Position Paper:
Display Hardware for Boolean Expression Models

A. L. Thomas

School of Engineering & Applied Sciences, The University of Sussex
Brighton, Sussex

In any discussion of graphics hardware there appear to be two basic positions which can be adopted. The first is that of the technologist, who is primarily concerned with what it is possible to make and how to make it. The second is that of the system designer who is more interested in what it would be desirable to make. To be a designer it is necessary to have a view of the future ... or at least a view of a plausible future! This is only possible with a reasonably sound idea of what the technologists might be persuaded to provide. I suspect that most of the "images of the future" which have guided or moulded current proposals have been around for some time. In spite of this it is a good preliminary exercise to set out a brief statement of the main ideas which lie behind current developments, before homing in on specific hardware proposals.

Perhaps the most important idea which has emerged with the development of graphics hardware is the provision of a common base for all electronic images. This was made possible by the development of the digital frame store (Newell, Newell, Sancha, 1972), which depended on the evolution of memory integrated-circuits. The second important idea is the ability to interact with synthesised images in real-time. The third important idea is the development of higher level languages for interacting with objects in displays. Fourthly, the least exploited idea to date is the capability of automatically extracting higher-level object information from found images, such as those received from the TV camera. Underlying all these ideas is the unifying concept of improving the man-machine and world-machine interfaces to improve communication which, it is hoped, will allow machines to operate at a higher or more intelligent level.

A different set of ideas, linked with the development of display hardware, seems to be emerging from work on array processors developed for image analysis work. These ideas are concerned with a graphical counterpart to a program listing in the sense that program code can be read meaningfully as text on one hand, but can also be employed as instruction codes, on the other. Diagrams, charts and graphs, in a similar way, can be understood as images but can also act as instructions to an array of processors considered as a display. There is at least the possibility that display generating hardware could be designed to have very powerful general, parallel processing capabilities. This would provide an interesting twist to the "wheel of reincarnation"!

The initial idea behind the specific hardware proposals outlined below was that in a computer environment drawings would be replaced by mathematical models for design and analysis work. This meant that the highest level of modelling language that was compatible with the interactive capabilities needed for design work could be

sought to replace the line based systems that had evolved from a draughtsman's methodology. At present this appears to be provided by a Boolean-expression modelling system (Thomas, 1986), though in the future there is no reason why more complex modelling structures should not take over. The advantage that the Boolean model has is that it provides a system with a built in hierarchy, useful for developing efficient algorithms to take advantage of scene coherence. It also provides a natural way of representing object-interference or volume overlap, which often needs to be located as an unwanted consequence of editing operations. These are both capabilities which will have to be included in future systems even if they provide higher level modelling facilities.

The second idea controlling the way in which the hardware described below evolved was the need to generate real-time, moving, synthetic-images. This was important for simulation and teaching systems. It was useful in CAD work even if not always demanded by current practice, and in the long run appeared essential if better man machine communication was to be achieved.

Real-Time Boolean Expression Modelling System

The first real-time raster based systems were developed by Schumacker (Rougelot, 1969) and Watkins in 1969 and 1970 respectively.

Watkins' algorithm was a solution to real-time display generation which in principle was hard to improve. There were four steps to producing a moving synthetic image.

1. Transformation Processing.
2. Divide and Conquer Scan-Line, Scene Coherence Processing.
3. Pixel Rendering or Value Interpolation.
4. Illumination Calculations.

What did seem open for improvement was the object modelling stage of the process.

Two main approaches to modelling emerged from a study of this part of the system (Thomas, 1976). The first was based on a set of homogeneous vertex coordinates, (x,y,z,w), the second on a set of facet plane equation coefficients, (a,b,c,d). The vertex based approach consisted of different ways of setting up triangulated networks. A characteristic of this model was that shape changes could be most simply achieved by transforming vertex coordinates: edges and facets depending on interpolation to complete the model. One way of implementing this system was to represent a set of tetrahedra as 4 x 4 graph matrices and then employ a series of operations to glue the tetrahedra together by adding together the appropriate matrices. A variation on this approach was based on a hierarchy of convex hulls.

The second main method grew out of a study of overlap operations. The facets of the target volume were represented by a set of plane half-spaces. These unbounded planes partitioned the object space into a collection of convex blocks: the required volume being defined by writing down the Boolean expression which selected the appropriate collection of convex blocks. This expression model was relatively easy to convert automatically into a graph-matrix or convex hull model when required, and vice versa. Shape modification following the second approach was achieved by

transforming the surface plane equations. However both point and surface based models were capable of being transformed using the same matrix multiplication hardware.

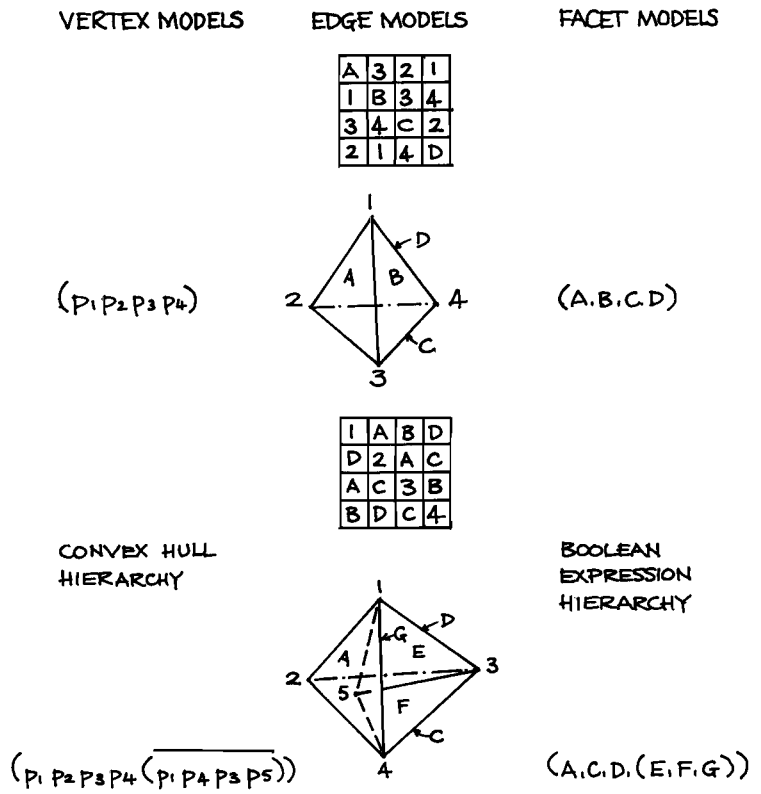

Fig. 1 Alternative Volume Models

If an object is entered into the computer system in the form in Fig. 2, defined as the interaction of a sequence of sub-models where the Target Volume

```
T = C.d.(!E+f).D;
```

the lower case names representing plane half-spaces, and the capital letter names representing the other objects:

```
C = a.b;

E = e.r.(l+m+!(k+j)+n+s);

D = t.u.v;
```

Then the first step is to substitute for the object names to get:

```
T = a.b.d.(!(e.r.(l+m+!(k+j)+n+s))+f).t.u.v;
```

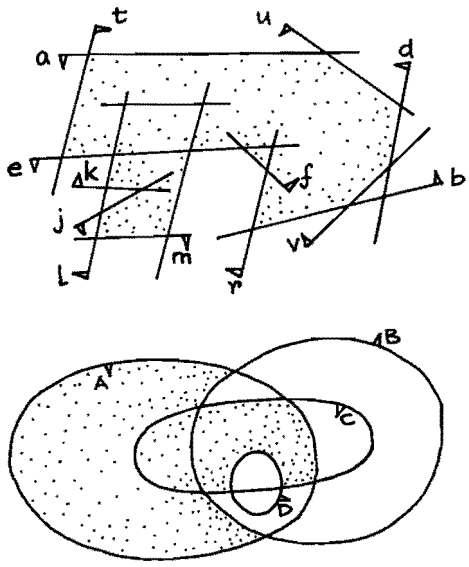

Fig. 2 Boolean Expression Model Standard Form

The next step is to apply De Morgan's theorem to all complemented sub-expressions to give:

T = a.b.d.(!e+!r+(!l.!m.(k+j).!n.!s)+f).t.u.v;

and the final step is to collect all the free plane half-spaces at each level into convex groups that can be renamed in the way shown in Fig. 2.

T = a.b.d.t.u.v.(!e+!r+f+(!l.!m.!n.!s.(k+j)));

(A (B (C (D))))

This process is implemented using the tree structures which result from parsing the input expressions defining the various objects, in the way illustrated in Fig. 3.

Figure 2 illustrates the input models which result from parsing the volumes labelled T, C, E and D in Fig. 3, in the trees numbered 1, 2, 3, 4. The first step in processing these trees was to link incomplete references to the relevant sub-trees. In this case the result is a single expanded tree for T. Applying De Morgan's theorem consists of switching all the operator nodes in the complemented sub-trees and complementing all the half-space references. This makes it possible to represent the expression by a Knuth tree where each deeper level corresponds to alternating union and intersection operators. The final step is to collect all the free half-space references together at each level in the Knuth tree at the beginning of each row. Renaming these units allows the schematic representation using convex elements shown in Fig. 2. This diagram corresponds to the tree labelled 6 in Fig. 3, which would be implemented in the form of the tree labelled 5. The

convex units in this example are (a.b.d.t.u.v), (!e+!r+f), (!l.!m.!n.!s) and (k+j). Renaming these convex units results in an object defined as A.(B+(C.(D))) or A(B(C(D))) where the first level is known.

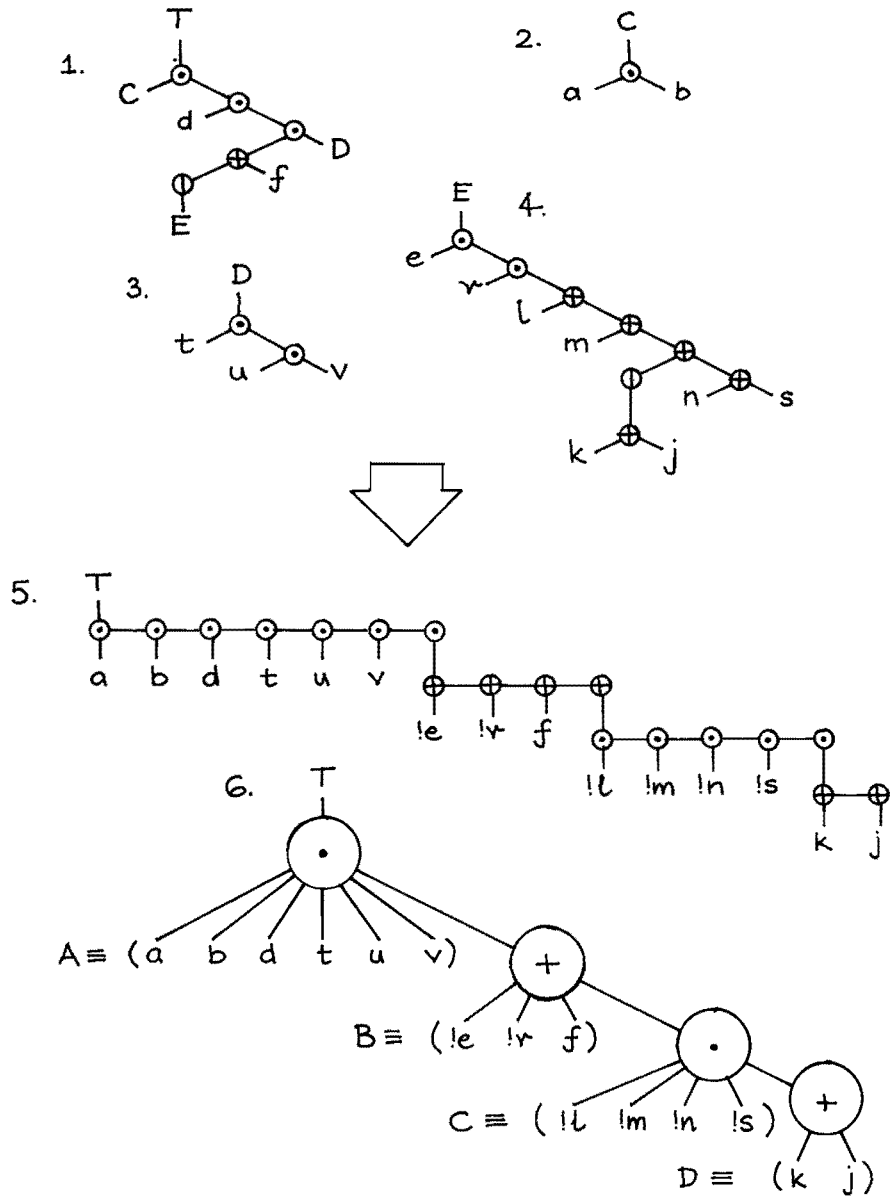

Fig. 3 Boolean Expressions as Tree Structures

Once the convex components have been set up in this way it is possible to carry out a further series of operations on these trees which correspond to the algebraic manipulation of the original expression described below. The purpose of these operations is to generate a Boundary model from the simple expression form used for input. This process was initially developed to control the conversion of the expression model into its corresponding graph-matrix form.

Take as an example the triangular object (A.B.C). It is a natural step to define the boundary of this object -- expressed as @(A.B.C) -- as made up from the three sides of the triangle. The first side of the triangle will be the boundary of the half-space A denoted by @A where it lies inside (B.C), in other words @A.B.C. Similarly the second and third sides will be @B.A.C and @C.A.B respectively, giving in total:

$$@(A.B.C) = @A.B.C + @B.A.C + @C.A.B$$

This idea can be extended by considering the triangle to be a hole rather than a solid. In which case the expansion becomes:

$$@(\overline{A.B.C}) = @(\overline{A}+\overline{B}+\overline{C})$$

$$= @\overline{A}.B.C + @\overline{B}.C.A + @\overline{C}.A.B$$

From this example two swapping rules can be extracted:

$@(A.B) \rightarrow @A.B + @B.A$ $\qquad$ $@(A+B) \rightarrow @A.\overline{B} + @B.\overline{A}$

If these swapping rules or productions are applied to the previous example, the results will be the set of boundary segments shown in Fig. 4 as B1, B2, B3 and B4 where these elements are defined as follows:

$$@(A.(B+(C.(D)))) \rightarrow @A.(B+(C.(D))) + @(B+(C.(D))).A$$

$$@(B+(C.(D))).A \rightarrow @B.A.(\overline{C.(D)}) + @(C.(D)).A.\overline{B}$$

$$@(C.(D)).A.\overline{B} \rightarrow @C.A.\overline{B}.(D) + @D.A.\overline{B}.C$$

Renaming:

$B1 = @A.(B+(C.(D)))$ $\qquad$ $B2 = @B.(A.(\overline{C}+(\overline{D})))$

$B3 = @C.(A.\overline{B}.(D))$ $\qquad$ $B4 = @D.(A.\overline{B}.C)$

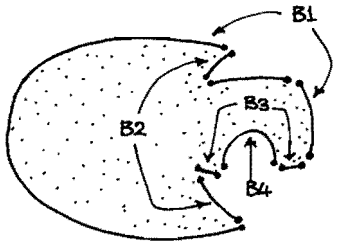

Fig. 4 Boolean Expression Boundary Operator

An alternative expansion can be obtained by using a different swapping rule for the '+' operator.

@(A+B) → @A + @B

Applying this rule to the previous example gives:

@(A.(B+(C.(D)))) → @A.(B+(C.(D))) + @(B+(C.(D))).A

@(B+(C.(D))).A → @B.A + @(C.(D)).A

@(C.(D)).A → @C.A.(D) + @D.A.C

Renaming:

B1 = @A.(B+(C.(D))) B2 = @B.(A)

B3 = @C.(A.(D)) B4 = @D.(A.C)

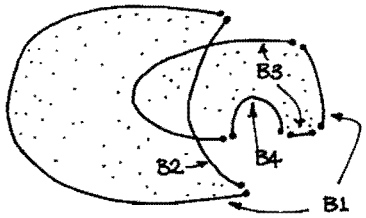

Fig. 5 Alternative Boundary Expansion

The alternative expansion makes use of the fact that pieces of surface which lie inside the solid interior of an object will not be seen. It was developed to handle the problem of hanging faces. This form of boundary expansion allows sheet like objects with no thickness to be handled by a display algorithm on one hand and applying a different interpretation, it also allows regularised set operations to be implemented in the display process on the other.

The Display of Boolean Expression Models

Consider a convex solid defined as the intersection of a set of plane surfaced half-spaces. From any viewing point this set will be partitioned into two sets: those facing towards the viewing position and those facing away from it. The front surface of the convex object can be defined by selecting the front plane which lies furthest away from the viewing position along a given viewing ray. Conversely the back surface of the same object will be given by selecting the back plane which lies closest to the viewing position along any viewing ray. These two surfaces however are infinite in extent, defined in this way, whereas the original object is not. The final step is to select that section of the front surface which lies in front of the back surface, as the visible surface of the convex object. Given a set of convex objects then the nearest one to the eye will be the visible object in a particular viewing direction.

Viewing Ray Distances

The distance to a plane surface from a pixel position R can be evaluated in the way shown in Fig. 6. The distances to two surfaces AA and BB are shown in the diagram as the lengths NR and MR.

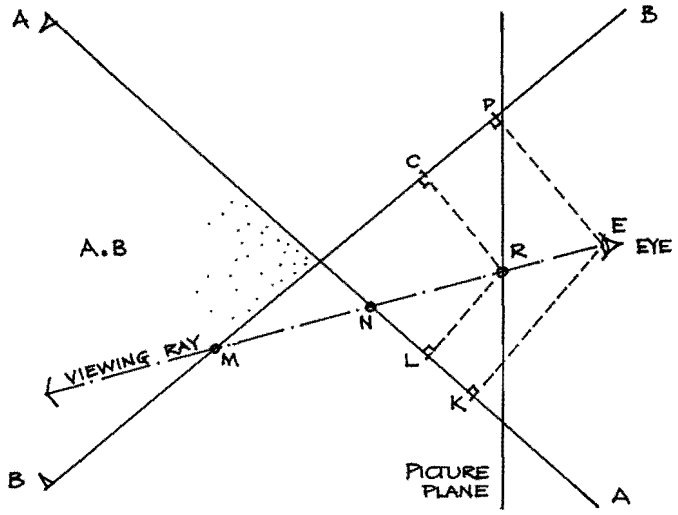

Fig. 6 Viewing Ray Distances

$\frac{MR}{ME} = \frac{CR}{PE}$ $\qquad$ $\frac{NR}{NE} = \frac{LR}{KE}$ $\qquad$ Similar triangles

$MR.PE = CR.(MR+RE)$ $\qquad$ $NR.KE = LR.(NR+RE)$

$MR = \frac{RE.CR}{(PE-CR)}$ $\qquad$ $NR = \frac{RE.LR}{(KE-LR)}$

Since RE will be a constant for any pixel position these relationships can be renamed to give the ratios

$$\frac{k_B^1}{k_B^2 - k_B^1} > \frac{k_A^1}{k_A^2 - k_A^1}$$

where k^1 and k^2 are the perpendicular distances from the plane to the raster point and the eye respectively. These distances can be obtained by calculating the dot product of the coefficients of the plane's equation with the coordinates of the positions of R and E.

Because it was the order in which surfaces cut a viewing ray which was important, the ratios:

$$\frac{k_A^1}{k_A^2 - k_A^1} > \frac{k_B^1}{k_B^2 - k_B^1}$$

could be replaced by:

$$\frac{k_A^1}{k_A^2} \quad > \quad \frac{k_B^1}{k_B^2}$$

These ratios have the advantage that their values lie in the range 0 to 1, as well as preserving the order of surface intersections with a viewing ray. If this ratio is expanded to give

$$\frac{k^1}{k^2} = \frac{a.x^1 + b.y^1 + c.z^1 + d.w^1}{a.x^2 + b.y^2 + c.z^2 + d.w^2} = Z \qquad (1)$$

where Z is the new distance value, then a new display space is defined, and the transformation from the object space to this new space can be summarised by rearranging (1) to give:

$$(a, b, c, d) \cdot \begin{vmatrix} 1, & 0, & -x^2, & 0 \\ 0, & 1, & -y^2, & 0 \\ 0, & 0, & -z^2, & z^1 \\ 0, & 0, & -1, & 1 \end{vmatrix} \cdot \begin{vmatrix} x^1 \\ y^1 \\ Z \\ 1 \end{vmatrix} = 0$$

This matrix defines the perspective transformation for the plane (a, b, c, d). The advantage of using this form is that this matrix can be concatenated with rotation and translation matrices in the same way that is done in point processing systems. The importance of this is that point, line, area and volume objects can all be transformed using the same hardware, more or less in the same way. Pipelining this operation, by first passing a scene description through a matrix multiplier to transform it, then passing it through a hidden area removal and illumination processor, gives a system which is capable of producing a real-time moving display if the second stage can be made fast enough.

If a plane (a, b, c, d) in the object space gives a new plane in the display space (A, B, C, D), then the depth value Z can be calculated at any pixel position (X,Y) by the equation:

$$Z = (A.X + B.Y + D)/(-C)$$

This distance is perpendicular to the display screen in the display space, and can be used to represent the plane in visibility tests at the given pixel position. An alternative distance which is important in several algorithms is the distance K from the pixel position to the plane, taken perpendicular to the plane itself.

$$K = (A.X + B.Y + D)$$

Working with a raster display it is possible to simplify the calculations of K and Z for each pixel, by calculating an initial depth value at one pixel position, and then calculating the size of the increments needed to modify this value, moving in regularly spaced steps over the display surface.

If the display transformation is set up so that the display space has its origin at the origin of the object space with the x and y axes corresponding to the X and Y axes and the direction of viewing being along the z axis, then the following values can be calculated:

$$Z = D/(-C)$$

$$\Delta Zx = A.\Delta X/(-C)$$

$$\Delta Zy = B.\Delta Y/(-C)$$

where ΔX and ΔY are the pixel spacing dimensions, or the screen width increments -- depending on the next stage. Where $C \rightarrow 0$ then there is a problem. This situation corresponds to the plane surface getting closer and closer to being perpendicular to the display screen. The perpendicular and near perpendicular plane can be processed in a different way, by using the value of K rather than Z. This value and its increments will already be available:

$$K = D$$

$$\Delta Kx = A.\Delta X$$

$$\Delta Ky = B.\Delta Y$$

$$\Delta Kz = C.\Delta Z$$

in other words the intermediate result, before carrying out the division operation. The critical situation where the value of K replaces the value of Z will occur when either of the pixel level increments of Z exceeds 1.0. This corresponds to a surface which is not visible at one pixel position but has crossed the display screen to give a cross section cut at a neighbouring pixel position. Although there are a variety of ways in which this set of operations can be carried out, a simple pipelined arrangement has been evolved where an approximation method for calculating the division is employed to make the test for perpendicularity easy to implement.

The division by multiplication is carried out in the following way. Consider D/C. Normalise C to be a fraction of the form 0.xxxx and consider it to be the value $(1-\Delta)$. Both D and C are then multiplied by $(1+\Delta)$ which is $(2-(1-\Delta))$, in other words 2-C.

$$\frac{D}{C} = \frac{D}{(1-\Delta)} = \frac{D.(1+\Delta)}{(1+\Delta).(1-\Delta)} = \frac{D1}{C1}$$

$C1 = (1-\Delta^2)$ which permits the process to be repeated:

$$(1+\Delta^2) = 2-C1$$

$$\frac{D1}{C1} = \frac{D1.(1+\Delta^2)}{(1-\Delta^2).(1+\Delta^2)} = \frac{D2}{C2}$$

$C2 = (1-\Delta^4)$.

This process is continued until the error term is below some fixed value. At this point

$$\frac{D}{C} + \frac{-Z}{1.0} = -Z$$

Using this scheme, it has been estimated that 4 or 5 multiplies in series will give the required accuracy for the division operation. If multiplication time is 100n secs then 500n secs are required for each plane transformation. This will allow 80,000 planes to be

transformed in the frame time of 1/25 sec. If the transformation time can be reduced to 400n secs then this figure goes up to 100,000 planes in a frame time.

Moving Images and Transformations

The selection of the front and back surfaces of a convex object at a particular pixel position can be achieved in one pass through a display list -- a list of plane half-spaces linked by intersection operators: A.B.C.D.E.F. It is necessary to hold two temporary values in registers during this process: the first holding the furthest away front surface encountered up to that point, the second holding the nearest back surface. In order to carry out this process it is necessary to know which are front surfaces and which are back surfaces. Clearly this is a classification task which has to be carried out after each plane transformation, if a moving scene is being generated.

Consider a plane defined relative to the origin by the value K. This distance will be positive where the origin lies inside the plane and negative where it lies outside:

$K = a.x + b.y + c.z + d$

$K = d \qquad (x=0,\ y=0,\ z=0)$

The sign of d can consequently be used to indicate the inside-outside relationship between a plane and the origin. Backness or frontness depends on the intercept of the plane on the Z axis, combined with this inside-outside value. From the diagrams in Fig. 7 it can be seen that the exclusive OR of the sign bits of K and Z defined at the origin gives a simple and fast way of maintaining this classification of frontness or backness.

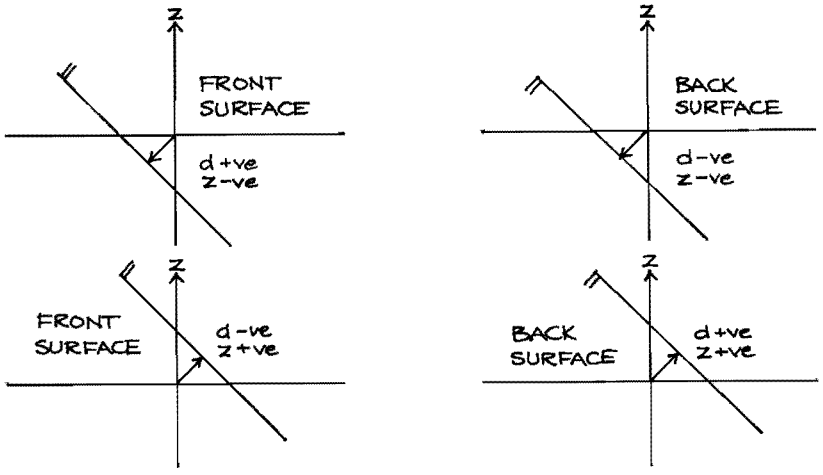

Fig. 7 Front or Back Planes

If display lists for convex units are pre-ordered so that all front surfaces are received for processing before the back surfaces in the same convex group, then only one temporary value needs to be stored carrying out the display algorithm. However this requires a

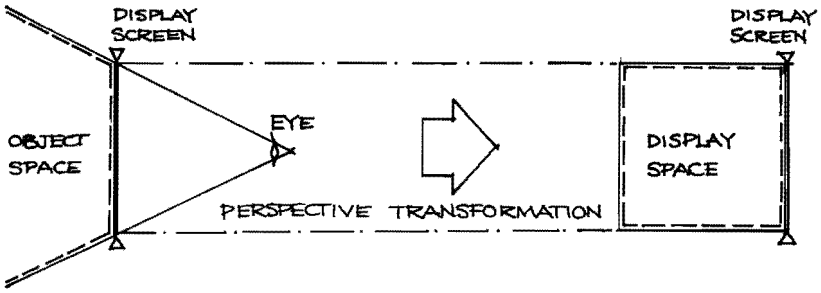

Fig. 8 Object Space to Image Space Transformation

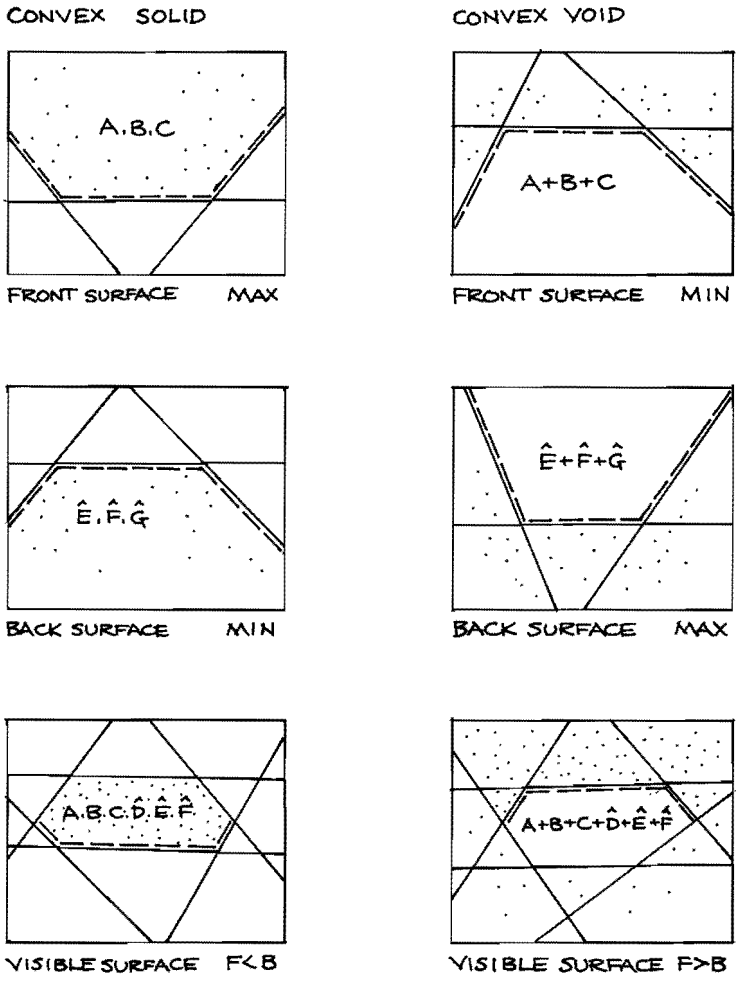

Fig. 9 Display Operations: Convex Units

reordering of the elements in the display list after transformation. If the length of each convex unit is held with each display list then a simple addressing operation can be used to achieve this reordering, where outputting the values to the hidden area removal processing unit, from the transformation unit.

Figure 8 summarises the transformation operation implemented in the way described above. The result is a display space which can be thought of as a cube lying behind the display screen. Objects in this display space can be displayed using a simple parallel projection operation. The display algorithms used for Boolean expression models in this context are shown in Fig. 9 for both convex solids and convex voids.

HIDDEN AREA REMOVAL HARDWARE

Given the ability to calculate depth values using a transformation unit similar to that used for point based models, hardware to implement the selection operations summarised in Fig. 9 had to be developed for hidden area removal. It was clearly possible to implement a Watkins' (1970) style scan-line algorithm. An advantage which resulted from having surfaces spanning the whole display space was that binary sub-division, instead of being applied to each polygon facet section, could in the manner of Warnock's algorithm (1969) be applied to the whole display space. Each scan line could be sub-divided using a common framework for all surface-surface comparisons.

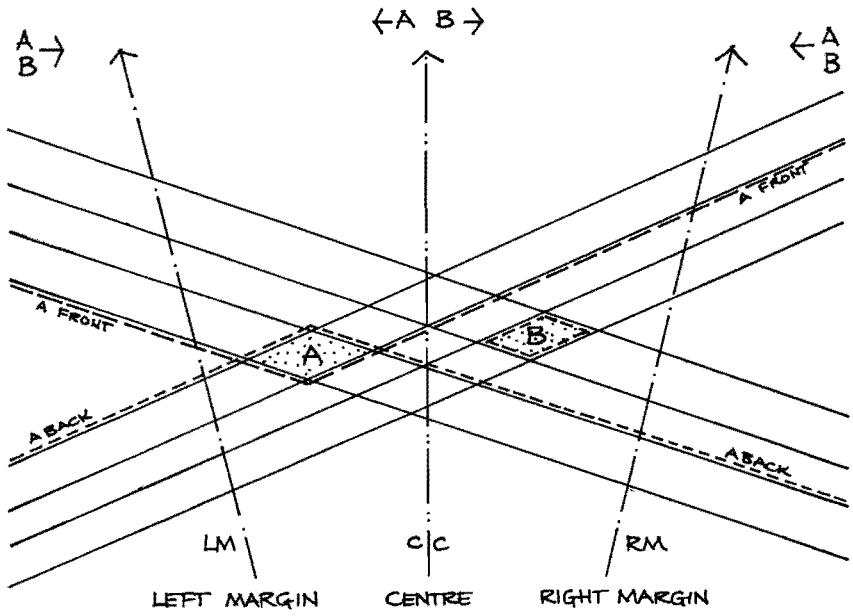

Fig. 10 Binary Sub-division of a Raster Line Cross Section Plane

Consider the two volumes A and B shown in cross section in Fig. 10. If the front surfaces and back surfaces of these two volumes are determined at the margin positions LM and RM, then their relative positions and orientations will indicate in which direction the

objects lie. In this case both A and B exist to the right of the left margin and to the left of the right margin. When the same tests are applied at the centre point, the results show that A lies to the left but B lies to the right. A recursive sub-division procedure based on these tests will sort objects into order along the raster line and considerably reduce the work in finding visible sections of objects' surfaces.

It appeared that if reasonably simple scenes were being processed then this approach would be adequate. All that was needed to obtain a display was a video-rate interpolator to fill in the pixels between the facet edges determined by the sub-division algorithm. However where an image complexity was required which might demand changes every few pixels, this sub-division process could not be carried out fast enough to support real-time without some form of parallel hardware implementation. At the time this seemed too difficult. What appeared much easier to implement was the simple hidden area algorithm of Fig. 9, in an iterative form using repeating hardware units, and then to use it in partnership with a software sub-division or other scene coherence algorithm. It now seems fairly clear that to obtain the speed and complexity desired will need both stages, both implemented in hardware, the overall system structure being extended to that shown in Fig. 11.

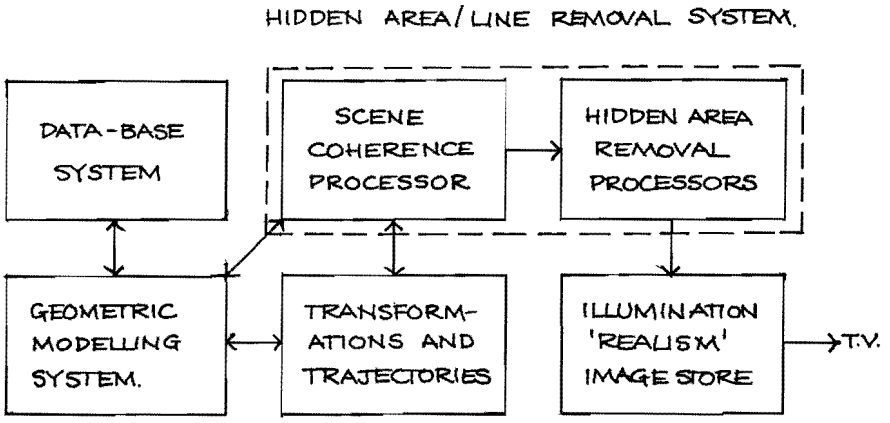

Fig. 11 Display System Block Diagram

Parallel Schemes

There were two stages to the simple iterative implementation of the algorithm which were:

1. Evaluating depth values at each pixel position for each plane half-space in the display list.

2. Comparing depth values to select visible surfaces, based on inside-outside or viewing ray priority tests.

It was found possible to construct two different parallel schemes. In the first, depth generating units from parallel processors were combined into a single entity. In the second the comparison and

selection units were combined together as a single unit. Each of these combined units could be implemented using either pipelined (systolic) processing or simultaneous (synchronous) processing. This gave four general schemes. The two main parallel implementations are shown in Fig. 12. The labelled sub-systems: A, B, C, D, E and F are shown in Figs. 13 to 18.

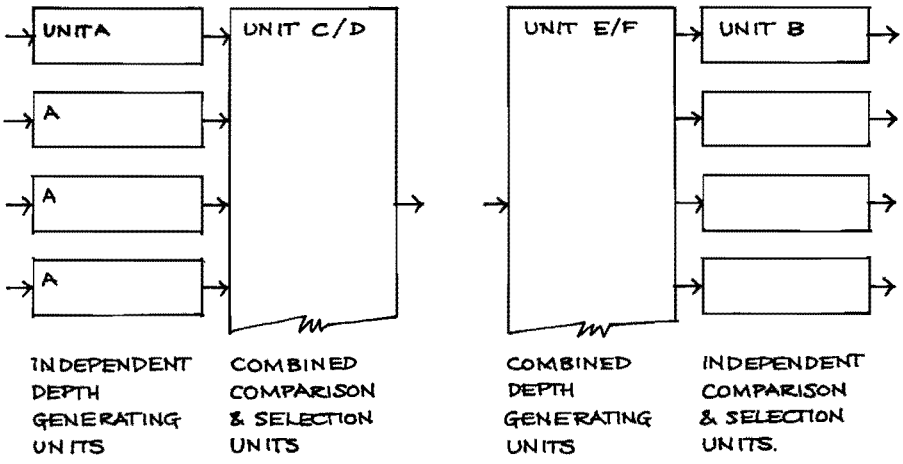

Fig. 12 Parallel Processing Schemes

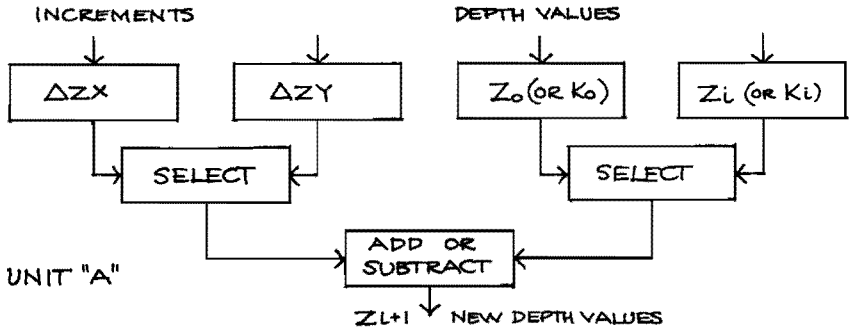

Fig. 13 Independent Depth Generating Unit

The independent depth generating unit can be a microprocessor. However, the minimum arrangement for modelling a plane surface is given in Fig. 13. The minimum unit can be combined in two ways to create a stream of depth values for neighbouring pixel positions. The first is a systolic solution, using a pipeline, the second is a synchronous solution, in this case using a binary, quad or oct tree of incrementing units in the way shown. The advantage that independent units have over the combined units is that they can be implemented using general purpose microprocessors: as long as video rate output is not required this arrangement appears to provide the simplest scheme for exploring the modelling of curved surfaces.

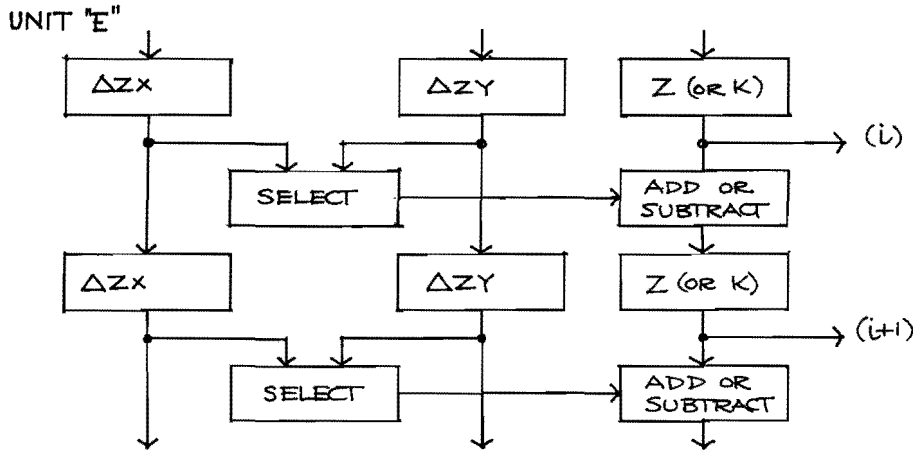

Fig. 14 Pipelined Incrementing Unit

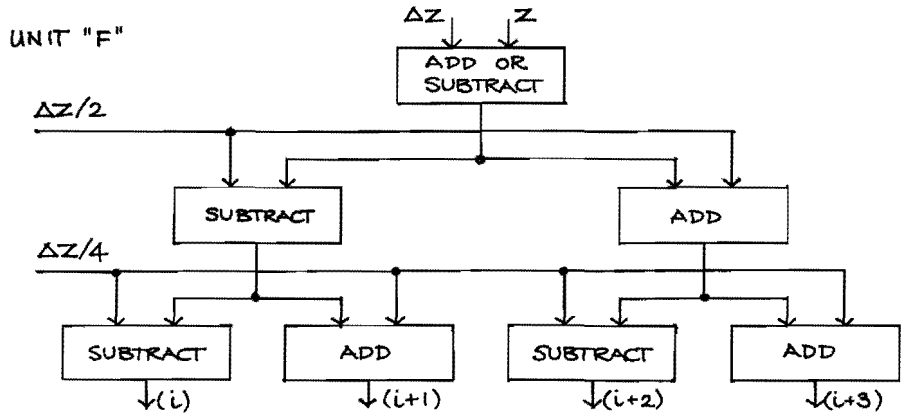

Fig. 15 Synchronous Incrementing Unit

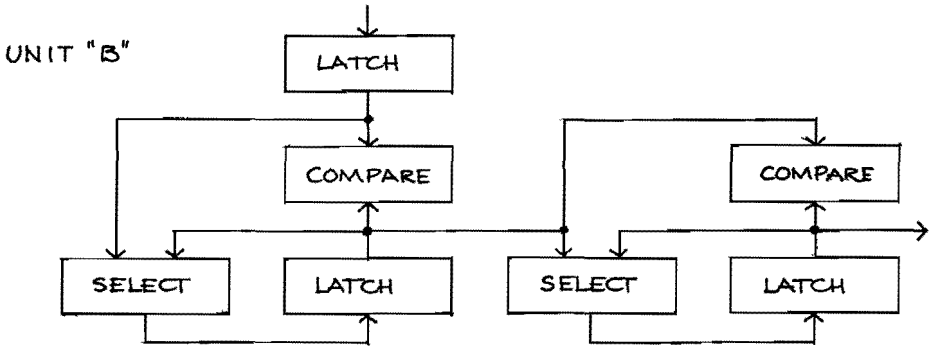

Fig. 16 Independent Comparison and Selection Unit

The combined depth generating units are shown in Figs. 14 and 15. Figure 16 shows an independent comparison and selection unit. Figures 17 and 18 show the two combined units. In unit C depth values are passed from unit to unit in a pipelined system, in unit D values are compared with a broadcast value on a bus. An alternative to D is a tree structure made up from comparison units. In the example shown a special bus driver was designed to maintain the maximum value on the bus from all the values being driven onto it, and to flag the units where this maximum value originated.

These comparison and selection units were initially designed in two stages corresponding to the product level and summation level of a simple Boolean expression.

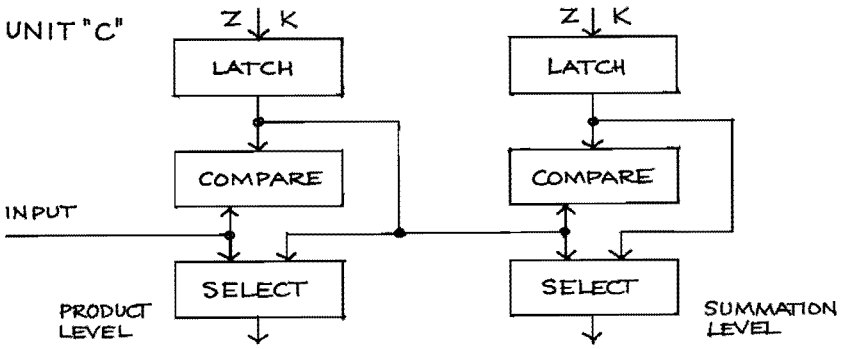

Fig. 17 Pipelined Comparison and Selection Unit

However, the structure of unit D suggested a way of implementing an array processor which would handle multiple level Boolean expression models directly. This approach also linked to a way of processing quadric surfaces.

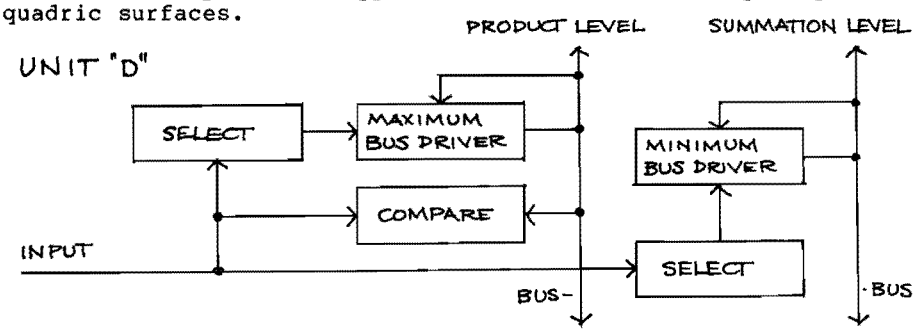

Fig. 18 Synchronous Comparison and Selection Unit

Quadric Surfaces, Convex Objects and Front and Back Pairs

A quadric surface can cut a viewing ray twice, and if it does it will intersect the ray with a front and back surface in much the same way that a product phrase made up from plane facets does. This means that the surface can be considered to be two variable surfaces called F for front and B for back. The hierarchical structure of a Boolean model allows a parallel scheme of the kind shown in Fig. 19 to be constructed, which is capable of generating video rate output.

In this arrangement, each of the comparison and selection units compares two sets of front and back depth values. Values passing from left to right in the array are either passed through or replaced by values on the vertical lines, depending on their relative location in the Boolean expression hierarchy and their relative values. This system generates the set of viewing ray spans which define where the viewing ray pierces solid objects for a particular line of sight. The final stage in the array represents a sorting bus or a sorting pipeline of the form described above, which produces the final visible surface.

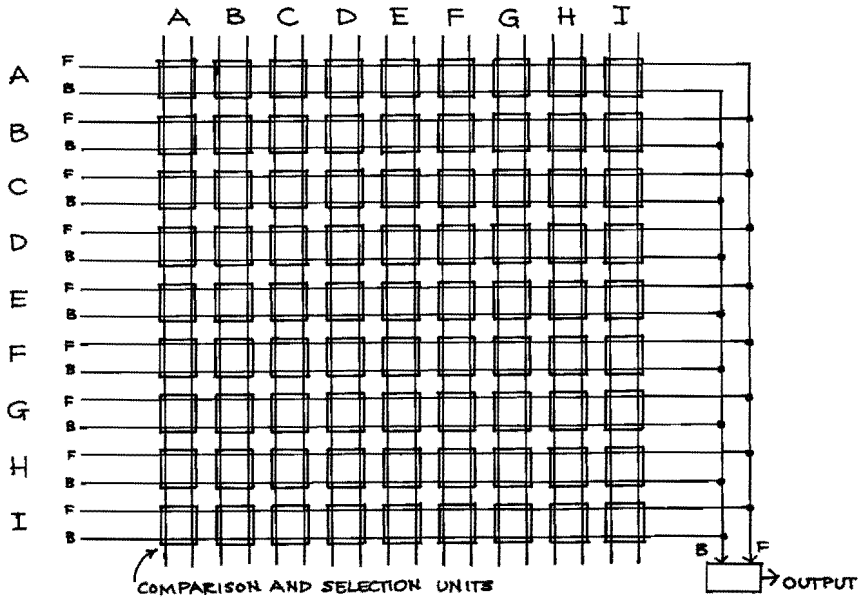

Fig. 19 Comparison and Selection Array Processor

A general real-time solution to this problem requires a square array of n.n comparison and selection units, because the worst case expression of the form (A(B(C(D(E(F(G(H(I)))))))))) requires (n^2) comparisons. The array processor can be implemented using either a synchronous or a systolic sorting process. In the latter case the data wave front has to flow diagonally across the array to maintain a continuous video refresh-rate output of pixel values. The problems arise with this system when the worst case does not occur -- many of the comparison and selection units become redundant.

A solution to this problem was provided by the boundary expansion. If the first form of the boundary expansion is applied to an expression model containing n convex units then a display list of n^2 convex units is created. This modified display list can be processed by a linear array of processors designed to handle boundary expressions. Such a processor is almost identical to the two stage processor already described. The primary data path for depth values is the same. Only the control is a little more complex having to contain a one bit wide push down stack to hold

intermediate inside-outside test values. This approach to the display problem seemed more versatile because given a pipe of n^2 processors it was possible to process a fully nested expression containing n convex units, at one extreme, and a simple expression of n^2 elements long at the other: both in real-time, both in the same processor. There was no complicated mapping task setting up an array in an optimal fashion, in order to use all its components for simple models.

The Boundary Expansion and Clipping Volumes

The boundary expansion of the model shown in Fig. 2 was:

$$@(A.(B+(C.(D)))) = @A.[B+(C.(D))] + @B.[A.(\overline{C}+(\overline{D}))]$$

$$+ @C.[A.\overline{B}.(D)] + @D.[A.\overline{B}.C]$$

Each phrase in the expansion consists of a convex element, nominally a surface, followed by a volume definition inside a pair of square brackets. The phrase represents the section of the surface which lies inside the volume. In the expression shown there are four pieces of surface making up the boundary of the total object. The visible section of each convex surface can be determined in the same way used to display simple convex solids. Since these convex boundary pieces can be voids, in other words concave surfaces, this process has to be extended in the way summarised in Fig. 9 to include convex voids.

The visible surface at each pixel position will be represented by a distance along the viewing ray. It is possible to test this distance against the front and back surface distances of the convex units making up the clipping volume in the square brackets. The inside-outside results of these tests can be combined using the Boolean operators in the clipping expression, to determine whether the surface point is within the clipping volume and should be retained or whether it is outside and should be discarded. If the convex surface is processed first and its visible surface distance is held in the first stage register of the comparison and selection unit, and the half-spaces in the clipping expression are passed through the input register of the same unit, then it is possible to implement the clipping operation using a one bit wide push-down stack to hold intermediate inside-outside results. The operation is cartooned in Fig. 20 for the simple object A+B.

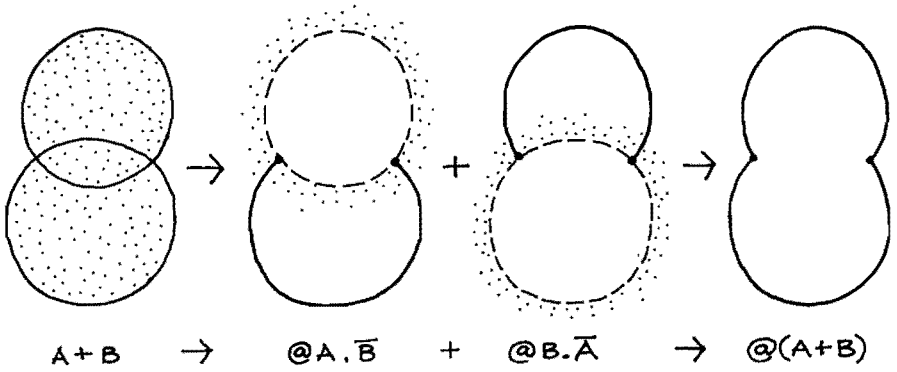

Fig. 20 Boundary Operator

A Prototype Display Processor

Of the four schemes outlined above the two processors with combined pipelined units appeared the easiest to implement in an efficient way, using the minimum of hardware. The one being investigated in current work is the one with a pipelined, combined depth generating unit. In this system depth values are established for a reference pixel point for all the plane half-spaces in a scene model. These depth values are then passed to the first comparison and selection unit, on one hand, and through an incrementing unit to the next procesor on the other. In this way the depth values received by the second processor represent the planes at the next pixel position, so that the identical process can be repeated by each processor in the line, the difference being that the values operated on by neighbouring processors are for neighbouring pixel positions.

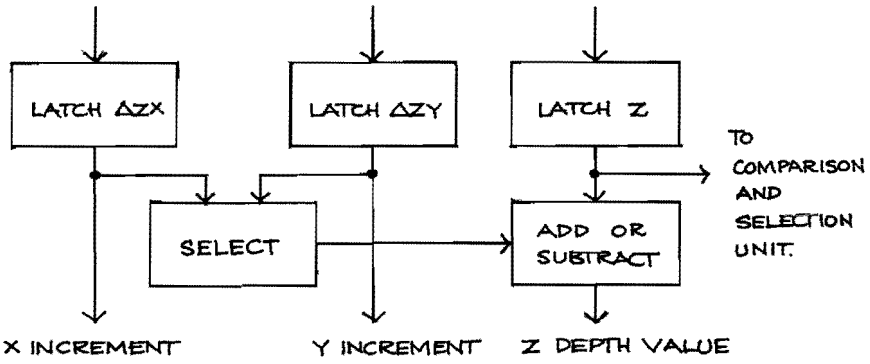

Fig. 21 Incrementing Stage

Block diagrams of a processor which can be used in a pipeline of this kind are given in Figs. 21, 22, and 23. If the arrangement in Fig. 21 allows increments to be subtracted as well as added, then these units can be made to follow any linear sequence of steps on the display screen, as an alternative to following a regular raster pattern. In Fig. 22 the results of the comparisons I and J are combined with control data to give the select signals P, Q and the output enable signal R.

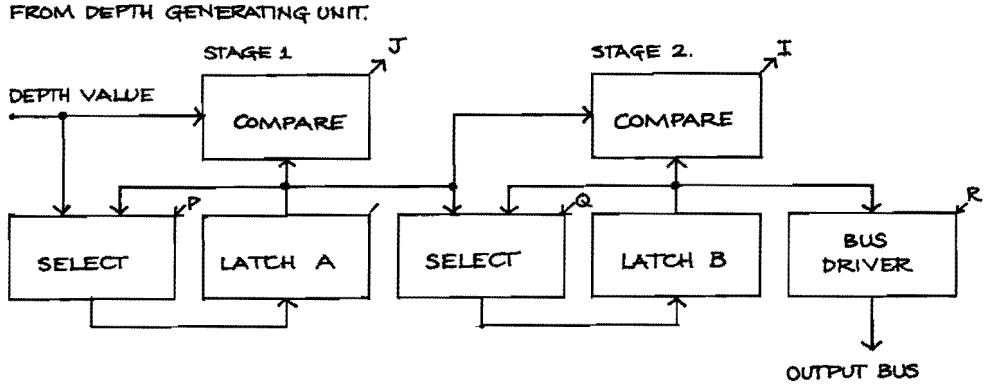

Fig. 22 Comparison and Selection Stage

There are two comparison and selection stages. In the first one the visible front surface facet for each convex volume is selected, for a pixel position, by retaining the front surface with either the maximum depth value for solids or minimum depth value for voids. These front surface values are then clipped either by back surface values or by an explicit clipping expression. The second comparison stage selects the minimum visible front surface value from the

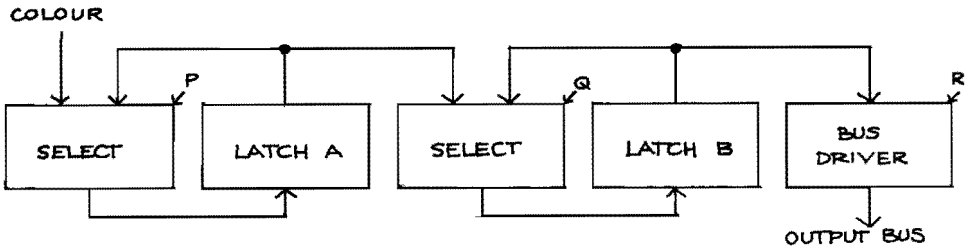

Fig. 23 Property Value Selection Unit

values passed from the first stage. It is necessary to select the depth value of the visible surface, but it is also necessary to select the name or some property of the visible surface for later reference. This is done using a slave selection unit controlled by the same signals P, Q and R shown in Fig. 22, this property selection unit being shown in Fig. 23. Outputs are driven onto a bus. Since only one unit contains the end of a display list there is no contention on the bus, as the last element in the list triggers the output.

INTEGRATED CIRCUITS

A prototype system to demonstrate this display system was built using eight processors constructed from TTL components. Although the prototype display system was successful, the approach was only going to be economically practical if the processors could be implemented as integrated circuits. To do this it was necessary to reduce the number of inter-processor links. Of the four parallel schemes outlined above, two permitted this reduction in a reasonably simple way: the tree structured incrementing scheme, and the pipelined incrementing scheme. The number of bits necessary for the comparison and selection stage ruled out both forms of combination into a single unit. Both the combined incrementing units could be rearranged to process their values serially, while outputs were still provided in parallel at the clock rate. The first system to be examined from this point of view was the tree structured processor. However the eventual desire to implement a scheme where increments were also incremented to give curved surfaces seemed easier to accomplish with the second arrangement.

Once the systolic method of processing had been established, there were a number of ways in which the internal processing of each unit could be arranged. The objective was to reduce the clock cycle time to as short a period as the technology and the architecture would allow.

The simplest scheme consists of a one bit wide input stream for depth data values, in the way illustrated in Fig. 24. The shift registers are shown schematically to be four units long. The only drawback of this arrangement was that a pipeline of processors had to be over thirty of these units long to hold the depth value for a single plane. An alternative solution resulted from increasing

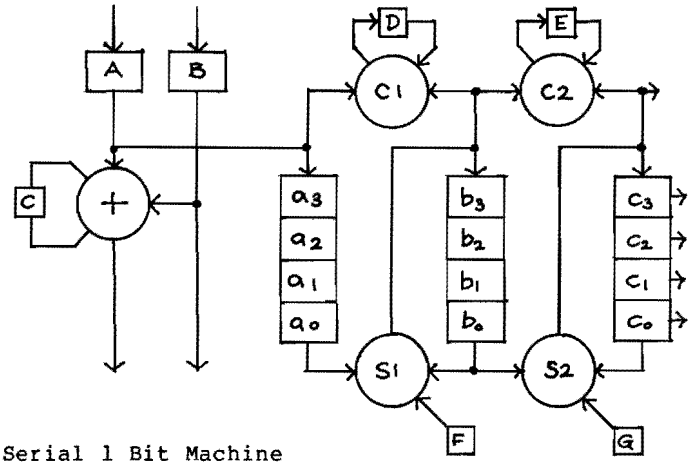

Fig. 24 Serial 1 Bit Machine

the width of the incrementing unit. In this case the addition (+) and the two comparisons (C) require a carry to be included in the operation which has to be completed each clock cycle. The result is that half the number of processing units are required to hold the value of one plane in their incrementing unit latches. This means that half the number of pixels have to be processed per plane. The limit to this approach is set by the carry chain speed.

The layout of Fig. 25 suggested an alternative way of arranging the processing. If a latch were placed between each of the one bit

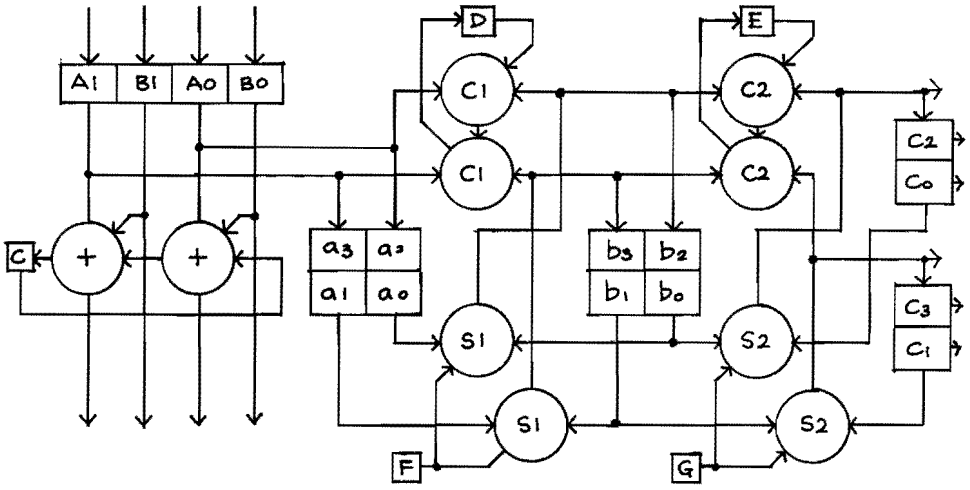

Fig. 25 Serial 2 Bit Machine

adders and one bit comparators, to make the operation truly systolic, then the delay time of the carry would be saved. The difficulty with this approach is illustrated if a map of the data passing down the pipeline of this kind of processor is made. Take as an example the completely parallel version of the schematic four bit system shown in Fig. 26.

At first sight this arrangement seemed to save components. The adding unit shown in Fig. 26 would take four cycles to process one value but it would be processing three other values at the same

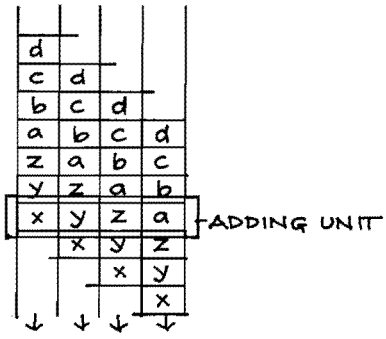

Fig. 26 Systolic Processing

time. It is possible to place a comparison either directly in parallel or one delay period after this addition. However, it is not possible to follow this by a selection operation, at least as far as a particular depth value is concerned. The whole four bit comparison has to be completed before the selection of the plane can be decided. It is therefore necessary to include delay elements in the data path in the way shown in Fig. 27 where an interesting similarity to the unit in Fig. 24 appears. The only differences are where carry bits are not looped round within the unit but are passed on to a neighbouring unit, and the depth values in each shift register are not from the same plane.

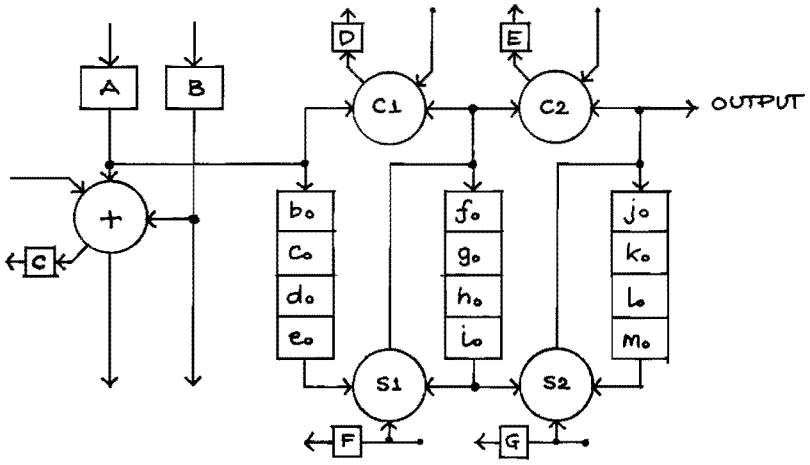

Fig. 27 Parallel Systolic Processing

Although there are many ways of combining these processors in parallel, there seem to be two principles which can be applied to produce an optimum IC. The first results in the use of the scheme shown in Fig. 24 rather than that shown in Fig. 27, because this partitioning of the system gives a unit which can be packed onto an IC in whatever number that the current technology will allow, without increasing the pin count for the IC. The equivalence between these two circuits is reflected in the way that both can be arranged to give a bit sliced implementation of the pipeline. The units from Fig. 24 have to be arranged in the way shown in Figure 28. By arranging the delays in the incrementing pipeline in the appropriate way and passing output to fast shift registers in the same way used in frame stores it is possible to reduce the pin counts to the same number that a scheme built up from the units in Fig. 27 would allow -- without a cumulative increase in carry pins as the IC's processor count is raised. The second principle is that doubling the data path for the arithmetic and comparison operations halves the necessary pipe length and reduces the ratio of memory cells to logic elements in each processor. On the other hand it increases the addition time, though not necessarily in a linear way.

There is consequently a design trade-off to be made depending on the nature of the implementation. Where register cells are relatively large, a wider data path appears justifiable; where they are smaller, less so. The advantage of the structure in Fig. 24, developed in 1975 for the tree processor in Fig. 15, was the way that shift registers from neighbouring processors could be stacked together as a block of memory.

Figure 29 shows a schematic floor plan for the ultimate IC. Because the shift registers are all the same length, the access point to an equivalent set of memory registers will be at the same place at the same instant in time. This makes it possible to use a cycling addressing scheme and a standard block of memory. The particular advantage that this has is that a 32 word wide unit can be provided, but the number of words cycled through can be software selected depending on the required resolution. Although one bit wide processing provides the shortest latch to latch delays, hence the fastest clock rates, the adoption of multiple pipelines makes this less critical for speed, and the arrangement shown in Fig. 29 will work with one or more bits-wide data-paths depending on the

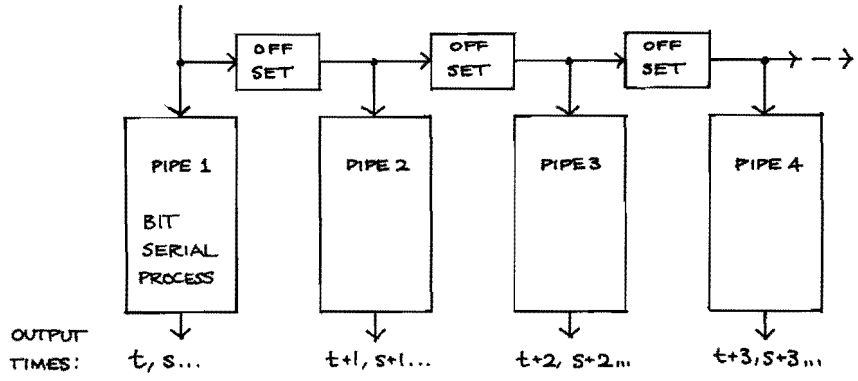

Fig. 28 Equivalent Bit Slice, Multiple Pipeline Processing

properties of the final design. One factor which it has been difficult to assess is how long a pipeline needs to be built to give the best system performance.

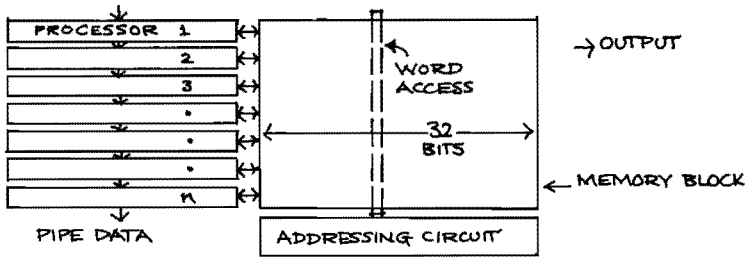

Fig. 29 IC Floor Plan

Pipeline Control and Priming

In concept, the simplest way of using this kind of machine in real-time is to have a pipeline long enough to hold the maximum length display list likely to be encountered. In practice the scene is subdivided until the display list is the same length as the pipe. However, while the primary display algorithms were being explored, the simple approach was adopted. If a technology could be found which reduced the size of these processors sufficiently, then this approach would give a very simple system useful in quite a variety of applications. With this objective in mind, first CCD technology (1975-6) was investigated, then n-mos (1976-84), and finally c-mos. However, it seems that the simple approach is still some way off. When it became clear that a complex scene would require a large number of integrated circuits, particularly if the processor was extended to handle curved surfaces, then it became important to consider what facilities were necessary to allow the pipeline to be used with a scene subdivision or other scene coherence pre-processor.

Consider a display list of 50 units long, and a pipe of 25 pixel processors in length. If the display screen is divided into four quadrants and the display list is subdivided to remove elements which do not feature in each quadrant, then four new display lists will result. If the new display areas require display lists of 20 elements, 4 elements, 15 elements and 17 elements respectively, then the display pipeline will be able to work in real-time. Each list can be padded out to give a list 25 units long, which can then be cycled through the pipe until the required number of pixel values have been generated for each of the display lists. The lists being swapped as each section of the display screen is completed in sequence.

Where complex scene models are being processed there may well be a need to sub-divide the screen down to the level of a few pixels, for part of the image. In this case the display list must be reduced to a length which is less than the number of pixels required, as well as less than the pipe length. This means that some way of managing several lists of less than the pipe length must be found, if continuous pixel rate output is to be maintained. There were two interrelated problems which had to be resolved to make this possible. Firstly, it was necessary to process variable length lists, and secondly it was necessary to prime the processor with new

incrementing directions each time a new list was entered. The simplest approach found so far is illustrated in Fig. 30. The first requirement is that only one list is generating output at any instant in time. A processor which is producing an output value can only do so once it has processed the whole display list. The output will always be generated, therefore, at the tail end of a list. The only way of maintaining continuous output is to allow each list to generate output, only when it is the last list in the line: the shaded lists in Fig. 30.

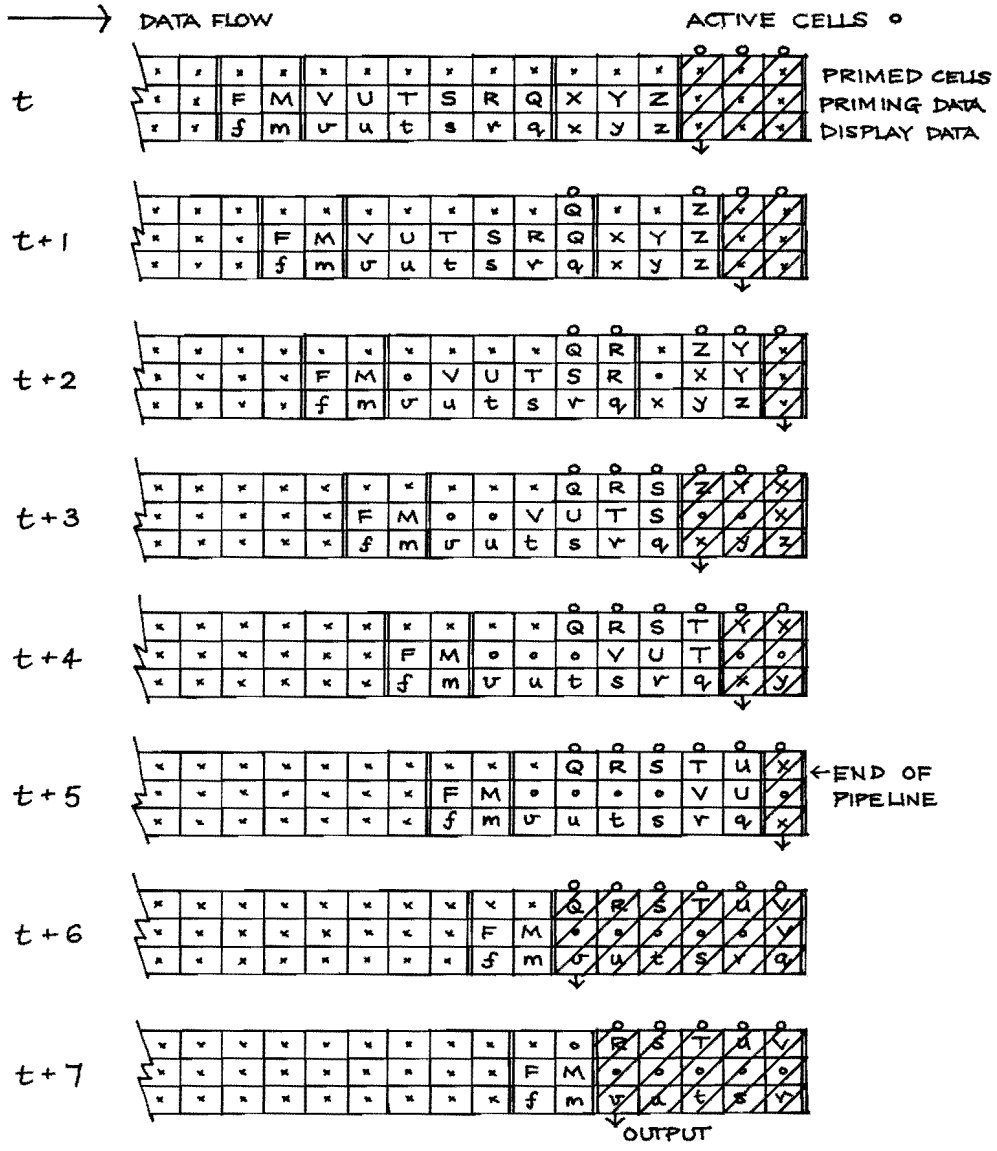

Fig. 30 Pipeline Control

In order for a list to be processed, the processors it is entering have to be primed with control data to determine the incrementing direction of each processor. This priming information is shown in capital letters in the diagram. At time t+1 the two lists (v,u,t,s,r,q) and (x,y,z) are each entering the processor which is their own list length from the end of the pipe, so evaluation has to start for each of these lists. In each of these cases the processors are primed with the incrementing control data: Q and Z respectively. In the next clock period the display list has moved one step down the pipeline. In order to keep up, the priming-data list has to be passed down the pipeline at twice the speed, once list processing starts. This ensures that the priming-data list reverses its direction in the processors in the way shown. This is the only scheme which has been found which allows multiple lists to be handled, and at the same time allows the priming information to be entered with the list it controls. Clearly before a display list reaches the "active" primed processors it has to be passed passively down the pipeline, the incrementing operation suspended, and the output instruction masked out.

It is possible to extend this priming operation. If partial results from a previous calculation are stored in a depth buffer they can be passed down the pipeline in this way to the appropriate processor where they can be used as the starting point for a subsequent calculation. It is this facility which will make it possible to use this kind of processor for real-time ray tracing algorithms.

CONCLUSIONS

There are clearly many applications for the kind of system outlined above. At the simplest level, instrumentation displays which require 3D symbolism can be driven by a minimum set of hardware. At a more complex level, the pilot-training simulator still appears to make the most stringent demands on a display system's hardware, though the requirements of CAD/CAM systems are developing fast. The design work station may well end up with the widest range of demands to be satisfied, some of which will justifiably require the real-time movement of objects, and the highest level of image realism that technology can provide. Because the approach which has been described is totaly general, it is possible to include many other options within its overall system structure - ranging from specialised curved surface processors to existing line and point based sub-systems (Sutherland, 1963) - without major changes to the system's architecture. This and the system's intrinsic modularity make it worthy of serious consideration as the underlying framework for future design work-station processors.

ACKNOWLEDGMENTS

Funding provided by Science & Engineering Research Council. Key developments helped by J. Downie, J. Woodwark, M. Sabin and Professor R. Forrest. Electronic circuit design and VLSI design help received from J. Mclean and M. Morant.

REFERENCES

Newell ME, Newell RG, Sancha TL (1972) A New Approach to the Shaded Picture Problem. Proceedings ACM National Conference.

Rougelot RS (1969) The General Electric Computed Colour Display. In Pertinent Concepts in Computer Graphics ed. M Fairman and J Nievergelt, University of Illinois.

Sutherland IE (1963) SKETCHPAD: A Man Machine Graphical Communication System. Spartan Books, Baltimore.

Thomas AL (April 1976) Spatial Models in Computer Based Information Systems. Ph.D. Thesis, University of Edinburgh.

Thomas AL (March 1986) Overlap Operations and Raster Graphics. Computer Graphics Forum, Vol. 5, 1.

Warnock JE (1969) Hidden Line Problem and the Use of Half-Tone Displays. In Pertinent Concepts in Computer Graphics ed. M Faiman and J Nievergelt, University of Illinois.

Watkins GS (June 1970) A Real Time Visible Surface Algorithm. Thesis Computer Science Department, UTECH-CSc-70-101, University of Utah.