Developments in High Performance CGI Systems

R. L. Grimsdale and P. F. Lister

Computer Graphics and VLSI Research Group
School of Engineering, University of Sussex, Brighton, UK.

INTRODUCTION

This contribution describes some work being undertaken in the design and implementation of architectures for high performance Computer Image Generation for a range of applications from workstations to flight simulator visual systems.

The work to be described uses a model based on a polygon representation and uses a Geometry Processor sub-system, with a flexible architecture known as MAGIC. This system performs the transformation of the polygon from the 3-D representation to the 2-D perspective projection to the viewing screen co-ordinates and also provides a clipping operation optional in 3-D or 2-D. Two different types of scan conversion system are described, the first the *Zone Management Processor* uses the coherence inherent in the polygon and the second system based on a *Line Processor* uses coherence with spans.

MAGIC

An architecture for a Multiple Application Graphics Integrated Circuit (MAGIC) has been developed. The aim is to produce a VLSI geometry processor for use in a wide range of graphics applications, from personal computers through workstations to real-time flight simulator visual systems. Major requirements for this processor therefore are flexibility and performance.

Flexibility in MAGIC is achieved by means of a writeable microprogram store, so the precise functionality of MAGIC may be defined by the user, rather than the chip designer. For the purposes of flexibility, support of floating point numbers is considered essential, ideally to a well-accepted standard such as an IEEE format. However, VLSI implementations of fast floating point processors require relatively large areas of silicon, whilst the speed of smaller serial ALUs is a limitation for high performance applications. These problems have been reconciled by designing MAGIC to be a controller for geometric operations, and providing a fast floating point co-processor (AMD 1985, Analog Devices 1985) to perform numerical computation.

Real-time applications are likely to require a higher performance from the geometry system than that attainable with a single instance of MAGIC, even with the fastest floating point co-processors currently available. Parallel processing of geometric data is possible using a pipeline of processors, each computing one step of the transformation and passing the data on, as Clark (1982) has shown. However, the transformation sequence is then fixed in hardware, and the number of processors is related to the number of separate operations into which the transformation has been partitioned, irrespective of the performance required. Geometry systems based on MAGIC use a different approach to parallel processing. Each MAGIC processor handles a surface or object at a time, and performs the entire transformation from world co-ordinates to screen coordinates. Whilst a low end system may require only one MAGIC processor, a flight simulator visual system might employ several instances of MAGIC transforming different areas of the active database concurrently, passing their results on to the display system. This strategy therefore allows greater flexibility according to application requirements, and potentially a much higher performance.

MAGIC Operations

Transformation of objects from the database to the view screen requires a sequence of geometric operations which are essentially vector computations. All of the computations may be achieved using cartesian (3-element) co-ordinates, and this approach generally leads to a lower processing requirement than the use of homogenous (4-element) vectors. Vertex points may have an associated intensity value for Gouraud (1971) shading, which must be correctly interpolated during clipping, and may be modified for distance face as a part of the perspective projection.

The geometric operations used for transformation of polygonal surfaces onto the screen are listed below:

- Back-facing surface removal; computation of a dot product plus a constant to test the surface normal against the observer's position.

- Transformation to eye co-ordinates (or parallel projection); a matrix transformation which may be implemented as three dot product plus constant computations.

- Perspective projection into screen co-ordinates and 'proximity', described below.

- Clipping in 2-D and/or 3-D; computation of a distance interpolant, and interpolation along a polygon edge. The Sutherland-Hodgman (1974) clipping algorithm is used to clip a polygon at a time to each clipping boundary.

Although only operations for the projection of planar polygons have been mentioned, the versatility afforded by use of a writeable microprogram store may allow MAGIC to be equally appropriate to other graphics systems, such as those based on ray tracing.

Generation of proximity in the perspective projection

A point is transformed to eye co-ordinates (x, y, z) by a view matrix which has scaling factors for the view screen incorporated in it. The perspective projection to screen co-ordinates (x_s, y_s) may then be achieved by the following calculations:

$$p = 1/z \quad \text{where} \quad p \text{ denotes proximity}$$

$$x_s = x_{cen} + (p * x)$$

$$y_s = y_{cen} + (p * y)$$

where (x_{cen}, y_{cen}), are the co-ordinates of the centre of the screen. The proximity p is the inverse of the distance of the point from the plane of the eye, as shown in Fig. 1. It will be noted that although pixels are linearly spaced across the screen, the corresponding visible points are non-linearly spaced across the planar surface in a perspective projection. Depth values at intermediate positions cannot therefore be linearly interpolated from endpoint values of z; hence for the purposes of 2-D clipping and z-buffer comparisons, some other technique must be used.

Fortunately, it can be shown that the proximities of visible points across a perspective projection of a planar surface can be linearly interpolated. It is much more convenient to implement a hidden surface algorithm based on proximity comparisons in the display processor, since the proximities at each pixel may be generated incrementally. Thus, for example, a proximity buffer

(p-buffer) algorithm may be used in place of a z-buffer; the surface with the highest proximity is the visible surface at each point. When performing the 2-D clip to edges of the screen, proximity values may be interpolated in exactly the same way as x_s and y_s (and Gouraud intensities). The proximity value generated as a part of the perspective projection can therefore be of great use in the display processor.

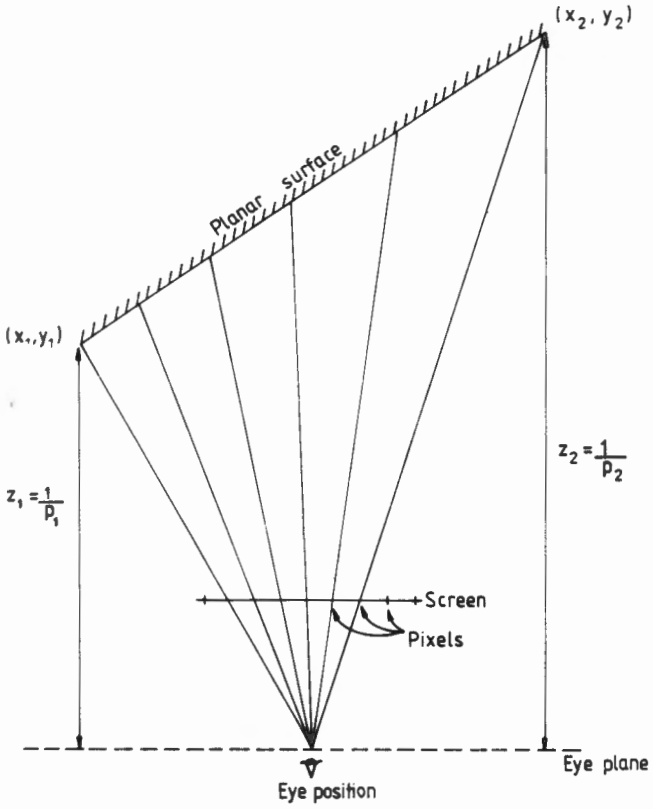

Figure 1 The incremental property of proximity

MAGIC Architecture

The proposed architecture of MAGIC, with its associated memory and floating-point co-processor, is shown in Fig. 2. The principle components are the writeable microprogram store, operand address counters (OACs), and pipeline registers for optimal transfer of data between the memory and the ALU.

Memory Map

The data memory on which MAGIC operates is of the same width as the word length used for floating-point representation - typically 32 bits. A real-time geometry system for transformation of planar polygons has the memory organised in 3 sections, shown below the MAGIC chip in Fig. 2. The first of second is the screen and view parameter memory, also used as workspace

these is the active database or part thereof (i.e. untransformed polygons). The second is the screen and view parameter memory, also used as workspace during transformations, and the third holds the transformed data from which the image is generated. In a typical transformation sequence, MAGIC tests each surface for being forward-facing, and if so, performs the parallel projection on the untransformed data, storing the results in workspace. All subsequent transformations take place in this workspace area until the resulting polygon is written into the transformed data memory.

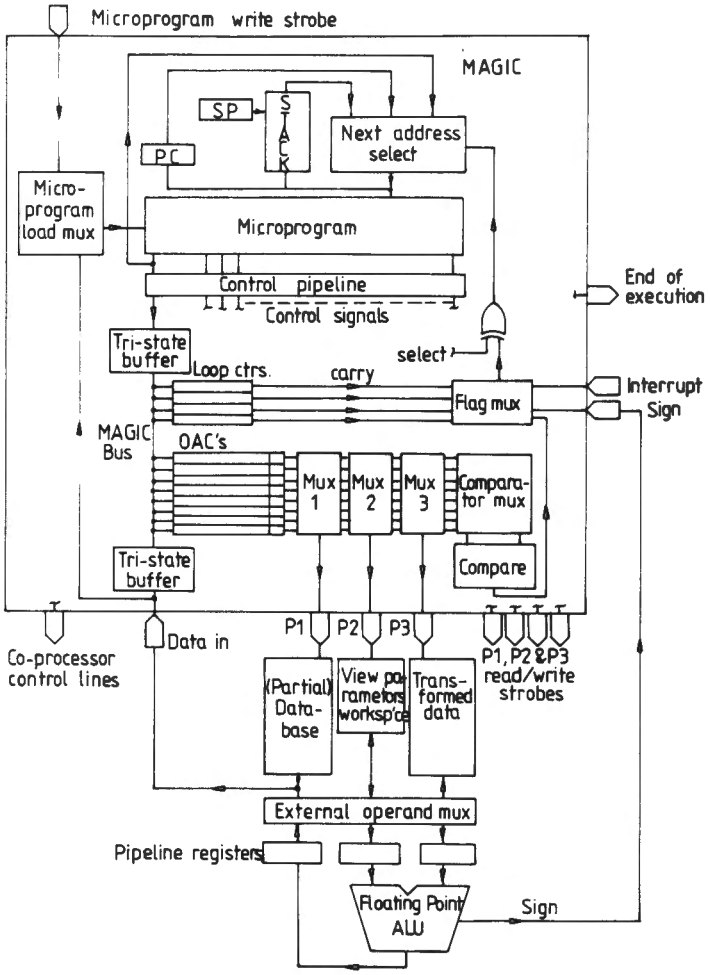

Figure 2 MAGIC Architecture

By contrast, a non-real-time application might have all three data areas implemented as one contiguous memory, so that only one address bus is needed, but multiple memory accesses are required for each floating-point operation.

In each memory section, the data is arranged so that all vector quantities, such

as vertices and rows of the view matrix, occur on 4 word boundaries. This allows a very simple technique of address generation to be employed, which will be discussed below.

Operand Address Counters

Each operand address counter (OAC) comprises a vertex address counter (VAC) and an element address counter (EAC) as shown in Fig. 3.

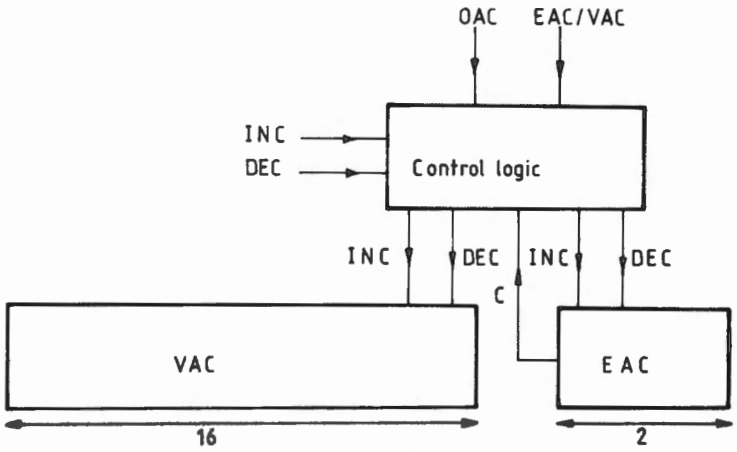

Figure 3 Operand Address Counter

In normal mode, the OAC is enabled as an 18-bit counter so that, on incrementation or decrementation, an overflow from the EAC will cause the VAC to increment or decrement. However, if this overflow is disabled, the EAC can be incremented to allow the addition of 4. Decrementation modulo 4 and subtraction of 4 follow similarly.

With vertices of 4 elements on 4 word boundaries, the VAC specifies which vertex is currently being examined, and the EAC points to the required element within that vertex block. This allows convenient and rapid generation of operand addresses without the need for an address ALU. The OAC block is a regular hardware structure, which with microprogram control supports a high level of concurrency in address generation.

Pipeline Registers

Ideally, the workspace RAM used by MAGIC might be a 3-port structure, so that two operands could be fetched while the result of the previous computation was being written back. However, this is not necessary if three memory accesses may be made within one floating-point ALU cycle, so the excessive interconnection problems of a 3-port memory may be eliminated by use of a high-speed RAM. The purpose of the pipeline registers is therefore to allow the two operands for the next floating point computation to be pre-fetched and the previous result to be written back, all within the time of one ALU cycle. In the real-time implementation with three separate memory blocks (Fig. 2), only the small view parameter/workspace memory need satisfy this special speed requirement, since each of the others are only required to be accessed once during each floating-point operation.

Input/Output and Control Signals

The main functions of MAGIC are to supply the three operand addresses and the control signals for the external multiplexer and floating-point co-processor. A *data in* port is required so that MAGIC may read either the number of vertices or the last vertex address for the current polygon from the database; this port is also used to load microprogram into the writeable store. The sign of some floating-point results is required for conditional operations such as back-facing surface removal and clipping, and a pin is therefore allocated for this purpose.

Once MAGIC has completed its transformation sequence on all the data supplied, the *end of execution* signal is given, indicating that the memory may be accessed by other processors in the system. The interrupt must be polled at the microprogram level so that execution may be aborted without generating corrupt output data. This may be required, for example, when a geometric signal indicates the start of a new frame, so the transformation sequence is re-commenced on the new data.

Other Features

Much of the microcode for geometric transformations may be eliminated if subroutining is available, thus a stack is essential. Loop counters are similarly required, since many of the operations are of a repetitive nature. Tri-state units are used to isolate the OAC bus from the microcode lines and *data in* port when immediate addresses or data are not being loaded. This permits transfer of contents between OACs, and between OACs and loop counters, using the internal bus.

When using the three-memory configuration as shown in Fig. 3, an external multiplexer is necessary so that the two operand sources and the result desination for each floating-point operation can be specified. Ideally, this multiplexer should form a part of the co-processor used with MAGIC, together with the pipeline registers and floating-point ALU. However, the multiplexer is superfluous when using the single memory configuration, since the sources and destination of data for each operation are then loaded.

THE ZONE MANAGEMENT PROCESSOR

Scan conversion is an important aspect of the production of raster scan displays of perspective views from a viewpoint moving in real time. This is a process in which each pixel of the raster display is painted to the appropriate colour value.

As previously noted the model is constructed from a set of planar surfaces or polygons defined with respect to a three-dimensional set of world co-ordinates. In order to achieve the necessary picture update rate of up to 60 frames/sec it is necessary to introduce some degree of parallel processing.

There are a number of different possible approaches. In Pixel Planes (Fuchs 1982) there is a notional allocation of one processor to each pixel. If indeed there was an actual allocation of one processor to each pixel the whole frame time would be available for processing each pixel. The system operates effectively by sending every polygon to each processor simultaneously; the polygons being specified in terms of their three-dimensional representation. The polygons are presented in an order which takes no account of their distance from the viewpoint. A number of processing operations are performed. The first is the rejection of polygons facing away from the viewer. Next, the distance of the polygon, at the particular pixel determined.

As successive polygons are processed, a nearer surface replaces a more distant surface which had been previously recorded as the nearest at that pixel.

An alternative system, described below, employs up to one processor per scan line. This requires that the perspective transformation of polygons into screen coordinates has been performed. This is followed by the creation of an edge table which defines the intersection or span of each polygon by the scan lines.

The third alternative, which is now described, is a technique in which there is a notional allocation of a processor to each polygon. In the basic form of the scheme, the polygons are transformed from their three-dimensional representation to the perspective projection in screen coordinates. Backward facing surfaces are rejected and visible surfaces are clipped and culled against the viewing cone.

The set of polygon processors or Zone Management Processors (Grimsdale 1979), or ZMPs, operate in parallel at pixel rate in synchronism with the raster scan display.

For a given scan line, the processor computes the start and end points of each span and the distance at each pixel from the viewpoint. There is a common bus to which all processors are attached. Using a mechanism, which is described below, a mutual resolution process is performed which determines the nearest polygon at each pixel and arranges that information about this polygon is sent for display (Price 1984). This resolution process operates at pixel rate, but the operation is pipelined and therefore introduces a lag of a few pixel times.

The special advantage of this technique is the exploitation of area coherence. The polygon is defined by the position of the apexes and the gradient of the edges. The computation of the start and end points of each span are computed from one scan line to the next by the addition or subtraction of the gradient increment. Similarly, incremental calculation can be used to determine the inverse of the distance of the viewpoint to the surface, a quantity known as the proximity.

Further properties of the polygon are computed on an incremental basis. Smooth shading is also performed by interpolation from pixel to pixel. Surface texturing can be performed using the proximity value as a parameter. The availability of this proximity value at each pixel permits distance fade to be computed accurately.

The problem of aliased edges can be minimised by computing the fractional contribution of the surface to the edge pixel. This technique which effectively increases the resolution of the system uses the information which is available within the ZMP about the whole polygon.

The priority or hidden surface resolution has been demonstrated in a particular implementation of the system, in which the distance values have been expressed as 16-bit integer values. The problem is to select the largest (or smallest) element of a set of integers in which the element values and number of elements present vary at successive pixel positions. The technique which has been employed is an iterative bit-wise comparison of all the priority values. Those which fail to reach the common maximum priority value for a given bit are removed from subsequent lower-order bit comparisons. The comparison is pipelined so that results occur on every clock cycle in the form of an acknowledge signal to the device with the highest priority value. The algorithm may be explained by considering four 4-bit binary numbers and assuming the larger the value the higher is the priority.

	2^3	2^2	2^1	2^0
first number	1	1	0	1
second number	1	1	0	0
third number	1	0	1	1
fourth number	0	1	1	1

It will be observed that all the numbers except the fourth have the bit with the value 2^3 set. The fourth number therefore fails to reach the maximum for that value and is thus eliminated.

	2^3	2^2	2^1	2^0
first number	1	1	0	1
second number	1	1	0	0
third number	1	0	1	1

The first, second and third numbers proceed to the comparison of the next most significant bit where the third number is found to fail and is removed from lower-order comparisons. The first and second numbers continue to the next stage for the third comparison where they are both found to be of the same priority. They both proceed, therefore to stage four where the first number is found to be the winner.

The implementation of this algorithm is performed at the hardware level in order to achieve pixel-rate comparison. On each ZMP there is a priority resolving circuit. This takes the form of a one-dimensional systolic array driving an open-collector bus. For a given ZMP each element of the systolic array performs a comparison between the currently ascribed priority value of that ZMP and the value asserted on the open-collector bus, for one particular bit of the priority word on every pixel clock cycle.

The number of ZMPs can be reduced below the number polygons in the scene by a careful allocation strategy. With the convention that the scan lines are horizontal, the vertical extent of each polygon is determined. To permit a ZMP to process more than one polygon per frame, an allocation strategy is adopted in which ZMPs are assigned to polygons in the order distance of the uppermost edge or apex from the top of the screen. Immediately a ZMP completes the processing of a polygon it is allocated to a further polygon. The number of ZMPs required therefore depends on the statistics of the number and distribution of polygons within typical scenes.

THE LINE PROCESSOR

The Scan Conversion Process to be described here, which is an alternative to the Zone Management Processor, creates an image in a frame store from a set of polygons transformed from 3-D to the 2-D perspective projection by a geometry processor. A large amount of processing is necessary to generate an effective and realistic image. As well as determining the position of each polygon on the display screen, processing is required to resolve occlusion of more distant polygons by those which are nearer; translucent polygons must be processed to allow partial visibility of more distant polygons; distance fading may be included to simulate mist and fog; the image can be enhanced by the inclusion of texture, not only to produce a realistic image but also to provide distance and velocity clues in a flight simulator visual system; polygons can be shaded to aid visualisation of curved surfaces and finally anti-aliasing is necessary to overcome the effects of spatial quantisation.

The various applications of CGI impose different requirements on the display device. A CAD workstation may not need the complication of a texture generator and a video game may not require translucent surfaces whereas a flight simulator would probably require most of the facilities. There is therefore a requirement for a modular system whose power can be enhanced to meet the more demanding applications whilst providing an economical solution for simpler usage. The system is accordingly based on a number of functional units, with the possibility of sharing each function between several processing units.

The architecture thus takes the form of a pipeline of parallel processors. More complex applications can be satisfied by increasing the number of processors in the pipeline.

System Configuration

In the following description it is assumed that the display scan lines are horizontal and scanning is from left to right and from top to bottom.

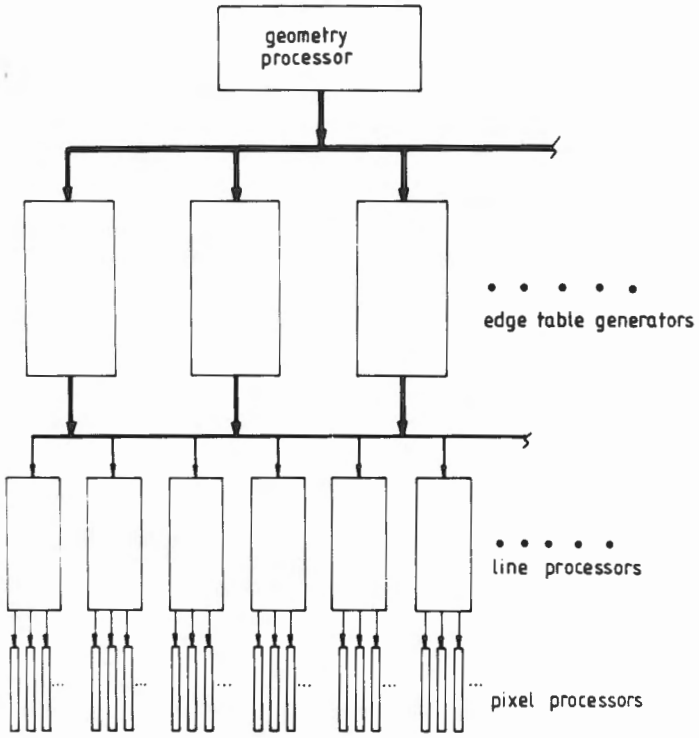

Figure 4 System Configuration

Coherence of the image in the polygon based display can be exploited to achieve efficient processing. In practice only small differences exist between adjacent areas of the display. The areas of consideration can be adjacent

pixels, spans of a polygon on successive scan lines, entire successive scan lines and successive frames.

Frame coherence, is not generally used in contemporary display systems since a large amount of information needs to be passed from one frame to the next. Span coherence is used in the ZMP system. Line and pixel coherence are used in the present system.

The system is divided into three main parts (Fig. 4), the edge table generator, the line processor and the pixel processor. The edge table generator associates polygons with scan lines. The line processor determines, for each scan line, the polygon that is present at each pixel, resolving occlusion and managing translucency where necessary. The pixel processor determines the actual colour of the pixels and adds visual effects, such as smooth shading and texture.

Edge Table Generation

The edge table processor uses line coherence. The processor commences operation when it has received information about all the polygons from the geometry processor. Each polygon is defined by its bounding edges. At the top of the polygon the positions of the intercepts of the edges on the scan line are computed, and on subsequent scan lines the positions of edges are computed incrementally using the positions on the previous line and the gradients of the sides of the polygon.

The Line Processor

The edge table passes its information for each scan line to a line processor in the form of spans. Each span is the intersection of a polygon with a scan line and is delimited by the left and right hand edges of the polygon (Fig. 5). The line processor uses this information to insert the span, a pixel at a time, into a line buffer. This is a memory containing information for each pixel on the scan line.

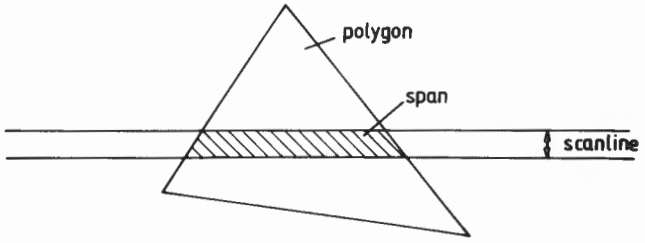

Figure 5 The Span

For each insertion at each pixel, comparisons are made to determine if the newly added span is closer than any span already recorded at this pixel. The distance comparison is based on the use of proximity, the inverse of the distance of the object from the viewpoint. The proximity value is updated at each new scan line by the addition of the gradient increment. Each span also has a proximity gradient over its length which is calculated incrementally from the values of proximity at the left and right edges of the span. The proximity value is used to determine, at each pixel, the proximity ordering of all the spans at that pixel using a simple insertion sort. By limiting the number of

spans at a particular pixel to four, the time for this process is not excessive. The process is very much like Z buffering, but with the use of proximity which can be computed by linear incrementation. After all the spans have been sent by the edge table generator, and have been processed by the line processor, there will be a list of data for each pixel on the line. Each list will contain information about the closest polygon at that pixel and any occulted polygons.

Pixel Processor

The lists for each pixel are then sent to the pixel processor which calculates the colour of the pixel, modified to incorporate any special effects.

Unlike the edge table generator and the line processor, this part of the system will change in function as well as size, depending on the application. The number of processors required would also depend on the application, normally with at least one pixel processor per line processor (Fig. 4).

Distance fading can be performed using a lookup table, accessed by proximity to yield giving a factor used to modify the intensity of the pixel.

Translucency can be calculated for the polygons present at a pixel since these have previously been listed in order of decreasing proximity.

Anti-aliasing

Aliasing occurs as anomalies on the display, caused by the finite sampling of the display image at pixel intervals. One method of overcoming aliasing is to increase the resolution of the display. This is not always feasible or economical, since every doubling in the display resolution quadruples the display processing requirements.

A more economical technique employs sub-pixel masks, consisting of a four by four binary array of sub-pixels. These are used to indicate the amount that a polygon intersects with a particular pixel. A full mask would indicate an interior value whereas a partial mask would represent the edge of a polygon.

These masks are created by the line processor for each pixel of each span, using the gradients of the left and right hand edges of the span (Fig. 6).

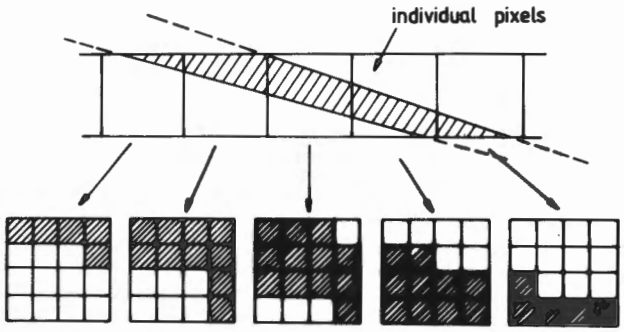

Figure 6 Pixel Masks

The pixel processor, in calculating the colour of a particular pixel, uses these mask values to determine the contributions of each polygon at this pixel. The contribution of the closest polygon will be determined by the number of bits in its mask set out of sixteen. The contribution of the next closest polygon will be the number of bits set after the operation of logically ANDing its mask with the inverse of the previous mask, and so on.

Acknowledgements

The contribution of our colleagues M.Agate, H.Finch, A.Garel, and A.Simmonds is acknowledged. The work has been supported by the UK Science and Engineering Research Council.

References

AMD (1985) Am29300 Family, Advanced Micro Devices, Sunnyvale, CA.

Analog Devices (1985) High-Speed 64-bit IEEE Floating Point Multiplier and ALU ADSP-3210 and ADSP-3220 Analog Devices DSP Division, Norwood, MA.

Clark JH (1982) The Geometry Engine: A VLSI Geometry System for Graphics. Computer Graphics, Vol 16/3

Fuchs H, Poulton J, Paeth A, Bell A (1982) Developing Pixel Planes, a smart memory-based raster graphic system. Proc MIT Conf on Adv Res in VLSI, p 137

Gouraud H (1971) Continuous Shading of Curved Surfaces. IEEE Transactions on Computers, Vol 20/6

Grimsdale RL, Hadjiaslanis AA, Willis PJ (1979) Zone Management Processor; a module for generating surfaces in raster colour displays. Computers and Digital Techniques, Vol 2/1

Price SM (1984) D.Phil. Thesis, University of Sussex

Sutherland IE, Hodgman GW (1974) Reentrant Polygon Clipping. Comm ACM Vol 17/1, No. 1