

AN EXPANDABLE MULTIPROCESSOR ARCHITECTURE FOR VIDEO GRAPHICS
(Preliminary Report)*

Henry Fuchs
The University of North Carolina at Chapel Hill

Brian W. Johnson
The University of Texas at Dallas

Abstract

Presented is the design of a flexible expandable multi-processor system for video graphics and image processing. The design involves a central controller which broadcasts data to a variable number of independently executing processing units, each of which in turn controls a variable number of memory units among which the video (frame buffer) image is distributed. An interleaved addressing organization of the video memories guarantees both an even workload distribution as well as maintenance of image coherence for each processing element. Execution speed and image resolution can be independently altered (at any time) by varying the number of processing and memory units. Sample applications of the system -- for rapid line drawing and "electronic scene generation" (visible surface algorithms) -- are described. Variations of the design for low cost and for powerful, real-time configurations are outlined.

Introduction

A long-standing goal of researchers in computer graphics systems has been the development of real-time three-dimensional modeling systems. These systems, which produce a realistic image of a simulated three-dimensional environment, have a wide variety of potential uses -- from simulators for pilot training to interactive design of houses and automobiles. The most sophisticated of these systems produce, in real-time, images on color video displays (TV's) of startling reality. The only limitation to widespread use of these systems has been their prohibitive costs (\$500,000 and up). Thus virtually the only uses today are those for which there is no real alternative -- e.g., simulating maneuvers in gravity-free space or training simulators for pilots of large (and

expensive) airplanes. If such modeling systems could be provided at significantly lower costs, it is safe to presume that their use would become dramatically more widespread.

A short examination of the computational expense of the problem suffices to justify the complexity and expense of current systems which solve it. A video image to a digital system normally consists of a matrix of picture elements ("pixels") of between 480 and 512 rows (scan lines) with from 512 to 640 pixels in each scan line. (Until recently this size was limited by the resolution of video monitors. Within the past two years, monitors with 900 to 1000 scan line capacity have become available; the factor of four increase in number of pixels per image only exacerbates the computational problem.) The image is then simply a set of some 300,000 pixels, each of which (for a color image) contains three independent components -- Red, Green, Blue -- each usually to 8-bits of resolution. The entire problem at hand is simply calculating these 900,000 values each time the image is scanned out onto the video screen, usually 30 times per second.

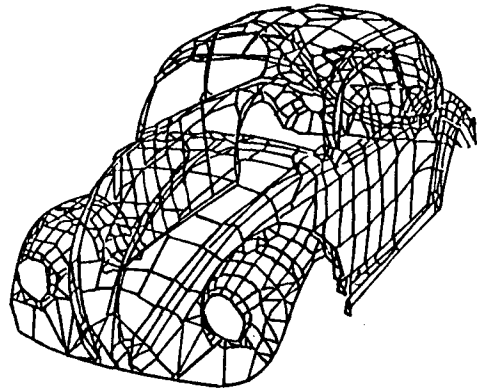


Fig. 1
(From Sutherland, Sproull, & Schumacker(1974))

* This work was partially supported by NSF Grant MCS-77-03905, and by Naval Electronics Systems Command Contract N00039-78-C-0431, (through Research Triangle Institute Grant 43U1667).

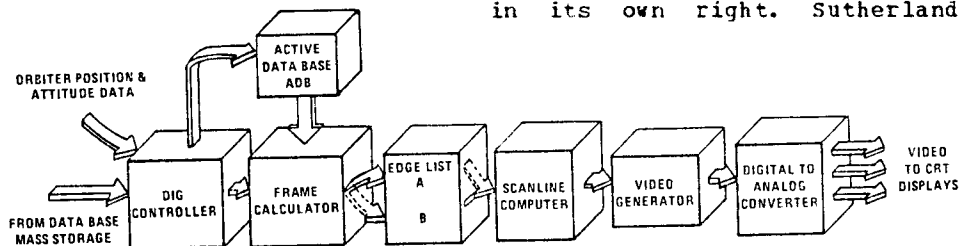
The proper value at each pixel is a function of the data base (the simulated environment), the viewing position and

orientation of the simulated viewer, and the location(s) of the light source(s) in the simulated environment. The environment is most often described as a set of objects in the environment (Euclidian three-space) coordinate system. Each object is usually described by a set of planar tiles ("polygons") which form its various surfaces. (Fig. 1, from Sutherland, Sproull, and Schumacker (1974), shows the boundaries of a set of polygons defining the surface of a 3-D object.) (Other methods of object description are sometimes used -- e.g., as collections of geometric solids (MAGI(1968)) or curved surfaces (Catmull(1975), Blinn and Newell(1976)). Since the particular object definition method does not significantly affect the system architecture, we shall assume hereon that the common planar-polygon descriptions are used.) In order to compute the Red, Green, and Blue values for a particular pixel, the system has to determine:

- a) which, if any, polygons map onto this pixel's area,
- b) which one from this set is closest to the viewer (and thus is the one visible obscuring all the other polygons), and
- c) the details about the precise part of this closest polygon which maps onto the pixel -- its assigned color, its angle and distance from the light source(s), and its angle and distance to the viewer.

When programmed on a conventional general purpose computer, computing such a simulated image may well take several minutes, and easily longer; so developing a system to do it in 1/30 second is a non-trivial task. (The bibliography lists references to algorithms and systems which influenced the design presented in this paper.)

To understand our solution, let us first examine the overall sequence of steps which need to be performed in order to produce a visible surface image on a video display.



Courtesy of
Link-Fingers

Fig. 2
(From Shohat & Florence(1977))

- a) The original polygons (in object coordinate space) are transformed into the position as seen from the simulated viewing position. (This is a sequence of rotations and translations.)
- b) The parts of the environment data base which are not in the field of view are discarded from further consideration by clipping all polygons against the boundaries of the field of view.
- c) Perspective transformation is applied to foreshorten the appropriate environmental parts as a function of distance.

It is at this point that a visible surface algorithm is invoked.

Since steps a), b), c) can be achieved in real-time by current affordable line drawing systems (e.g., Evans and Sutherland(1976), Vector General(1978)), we will concentrate our attention on the actual visible surface computations. (Of course, these line-drawing systems are affordable partially because they do not have to perform the laborious visibility computations for some 300,000 pixels!) Most current real-time video systems (Evans and Sutherland(1977), Shohat and Florence(1977)) use a pipeline architecture to achieve the necessary high throughput rates. (See fig. 2 from Shohat and Florence(1977).) Each module in the pipeline is typically a highly specialized processing unit. Thus, these designs do not easily lend themselves to substantial upgrading (to achieve higher capacity) or downgrading (to achieve lower cost).

Our own design capitalizes on the newly plentiful resource of inexpensive LSI circuitry. It allows a significant but bounded increase in both memory and processing requirements in return for architectural flexibility. Specifically, our solution is tailored -- although not restricted -- to what may be the simplest visible surface algorithm, the so-called "Z buffer" algorithm, one so simple that it seems never to have appeared in print in its own right. Sutherland, Sproull,

and Schumacker(1974) mention it in passing (p.51), saying "that if a large memory is available ... This method results in a computing cost which depends only on the depth number (Dc) and not otherwise on the environment complexity." (Dc is the number of front-facing polygons "pierced, on the average, by an arbitrary ray from the viewpoint.") Catmull(1975) used the method as part of a more sophisticated algorithm for visible display of curved surfaces. The basic algorithm utilizes two large buffers each containing an entry for each pixel on the screen, an "image" buffer which contains the (RGB) intensities at each pixel, and a "Z" buffer which contains at each pixel the distance of the closest object encountered there so far (fig. 3).

value is compared with the entry in the Z buffer for this pixel. If this new value is smaller than the current entry then this new polygon is closer to the viewer at this pixel than the closest previously encountered polygon and so this new polygon would now be visible at this pixel. Thus in this case the new Z value is put into the Z buffer and this new polygon's (RGB) intensity value is computed and inserted into the image buffer. If, on the other hand, the new Z value is greater than the value currently in the Z buffer at this pixel, then this polygon is farther than the closest polygon, and processing is terminated for this pixel for this polygon without any changes to the buffers. Processing continues with the next pixel into which the current polygon "falls."

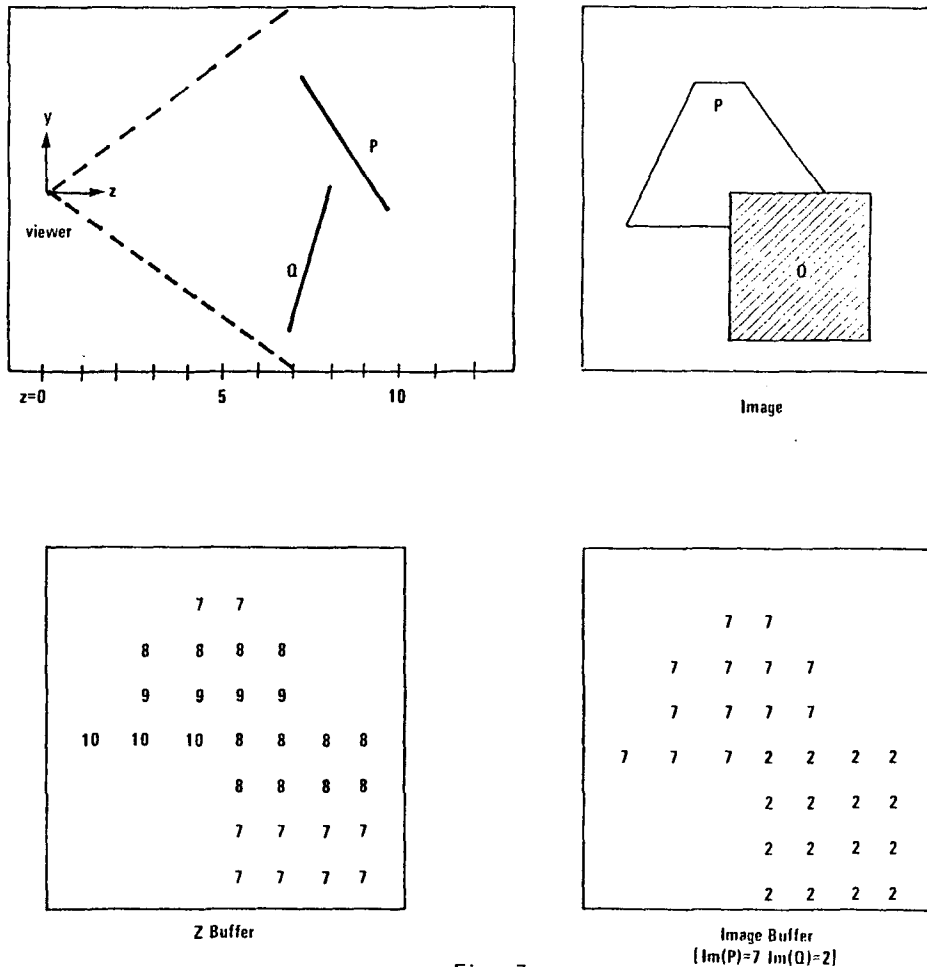


Fig. 3

The polygons are processed sequentially, in any order. Each polygon's processing starts with determining the pixels upon which the polygon "falls" in the image. For each such pixel the distance of the polygon from the simulated viewer is computed. (This is the "Z" value.) This

This simple algorithm is seldom used, principally for two reasons: 1) few current systems have sufficient memory for two such large buffers, and 2) every pixel of every polygon needs to be computed. To understand the potential severity of this

second reason, let us recall that traditionally designers of visible surface algorithms (e.g., Watkins(1970)) have attempted to gain efficiency by avoiding, whenever possible, consideration of all but the (single) nearest polygon. For example, if all the polygons potentially visible on a particular scan line can be considered together as a set, then determining the Z ordering on this set at just a few key points along the scan line is sufficient to determine the sequence of visible polygon segments along the entire line (fig. 4).

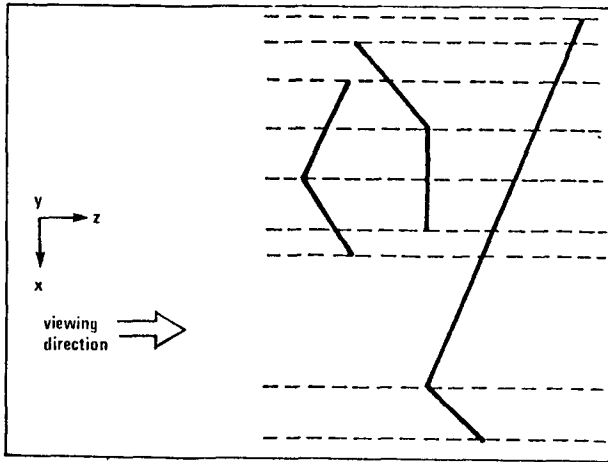


Fig. 4

At intermediate points all the obstructed polygons are simply ignored. A "Z buffer" algorithm, since it handles each polygon separately, computes every affected pixel for each polygon -- a procedure which certainly seems to be wasteful and inefficient. A closer examination of the situation, reveals that for multiprocessor systems the procedure may in fact be very attractive. Sutherland, Sproull, and Schumacker(1974) estimate that the average number of polygons "falling on" a pixel is only 3; that is, many (most?) images contain large areas of sky, water, ceilings, floors -- areas in which there are not too many polygons stacked one behind the other. This implies that the (in)efficiency of the Z buffer algorithm is constant; at worst it is some constant multiple (e.g., 3) of the most efficient possible algorithm -- one which can determine with negligible cost the visible polygon at each pixel. Since LSI technology is rapidly diminishing the cost of simple arithmetic processing units, a factor of 3 is no longer burdensome.

System Description

The fundamental system organization is as illustrated in figure 5.

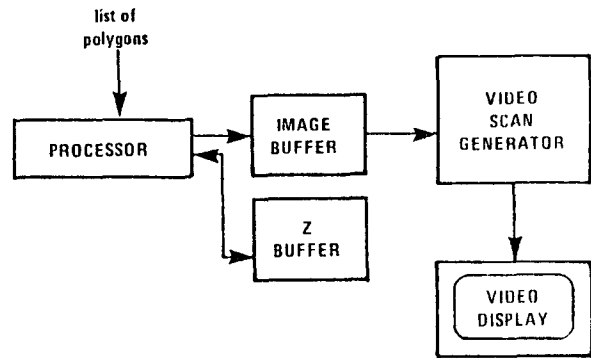


Fig. 5

Figure 6 shows in somewhat greater detail the organization of the image buffer, which is accessed by both the processor and the video scan generator.

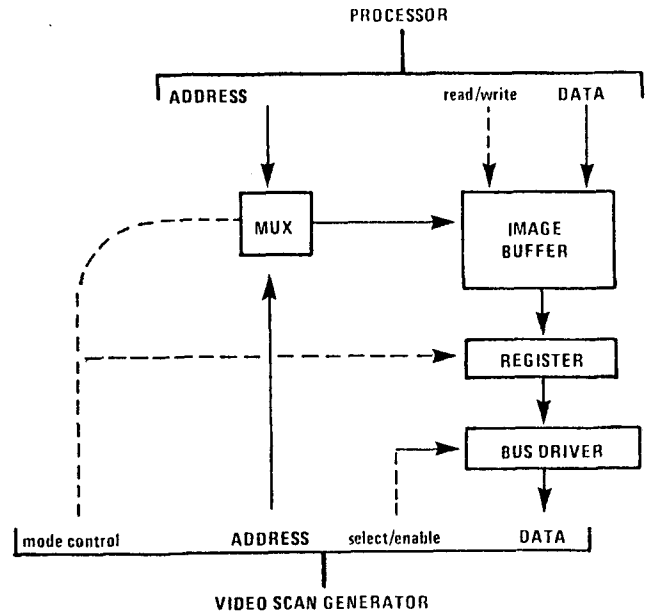


Fig. 6

Figure 7 illustrates the simple time division multiplexing between the processor and the video scan generator. We note here that the current pixel's data remains on the video scan generator bus even during the period which is assigned to the processor.

If we consider using only commonly available inexpensive LSI RAM's then the requirement of the scan generator (needing to cycle through the entire image in approximately 30 milliseconds) will limit the usefulness of this simple design to very coarse images. To increase the bandwidth we simply insert additional memory units onto the system bus.

ACCESS MODE

VIDEO SCAN GENERATOR

PROCESSOR

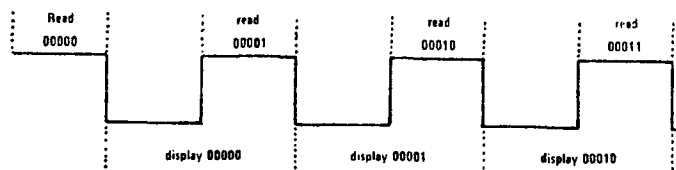


Fig. 7

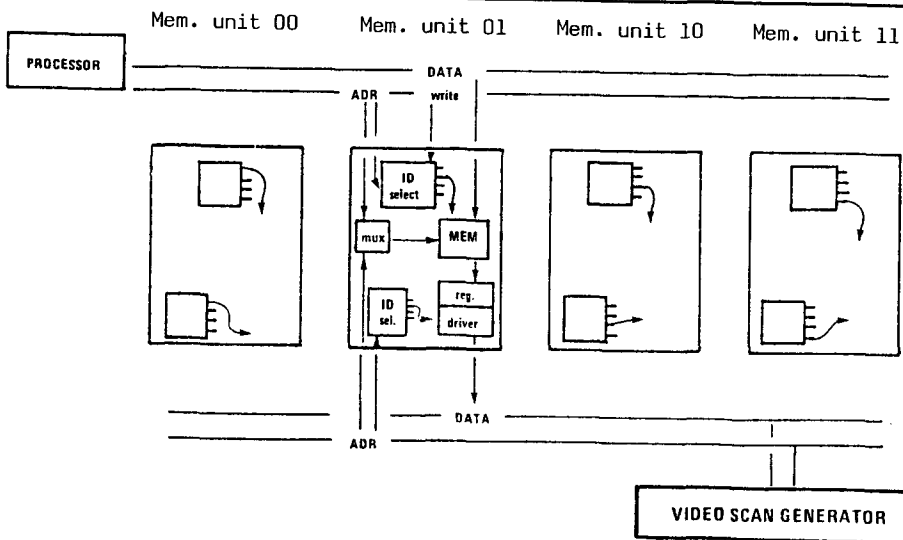
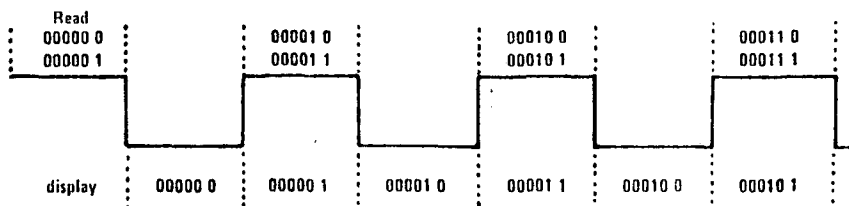


Fig. 8

ACCESS MODE

VIDEO SCAN GENERATOR

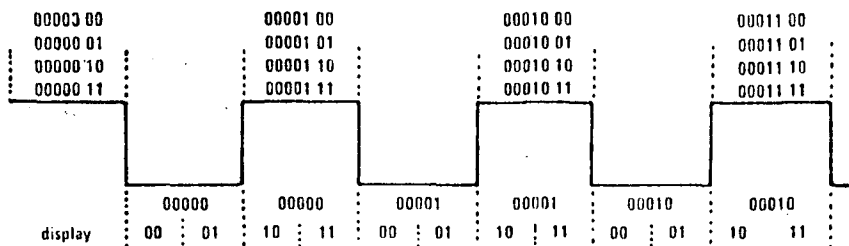
PROCESSOR



TIMING - 2 MEMORY UNITS

VIDEO SCAN GENERATOR

PROCESSOR



TIMING - 4 MEMORY UNITS

Fig. 9

Figure 8 illustrates the organization of this enhancement and figure 9 shows the timing cycles. Given that a scan-line is drawn on standard video monitors in approximately 40 microseconds, a video scan generator bus cycle of about 75 nsec. is adequate for images with up to 512 pixels per scan line. It is important to note that the actual bus to the scan generator does not increase in size or speed. All memory units are read in parallel during the scan generator access times. During the following complete timing cycle, the various results are put onto the video bus by enabling, in sequence, the bus drivers of the various memory units. This enabling is directly controlled by the least significant bits of the video scan generator's X address. In this fashion the number of memory units need not be known to the scan generator; if there are fewer units, some of the least significant address bits are ignored and thus consecutive locations on the video screen will be accessed from the same image memory unit's output register. The result will be a coarser image (128 x 128, say, instead of 512 x 512) than the scan generator is capable of producing. (It will be seen later that a somewhat different resolution-independence scheme for the processor side of the memories will free the entire system -- both hardware and software -- from reliance on a fixed resolution.) The proper ID selection in each memory unit (as seen in Fig. 7) is a function of both the unit's ID number and the total number of memory units currently in the system. Although such selection settings are normally set manually through jumpers or DIP switches, we prefer for them to be set automatically. This is done through the following mechanism. In addition to the processor bus and video scan generator bus, the system includes a set of lines for ID numbers and the "total-units" number.

As illustrated in fig. 10, the ID lines consist of a set of lines sufficient to represent the largest possible number of memory units in a system. (For example, for a 1024 maximum memory unit system this number would be 10.)

In this fashion the set of lines are started at 0 on one side, each board has an incrementer circuit on it, and thus the number on the backplane ID lines is incremented by one each time it passes through a memory unit board (fig. 11).

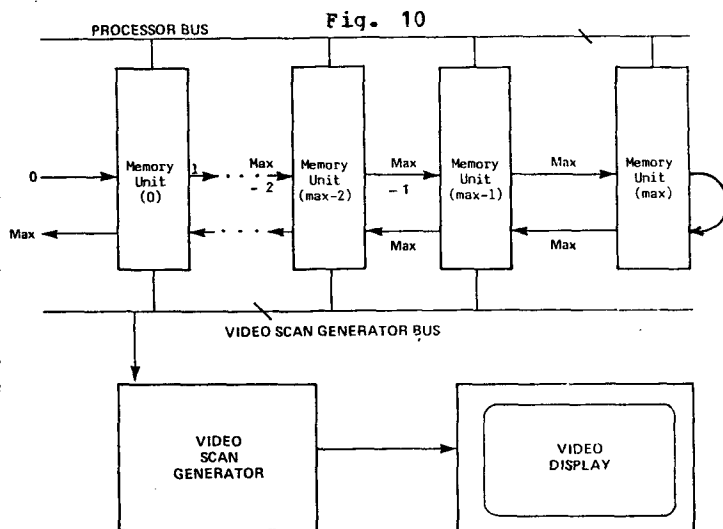
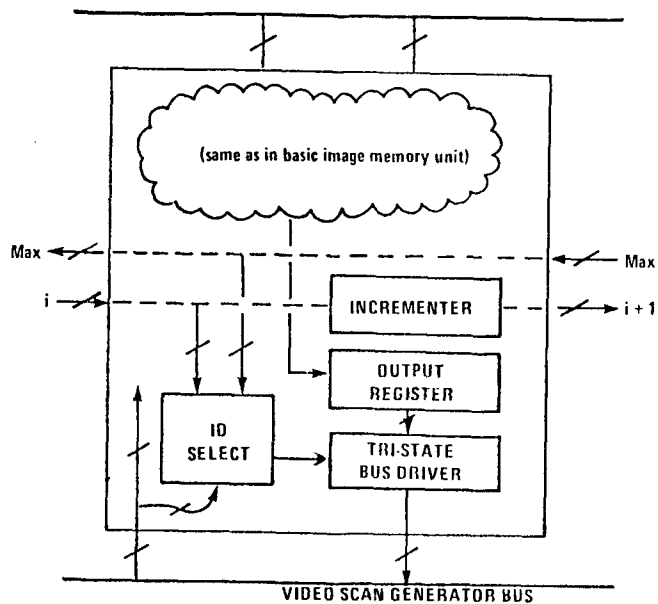


Fig. 11

A similar set of lines is used to return the ID signal value from the end of the system. (This number is simply the total number of memory units in the system at the present time.) With this technique boards can be inserted into or extracted from any position at any time without the necessity of any hardware (or software!) modification. Empty backplane slots will pass on all signals either by the use of shorting-type PC board connectors or by the insertion of dummy boards.

We also note at this point that neither the video scan generator nor the image memories rely on any mechanism for altering the contents of the image

memories. Thus we can distribute responsibility for computing the image memories contents to a number of different processors.

Fig. 12 illustrates a modified organization which achieves this increased capability. Virtually the only addition has been the introduction of a central broadcast controller (CBC) which "announces" the description of each new polygon to all the processing elements (PE's). The system is designed to operate as follows:

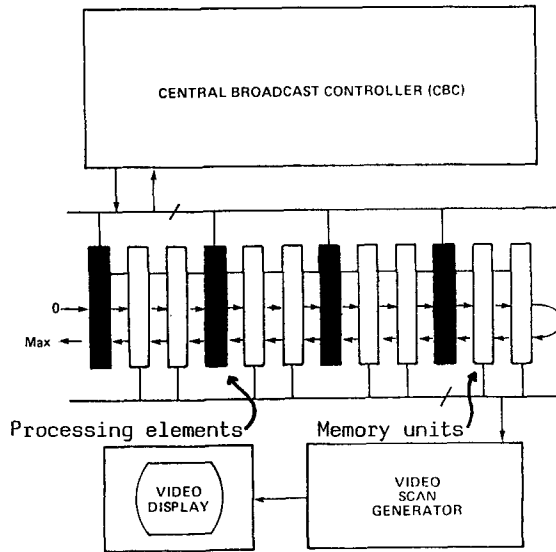


Fig. 12

- a) Immediately upon power-on, the CBC broadcasts the (possibly new) software to all the processing elements. (All PE's execute the same program, but each has a separate copy of it and each may be executing different parts of it at any instant.)
- b) The CBC instructs the PE's to survey the memory units under their control. This consists simply of each PE attempting to read and write a single word into each possible memory unit under its control. (Each knows (from the ID lines), 1) the total number of units in the system at this time, and 2) the first memory unit that is under its control; it simply needs to find the upper limit of its domain.)
- c) The Z and image buffers are initialized by each PE.
- d) The actual processing proceeds now with the CBC broadcasting description of one or more polygons

to be processed. A polygon description may consist of the sequence of the polygon's vertices, with each vertex being described by its X,Y,Z location and its calculated color/intensity value. Broadcasting additional information (to be described later) will speed up processing by eliminating the need for calculating certain common values in each of the PE's. Since each PE knows which MU's are under its responsibility and how many MU's are in the system, it can easily compute the location of each of its pixels on the screen. For each polygon it does the appropriate Z buffer algorithm computations (as outlined before) for all its pixels affected by this current polygon. When done, each PE signals to the CBC. When all the PE's are done, the CBC broadcasts the next polygon (or set of polygons). The procedure continues until the complete set of polygons in the scene is exhausted.

By having the MU's and the PE's implemented on the same size PC cards and utilizing the same connectors, all the PE bus lines can be implemented on a single set of backplane lines. A PE ignores any such signals coming in from its left, and generates its own signals on the lines to its right. (PE's pass these signals through.) Thus a PE controls all the MU's between it and the next PE on its right. Configurations can be altered by simply repositioning the boards.

Fig. 13 illustrates that regular interlacing is possible for both the MU's and the PE's. Fig. 13a illustrates the physical backplane ordering of PE and MU boards in a typical configuration. Fig. 13b shows part of this raster configuration's image; each rectangle is an individual pixel, with the integer enclosed indicating the MU in which its value is stored and the letter indicating the PE which calculates the pixel's value. The regular interlacing increases the efficiency of processing, in two ways: 1) it guarantees that for practically any polygon the pixels on which it lies will be located in the domains of a number of different PE's, so that the workload will always be distributed, and 2) the regular pattern of affected pixels in any one MU allows rapid incremental computations for Z, and eventually for RGB. (Recall that all these polygons are planar; so the amount of change per each pixel step will be constant.) Also, the same regular pattern occurs in each affected MU; for example, if adjacent pixels in a particular MU are 2 units apart in X and 4 units in Y, then they will be that way for every affected memory unit. This allows

the CBC to compute the appropriate incremental change values during the time the PE's are processing the previous polygon. The CBC can then broadcast these values directly, thereby avoiding a computation step in each PE.

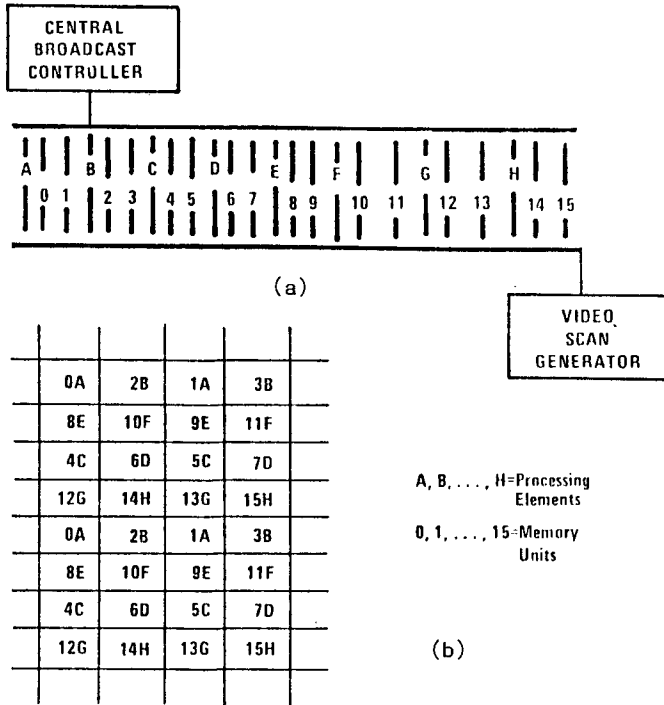


Fig. 13

Fig. 14 shows how particular configurations can be modified to increase or decrease image resolution or processing speed. The grids again illustrate a small raster-image area and how its PE and MU assignment varies with different configurations. Note that not only pixel size, but pixel aspect ratio can be controlled by the MU allocation and mapping. (The variations in processor-memory assignments from those of fig. 13 reflect the computations performed by the memory ID select modules illustrated in fig. 10.) Fig. 15 illustrates the physical organization corresponding to the various resolution/speed configurations of fig. 14.

Determining the size of MU's for a particular system involves balancing a number of conflicting demands. Larger MU's allow larger (say, 16K) LSI chips to be utilized, and allow fewer MU's to realize a reasonably high resolution (say, 512 x 512 pixel) image. Since only a single word can be accessed from any chip in a given memory cycle, however, and since words correspond to pixels in this design, only a single pixel can be accessed in a chip in a single memory

cycle; larger MU's thus need faster memory cycles to keep up with the raster-scanning

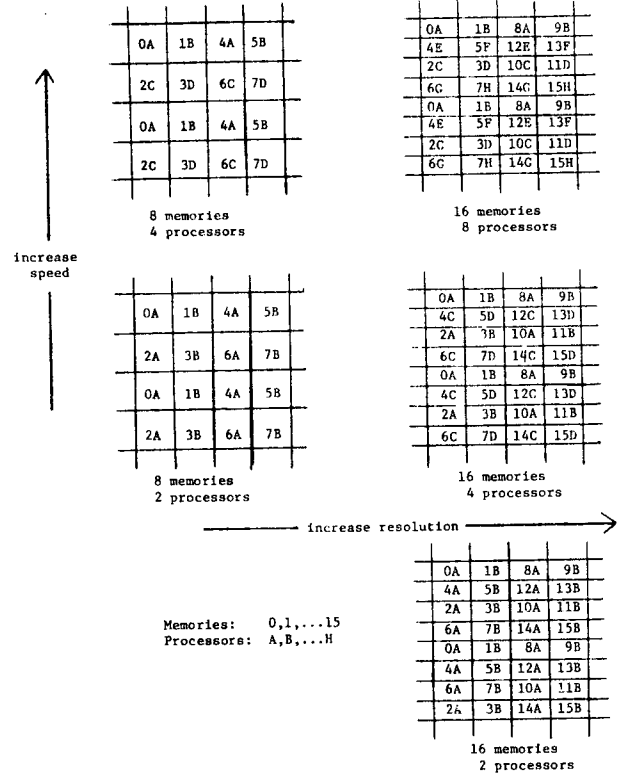


Fig. 14

of the image. In addition, since no more than one PE can control any MU, smaller MU's allow more PE's and thus a faster system. One configuration currently under investigation uses 8K pixel MU's with 300 nsec. cycle times.

Let us consider some of the capabilities of this kind of organization. At this point we limit our comments to structural, qualitative issues, since we cannot yet give an accurate quantitative assessment. (We are just beginning performance prediction simulations; preliminary results indicate that an execution time of 50 microseconds per pixel per polygon can be achieved.) The basic design allows significant flexibility in tradeoff between power and economy. On the one extreme there can be systems with only one PE and one MU. Of course such a system would exhibit a very coarse image, but it may be suitable for simple video games, for instance. On the other extreme one can configure a system with high resolution and very high throughput. Such high-resolution and high powered systems would be appropriate, for instance, for interactive pilot-training simulators. The only difference, however,

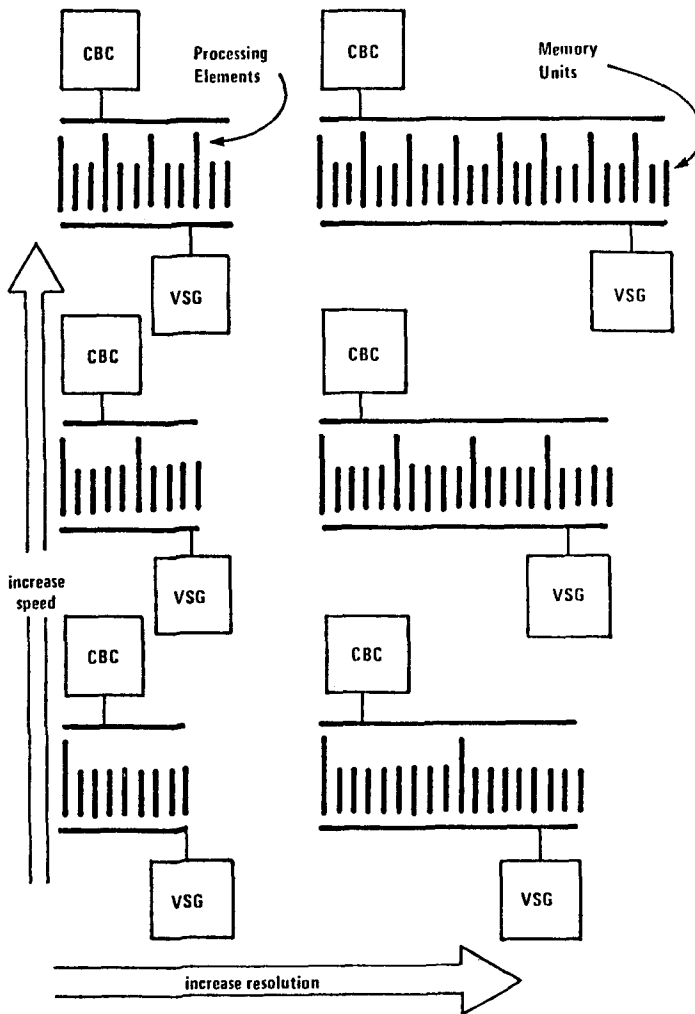


Fig. 15

between these two extreme configurations would be the number of PE boards and the number of MU boards. The software in the PE's of both systems would be identical. The CBC's would be identical. (The polygons are broadcast in highest resolution units.) The video scan generators could also be identical. (They assume a high-resolution system; since both MU and image resolutions are powers of 2, small systems' MU's ignore some least significant bits of the VSG.) It is reasonable to speculate that a large computing facility may have a number of machines, each with a different number of MU and PE boards -- many small configurations for program development, a few large ones for interactive simulation, and some high resolution but slow ones for non-time-critical applications. For special occasions, larger configurations could easily be constructed by simply consolidating several small configurations. Also, faulty boards could

simply be removed from a system.

These systems should also degrade gracefully. Some current real-time systems encounter difficulty due to computations being done "on the fly" as the video beam scans the image. These systems thus avoid using an image buffer between the processing and scanning-out modules. If a certain spot in the image is particularly complex, however, the scan either has to wait, or it "paints" incorrect data. The design presented here would not exhibit such behavior. The system would simply take slightly longer to compute the new image. If the memories were double buffered, the switch between the old image and the new one would be made slightly after the start of the second scan of the old image -- or if the situation were really complex, the switch would be made after two or more complete scans of the (old) image.

Other Applications

It is easy to see at this point that the system is not restricted to executing a Z-buffer visible surface algorithm. Software could be loaded into the PE's, for instance, to perform digital vector generation and rapidly create line drawings on the video screen. In this case, the CBC would simply broadcast endpoint information, each of the PE's would determine the pixels under its control which are affected by the new line segment; it would then set each of these pixels appropriately.

Implementation

We are currently in the process of implementing various aspects of the above design. We have prototyped simple versions of each module and plan to have a small, but complete prototype system in the near future.

Future Developments

We are currently generalizing the scope of the present design. For example, the simple selection and multiplexing for both memories and processors is most easily achieved when the number of units is an even power of 2. Although some sort of processor-memory-image assignment can easily be achieved for an arbitrary number of units of each, an optimal generalized mapping algorithm still remains to be developed.

Fault-tolerant and "highly reliable" versions of the current design may also be quite useful. Although some of this is

presently available with the capability to remove faulty modules, other capabilities can perhaps be added. For example, configuring the system to generate a higher resolution image (say, 1024 x 1024) than the one being displayed (512 x 512) would allow the scan generator to consider (in this case 4) separate sources from which to determine each single pixel. Such redundancy should easily allow significant number of faulty memory and processor modules without noticeable image or performance degradation.

Acknowledgement

We thank Mr. Joe Parks of the Department of Computer Science, University of North Carolina, for assistance in analyzing the performance of Z-buffer algorithms.

Bibliography

- Appel A. (1967) The notion of quantitative invisibility in the machine rendering of solids. Proc. ACM Annual Conference 387-393.
- Blinn, J. F. and M. E. Newell (1976) Texture and reflection in computer generated images. Comm. ACM 19(10): 542-547.
- Bouknight, W. J. (1969) An improved procedure for generation of half-tone computer graphics representations. University of Illinois, Coordinated Science Laboratory, R-432.
- Catmull, E. A. (1975) Computer display of curved surfaces. Proc. Conference on Computer Graphics: Pattern Recognition and Data Structures (IEEE Cat. No. 75CH0981-1C): 11-17.
- Despain, A. M. and D. A. Patterson (1978) X-tree: a tree-structured multi-processor computer architecture. Proc. Fifth Annual Symposium on Computer Architecture 144-150.
- Evans and Sutherland Computer Corporation (1976) Picture System 2. Salt Lake City, Utah.
- Evans and Sutherland Computer Corporation (1977) Improved scene generation capability. Final report, NASA contract No. NAS 9-14010.
- Hirschberg, D. S. (1978) Fast parallel sorting algorithms. Comm. ACM 21(8): 657-661.
- MAGI (1978) Mathematical Applications Group, Inc. Elmsford, NY. Promotional literature.
- Newell M. A., R. G. Newell, and T. L. Sancha (1972) A new approach to the shaded picture problem. Proc. ACM Annual Conference
- Roberts, L. G. (1963) Machine perception of three-dimensional solids. MIT Lincoln Laboratory, TR 315. Also in Optical and Electro-Optical Information Processing, Tipper, et al., eds. MIT Press, 159.
- Rougelot, R. S. and R. Schumacker (1969) G.E. real time display. NASA Report NAS 9-3916. General Electric Co., Syracuse, NY.
- Shohat, M. and J. Florence (1977) Application of digital image generation to the shuttle mission simulation. Proc. 1977 Summer Computer Simulation Conference.
- Schumacker, R.A., B. Brand, M. Guilliland, and W. Sharp (1969) Study for applying computer-generated images to visual simulation. U.S. Air Force Human Resources Laboratory. AFHRL-TR-69-14.
- Sutherland, I. E., R. F. Sproull, and R. A. Schumacker (1974) "A Characterization of Ten Hidden-Surface Algorithms." ACM Computing Surveys, 6(1): 1-55 (Figure 1 reprinted by permission of ACM).
- Vector General, Inc. (1978) System 3400, Woodland Hills, CA.
- Warnock, J. E. (1969) A hidden surface algorithm for computer-generated halftone pictures. Computer Science Department, University of Utah, TR 4-15.
- Watkins, G. S. (1970) "A real-time visible surface algorithm". Computer Science Department, The University of Utah: UTECH-CSC-70-101.
- Wieler, K. and P. Atherton (1977) Hidden surface removal using polygon area sorting. Proc. Fourth Annual ACM-SIGGRAPH Conference on Computer Graphics and Interactive Techniques : 214-222.