mathematics and ‡School of Engineering, University of Bath, Claverton Down, Bath, BA2 7AY, UK

A multiprocessor architecture for the solution of spatial and graphical problems by recursive subdivision is proposed. The trees that result from such divisions are mapped on to a closed graph, which is then considered as a network of processors. Simulations of these configurations have been performed with encouraging results. Proposals are made for the implementation of such machines.

(Received October 1980)

1. Introduction

A number of computational problems, especially those of a spatial or geometric nature, may be effectively solved by techniques of successive subdivision. As well as effecting a reduction in processing time over methods requiring global search, an overall simplification may result. Additionally, because such techniques generate independent sub-problems, they seem well suited to implementation on parallel hardware. This paper presents a possible processor architecture for such tasks.

As a problem is divided, the subdivisions may be considered as a tree, having the original problem as root and the solvable sub-problems as leaves. A first approach to a hardware configuration suggests that a processing unit be assigned to every possible node on the tree. This is clearly both impractical, because of the large number of processors which would be required, and inefficient, because for most real problems the tree generated is extremely sparse. The authors put forward a method of connecting a feasible number of processors in a way which retains many of the desirable features of a full tree. It also gives acceptable processor utilisation, while avoiding the problems associated with a single bus structure or shared memory (Willis, 1978).

2. Graphical basis for the architecture

How may a relatively small number of nodes be arranged in a pattern which resembles a large tree and retains some of its useful properties?

The Petersen graph (**Fig. 1**) is well known in graph theory. Its nodes have a *valency* of 3; that is to say, each node is connected to 3 other nodes. It has the property that, starting from any node, a circuit of at least five nodes must be traversed in order to return to the starting node. This minimal circuit length is called the *girth* of the graph, and the Petersen graph contains the minimum number of nodes for girth 5 to be possible in a valency 3 graph. In other words, 5 is the maximal girth for ten nodes of valency 3.

The Petersen graph can be redrawn as in **Fig. 2**, with the central node chosen arbitrarily, as the graph is symmetrical. In this form, it can be seen that the central node can be considered as the root of three binary trees, shown in Fig. 2 as solid lines. If this graph were implemented as a network of processors, a problem on any processor could be divided down any or all of the three trees which start at that processor. The maximal girth property of the Petersen graph gives the maximum depth of these trees for the ten processors employed. Thus a problem will be divided the largest number of times before arriving back at any node. Other graphs with this property may be constructed, and the theory may be extended to graphs of higher valency. We restrict ourselves to graphs of valency 3 to limit the problems of interconnection of processors in a practical machine.

It is not possible to construct graphs of valency 3 and any given girth with less than certain numbers of nodes. It can be shown (Biggs and Ito, 1980) that if the girth g of a valency 3 graph is even, then it must have at least m nodes, where

$$m = 2(1 + 2 + 2^2 + \ldots + 2^{r-1}) \quad (1)$$

where

$$r = \frac{g}{2} \quad (2)$$

For example, to achieve a girth of 6, 14 nodes are needed and to achieve a girth of 8, 30 nodes are needed. It has been proved (Singleton, 1966) that graphs of even girth containing this minimum number of nodes can only be constructed for girths of 2, 4, 6, 8 and 12. For other girths more nodes are needed. **Figs. 3** and **4** show the 14 node graph of girth 6 and the 30 node graph of girth 8. Like the Petersen graph these graphs are symmetrical, meaning that the structure looks the same from every node.

We have chosen these graphs as being representative of the size of machine that it would be practical to construct although

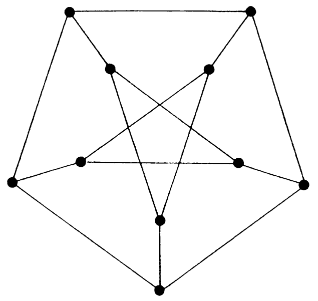

Fig. 1 The Petersen graph

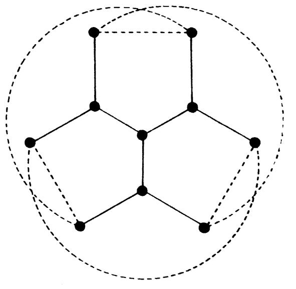

Fig. 2 The Petersen graph as a tree

3

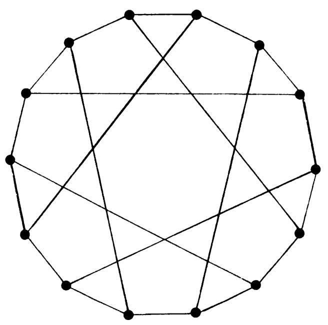
Fig. 3 The 14 node girth 6 graph

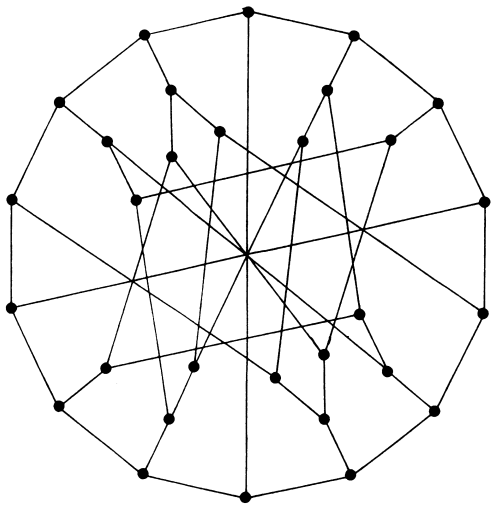
Fig. 4 The 30 node girth 8 graph

we do not rule out the possibility of smaller or much larger devices.

3. Applications

The authors have conceived the processor architecture presented in this paper in the context of certain spatial problems which they feel to be of importance, and which are susceptible to solution by subdivision, an approach commonly known as 'divide and conquer'. This does not preclude its application to other tree-structured problems, although the realisation of any such processor might depend on the attributes of the class of problems to be handled. Spatial problems are characterised in many cases by requiring a large amount of arithmetic, but more moderate data flow. This conforms well with the advantages and limitations of the approach presented here.

Subdivision techniques fall into one of two classes. In the first class are those techniques which divide the data according to a particular characteristic, such as colour. In the second class are those techniques by which the space that the structure occupies is divided.

The top down approach to the travelling salesman problem (du Feu, 1975) provides an example of the first class. In this case, the set of points to be visited is clustered into 'towns', which are dealt with in turn. Shamos (1975) offers other applications to structures in the plane, such as the computation of convex hulls and the Dirichlet tessellation. In the second class is, for example, the Warnock (1969) hidden-surface algorithm. This involves the recursive subdivision of the area of a picture until sub-areas are simple enough to be analysed easily, or small enough to reject. This approach may be extended into three dimensions as a search technique (Eastman and Lividini, 1975), or as an efficient method of evaluating a set-theoretic volume model (Woodwark and Quinlan, 1980). In 3-dimensional cases the amount of processing on conventional machines is considerable. Evaluation times for volume models, in particular, are notoriously long.

4. Simulation

To investigate how the proposed architecture might behave in practice, the authors have written a FORTRAN program to perform discrete time simulations of processor networks connected in graphs of the type described in Section 2.

The simulation starts by assigning to a randomly chosen processor an original problem of size n_0. That processor examines the size of the problem and decides whether to split it into a number of sub-problems or to solve it. The criterion for making this decision is discussed below. If the problem is to be split, the processor takes a number of time steps, t_s, to split the problem, which is a function of problem size. If the problem is to be solved this occupies t_w time steps, which is a second function of the size of the problem. The number of sub-problems, k, into which a problem is split at each stage is constant for each simulation. Each processor has a stack associated with it, and, when a processor splits a problem into k sub-problems these k sub-problems are pushed on to the stack. A processor with problems on its stack examines the top problem and decides whether to split it or to solve it as previously described. A processor with no problems on its stack interrogates its neighbours to find the largest problem on their stacks, if any, and then steals that problem and solves or splits it. This means that processors with problems do not spend time trying to dispose of some of them to idle neighbouring processors, but that the idle processors actively search for problems to solve. Suppose the first processor splits its problem. When it has finished splitting, its neighbours take some of the sub-problems off its stack and solve or split those sub-problems, leaving the first processor with a lighter load. When those sub-problems have again been split and pushed on to the neighbouring processors' stacks the neighbours' neighbours can steal problems from their stacks, and so on. The original problem is thus recursively divided and spread throughout the network.

If a real problem is being split into k sub-problems, the sum of the sub-problems often exceeds the size of the original problem. When a scene is divided by the Warnock algorithm, for example, some picture components may span the dividing lines, and must be considered in both sub-scenes. This is simulated by multiplying the size of each sub-problem, n_s, by a random variable to give a final sub-problem of size n_f

$$n_f = n_s(1 + x) \qquad (3)$$

x is a pseudo-random number realised from the exponential distribution with expected value 0·125. This value is chosen as a reasonable figure for the applications envisaged.

A problem of size n is split into the k smaller sub-problems required by realising k-1 pseudo-random variables from the uniform distribution on $(0, n)$ and slicing the problem at these points to give the values n_s mentioned above. Thus the expected size of n_f, $E(n_f)$, is

$$E(n_f) = \frac{n(1 + 0{\cdot}125)}{k} \qquad (4)$$

For reasons explained below it was decided that the time t_w for a processor to solve a problem of size n should be a function of n^2

$$t_w = an^2 \qquad (5)$$

and that the time t_s to split a problem of size n should be considered to be linear in n

$$t_s = bn \qquad (6)$$

These equations were designed to be consistent with the type of problem which the proposed processor architecture is intended to solve. The values of a and b were varied as part of the simulation study. Again taking the Warnock algorithm as an example: if it is required to eliminate the hidden portions of n picture components, each has potentially to be compared with all the others, leading to an $O(n^2)$ time for solution. To partition the components into k sets each component need only be examined once, leading to an $O(n)$ time for the partition to be performed.

Given this behaviour, how may a decision be made on whether or not to split a given sized problem? Suppose a problem of size n were to be solved by one processor only, and that, in order to solve it, that processor will split it i times, push the resulting sub-problems onto its stack, and then solve them all. In the following section capital Ts have been used for total times. If the expected increase in problem size on splitting is c (in this case $c = 1{\cdot}125$) then the expected time taken to perform all the divisions, $E(T_s)$, will be

$$E(T_s) = bn + \frac{c(bnk)}{k} + \frac{c^2(bnk^2)}{k^2} + \ldots + \frac{c^{(i-1)}[bnk^{(i-1)}]}{k^{(i-1)}} \tag{7}$$

The k terms cancel and the series sums to give

$$E(T_s) = bn\left(\frac{c^i - 1}{c - 1}\right) \tag{8}$$

The expected time, $E(T_w)$, to solve the resulting k^i problems each of expected size

$$E(n_f) = \frac{c^i n}{k^i} \tag{9}$$

will be

$$E(T_w) = \frac{ac^{2i}n^2}{k^i} \tag{10}$$

Thus the expected total computation time, $E(T_t)$, will be

$$E(T_t) = E(T_s) + E(T_w) = bn\frac{c^i - 1}{c - 1} + \frac{ac^{2i}n^2}{k^i} \tag{11}$$

$E(T_t)$ is at a minimum with respect to the number of divisions performed, i, when

$$i = \frac{\ln\{an\,(c-1)\,[\ln(k) - 2\ln(c)\,] - b\ln(c)\}}{[\ln(k) - \ln(c)]} \tag{12}$$

As a rough guide, it was decided that a problem of size n should be split when the values of a, b, c and k gave a value of i greater than 0·5 in Expression (12).

Is this a good criterion to use when there is more than one processor available to work on the problem? Even if more than one processor is available, the expected collective time for which all the processors will be occupied with that problem, and therefore unavailable for anything else, will still be $E(T_t)$. However, when many working processors have idle neighbours, division is more effective than the criterion indicates. In general this would only be the case at the beginning and end of the time that the machine is working.

The behaviour of the processor architecture is, in many ways, analogous to the spread of an epidemic across a regular lattice with re-infection allowed (see for example Hammersly, 1966). A great deal of work has been done on such epidemic spread, much of it by simulation, but the graphs of node interconnection proposed in this paper are more complex than those obtained from simple lattices, so the results of such epidemic studies are of limited use.

Fig. 5 shows the time taken for the 14 processor machine (Fig. 3) to solve problems of increasing initial size, n_0, for various values of the constants a and b in Eqns (5) and (6) with problems being split into two sub-problems at each stage ($k = 2$). The curves are the best fit parabolas of the

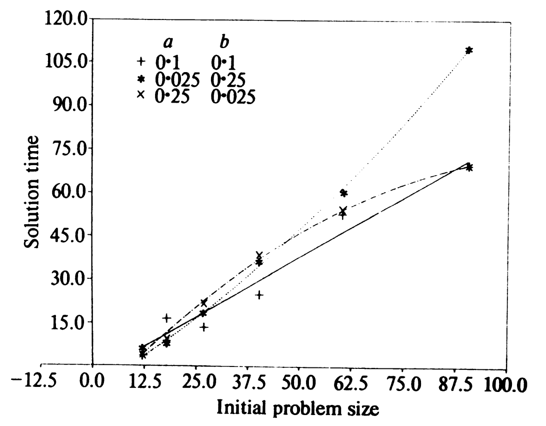

Fig. 5 Solution time against problem size for the 14 node processor

three sets of simulation results. Parabolas were chosen because of the quadratic nature of Eqn (5). It was thought that to fit straight lines would be prejudicial to the reader's perception of the results, although in most cases the quadratic term in the fitted curve is not statistically significant. Each data point on the graphs of results is an average of three independent simulations.

The device behaves much as would be intuitively expected. If the problem is difficult to split and easy to solve (i.e. b is large and a is small) it is of an unsuitable type for the proposed architecture. The device then takes much longer times to solve larger problems (the dotted curve through the data marked by an asterisk) and the quadratic term in the fitted curve is positive (though small). If the problem is as easy to split as it is to solve the device behaves linearly over the range of problem sizes illustrated in Fig. 5 (the continuous line through the points marked by a plus). If splitting is easier than solving (the chain line through the points marked by a cross) the problem is well suited to the device and it behaves rather better than linearly over the range.

Fig. 6 shows the behaviour of the 14 processors machine on very large problems with $a = 0{\cdot}1$ and $b = 0{\cdot}1$ (the linear result in Fig. 5). The device behaves slightly worse than linearly up to very large problem sizes. This is what would be

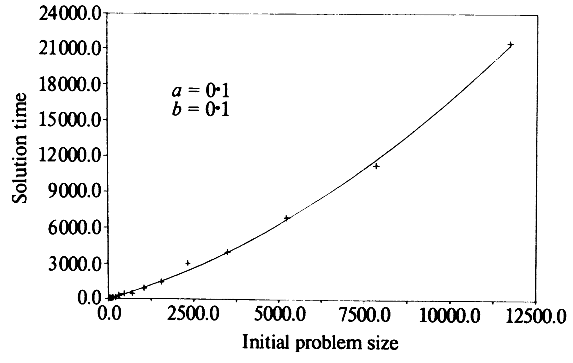

Fig. 6 Solution time against problem size for large problems

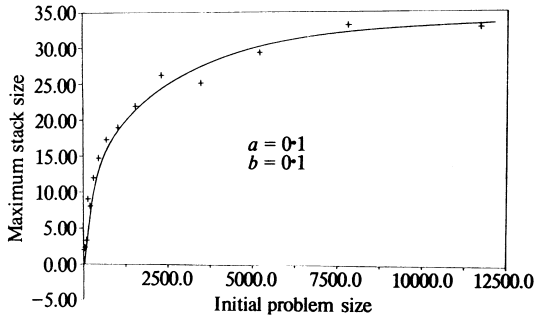

Fig. 7 Maximum stack usage against initial problem size

expected from the O[$n$log(n)] characteristic obtainable by the recursive divide and conquer technique on which the device is based.

Fig. 7 shows the behaviour of the maximum size of stack required over all of the processors throughout the solution of the whole problem under the same conditions as Fig. 6. The relative demand for stack space tails off sharply as problem size increases. This is as would be expected, as the depth of the division tree increases logarithmically with problem size.

Fig. 8 shows the percentage of the total time taken to solve a problem that the machine is idle. Idle time is defined as the sum of all the times for which processors are idle divided by the number of processors. Most of this idle time is concentrated at the beginning of the solution of a problem, when the processors are waiting for the first processor to split the original problem, and, to a lesser extent, at the end, when the whole of the network is not needed to solve the small part of the original problem which remains.

Fig. 9 is a histogram of the number of sub-problem solutions produced in one typical simulation against time for a = 0·1, b = 0·1 and an initial problem size of 60. The histogram is fairly typical of the behaviour of the device over a wide range of conditions. Answers are produced mostly during the last two thirds of the time taken to solve a problem. The early period is largely devoted to splitting up the problem and dividing it throughout the network. It is encouraging that the histogram does not exhibit sharp peaks as these would imply contention problems in the communication of answers out of the device.

Fig. 10 shows the time taken for the 30 processor machine (Fig. 4) to solve problems of increasing initial size operating under the same conditions as those depicted for the 14 processor machine in Fig. 5. The behaviours of the two machines are similar for the unsuitable problem type (solution easy, splitting difficult: the dotted curve through the data marked by an asterisk). This is because the machine has little incentive to split problems under these conditions, so the problems never spread throughout the whole graph and an increase in the number of nodes offers no advantage. Under the constraint that splitting and solution are of the same difficulty (the continuous line through the points marked by a plus) the smaller machine takes about 25% longer. When splitting is easier than solving (chain line through the data marked by a cross) the smaller machine takes about 65% longer.

All the simulations discussed thus far have involved the splitting of problems into two sub-problems (k = 2). This is because any tree can be represented as a binary tree and this is therefore of general interest. Most implementations of the Warnock algorithm, which has been used as an example throughout, split the picture data into four congruent rectangles similar to the original rectangular surround of the whole image. The authors have done some simulations of the device with k set to 4. The machine seems to work faster under these conditions than with k set to 2. This is consistent with the results that the machine runs fastest on those problems which are easy to split and difficult to solve. In both cases the problem is spread quickly throughout the network. Splitting into four is also of interest because the nodes of the graphs have valency 3. If a problem is split into four, three parts of it can be stolen by a processor's three neighbours and the fourth part can be retained by the original processor. Splitting fewer than four ways does not spread the problem throughout the network so quickly, splitting more than four ways can cause problems to be stacked unnecessarily.

These simulation studies demonstrate the behaviour of the proposed architecture. Solution times degrade gracefully as load increases, the requirement for stack memory grows better than linearly with increasing load, and, if a reasonable load has been placed on the machine, each processor does not spend too much of its time idle while waiting for problems. The times at which answers to sub-problems are produced are distributed uniformly enough over the total time taken

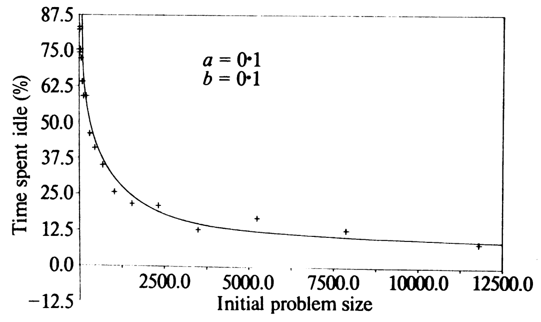

Fig. 8 The percentage of time spent idle against initial problem size

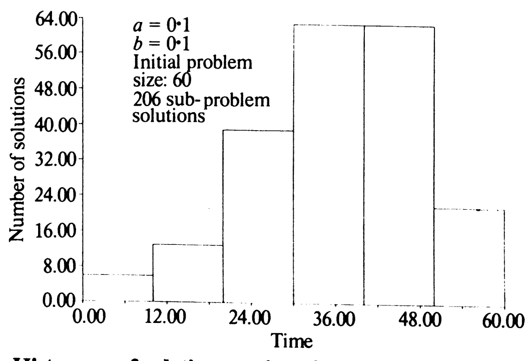

Fig. 9 Histogram of solutions against time

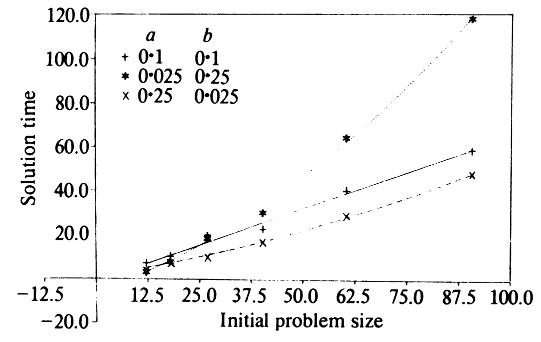

Fig. 10 Solution time against problem size for the 30 node processor

to solve problems that there should not be too many contention problems in passing those answers out of the machine.

5. Projected implementation

In this section are considered some of the possible solutions to the problems that would be encountered in constructing and operating a machine with the proposed architecture.

The simplest and cheapest way to implement a processor network would be to design the nodes around a conventional microprocessor. The microprocessor at each node will have a certain amount of random access memory associated with it and possibly some read only memory to facilitate resetting or starting the machine. The nature of the spatial problems which the machine is designed to solve make it desirable that each node be provided with a hard-wired floating-point processor. The links between neighbouring nodes will be made with parallel connectors. Processors can communicate with their neighbours by direct memory access to make the transfer of sub-problems quick and simple.

The whole network can be supervised by a single master processor, probably based on the same microprocessor as that used in each node. The master processor will be responsible for initialising all of the nodes with a program and any global data. It will also be responsible for inserting the original problem at some node.

The way in which a processor communicates the answers which it generates to the outside world depends, to a certain extent, on the nature of the problems which the device is solving. The answer to a sub-problem might need to be communicated to its parent problem and so on right up the problem tree to the root. If this is necessary a flag can be left on the stack from which a problem had been stolen and replaced by the solution when it became available. For most spatial applications the answer to a sub-problem is immediately useful, for example when it is part of an image to be displayed on a graphics device. In these circumstances the answer can be communicated directly to the outside world without the necessity of passing it back up the tree. To facilitate this, each processor can be made a leaf on a separate binary tree. Each junction in this tree continuously monitors the two junctions or processors below it to see if an answer to a sub-problem is available. When one is, it is passed up the binary tree to its root where it can be communicated to a graphics device or to the master processor. The junctions of this tree will be very simple and cheap. This tree might also provide an alternative method of loading all the nodes at the start of a problem. Programming of the device should not present an especial problem, as the same program is loaded into each processor. If these are standard microprocessors, conventional languages can be used to program the device. Alternatively, the special characteristics and requirements of the device could be made transparent to the user within a language oriented towards those problems which are amenable to recursive subdivision (Keller, 1980).

The network proposed is potentially highly fault tolerant. If, when a node was physically removed from the network for servicing, its location appeared to its erstwhile neighbours to be permanently idle, those neighbours would simply ignore it. In order to isolate completely a working node all three of its neighbours would have to be inoperative at the same time.

6. Conclusions

The authors believe that the processor architecture that they have proposed offers an attractive alternative to bus-structured multiprocessor systems, or systems with shared memory, with their associated contention difficulties. The range of applications of the device is not universal, but does include a large class of spatial and geometrical problems which are receiving increasing attention.

The authors are currently working on an implementation of the processor architecture discussed in this paper.

Acknowledgements

The authors are indebted to Professor Robin Sibson for his help with the graph theoretic aspects of this work, and to Mr David Cunningham for developing a program to check the graphs used.

References

Biggs, N. L. and Ito, T. (1980). Graphs with even girth and small excess, *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 88 No. 1, pp. 1–10.

Bollobas, B. (1978). *Extremal Graph Theory*. Academic Press, London.

Eastman, C. M. and Lividini, J. (1975). Spatial search, Research Report No. 55, Carnegie-Mellon University Institute of Physical Planning.

Feu, D. du (1975). The top-down travelling salesman, Research Report No. 8, University of Edinburgh Department of Artificial Intelligence.

Hammersly, J. M. (1966). First passage percolation, *Journal of the Royal Statistical Society Series B*, Vol. 28, pp. 491–496.

Keller, R. M. (1980). Data structuring in applicative multiprocessing systems, *1980 Lisp Conference Record*, pp. 196–202. Stanford, CA.

Shamos, M. (1975). Computational geometry, PhD thesis, Yale University.

Singleton, R. C. (1966). On minimal graphs with maximal even girth, *Journal of Combinatorial Theory*, Vol. 1, pp. 306–332.

Warnock, J. E. (1969). A hidden-surface algorithm for computer generated halftone pictures, Report TR 4–15, University of Utah Computer Science Department.

Willis, P. J. (1978). Derivation and comparison of multiprocessor contention measures, *IEE Journal of Computer and Digital Techniques*, Vol. 1 No. 3, pp. 93–98.

Woodwark, J. R. and Quinlan, K. M. (1980). The derivation of graphics from volume models by recursive subdivision of the object space, paper presented at the Computer Graphics 80 Conference, Brighton. I.P.C.