Fuchs 77

from Proceedings of the 1977 ACM Annual Conference, Seattle, Washington, Oct. 17-19, 1977.

DISTRIBUTING A VISIBLE SURFACE ALGORITHM OVER MULTIPLE PROCESSORS*

Henry Fuchs
The University of Texas at Dallas
Richardson, Texas

ABSTRACT

Described is a procedure for executing a visible surface algorithm in a new multi-microprocessor system which utilizes distributed image and depth ("Z") buffers. It is shown that despite image distribution over a large number of processing and memory units, object coherence can still be maintained and used to reduce the number of calculations needed to generate a continuous-tone visible surface image.

Key words and phrases: Visible surface algorithms, multi-processing, microprocessors, three-dimensional computer graphics

CR Categories: 8.2, 6.22, 6.35, 4.32

INTRODUCTION

Visible surface algorithms which generate continuous-tone, most often video, images have always been computationally expensive. It is easy to see why this is so. Such an algorithm has to calculate an appropriate intensity value for each picture element ("pixel") in the image array.

In order to do this, it first has to determine which object in the scene is closest to the viewer of all the objects which cover that particular pixel. (It is this closest object which will be visible and will obscure any objects behind it.) Once this closest object is found, the intensity of the pixel is calculated, based on 1) the shade assigned to that closest object surface, 2) the angle of the surface to the viewer, and 3) the illumination to the object surface. (This procedure is most lucidly explained in Blinn (1976).)

To find the object which is closest to the viewer at a particular picture element, all the object definition components -- usually planar tiles ("polygons") -- are sorted, in order, along all three axes: X and Y, the horizontal and vertical directions in the image, and Z, the distance away from the image. Sutherland (1973) has classified many of the major visible surface algorithms based largely on the order of the axes in which this sorting is performed. The classic algorithm by Watkins (1970), for example, uses a Y, X, Z sort, first sorting all the object polygons along Y -- from top to bottom of the image, according to the tops of each polygon, then for all the polygons which intersect a particular Y value (a scan line) it sorts along X, left to right, by the position at which each polygon first "appears." Then, for a particular X value of interest along the scan line, it sorts by Z all polygons which appear there, from front to back. The polygon at the head of this list is the one visible at the particular pixel at this X,Y location.

Sutherland (1973) notes that the major efficiency gained by these algorithms is the reliance on a high degree of picture coherence; that is, the structure and appearance of the image at a given pixel or line is almost always very similar to its structure and appearance at an adjacent pixel or line. This fact, together with the ease of updating the sorted list values at the previous pixel or line, allows the closest-object sort to be executed not from the beginning at each pixel, but simply as a modification of the sort's result at a previously calculated adjacent pixel or line. In general, for a Watkins-type algorithm (as well as for the one to be described below) this update consists of 1) updating all the polygon elements in the sorted list from the previous pixel to obtain each element's Z value at the new pixel location, 2) culling this sorted list to remove all those polygons for which a terminal edge has been encountered and which thus do not appear in the current pixel, and 3) merging this culled, but still sorted list with the sorted list of all those object polygons which have an entering edge here and thus begin to appear at this current pixel. Minimizing the number of steps in this update process is crucial to the effectiveness of the resulting algorithm. It will be shown that the new distributed algorithm can also perform such efficient updating, and in fact needs to update only a single polygon at any one time.

* This work was supported in part by NSF Grant MCS-77-03905

The algorithm for the new distributed system is most similar to a "Z buffer" algorithm, as described by Catmull (1974). Such an algorithm uses two buffers, each containing a cell for each pixel in the image: the "image" or "frame" buffer contains the intensity of the image at each pixel, the other, the "Z" buffer contains at each pixel the distance from the viewer of the closest object surface which has so far been encountered for that particular pixel. The "image" buffer is initialized to some arbitrary background intensity -- usually white or black. (Generalizations to color images are not included in this discussion, but they are straightforward.) The Z buffer elements are each initialized to the maximum possible value. In this algorithm no sorting is necessary; rather, the polygons can be treated individually, in any order. For each polygon, each pixel which is covered by the polygon is considered. The Z value of the polygone at this pixel is calculated. It is compared with that pixel's current value in the Z buffer. If the new polygon's value if greater, then there must have been a previous polygon which covered this pixel which was closer to the viewer; so processing of the current polygon at this pixel is aborted and processing continues with the calculation of the polygon's Z value at the next pixel. If, however, the new polygon's Z value is smaller than the value in the Z buffer, then this new polygon is closer than the closest previously encountered one; thus, 1) the new polygon's value is stored in the Z buffer, and 2) an intensity for this polygon's surface at this pixel is calculated and stored in this pixel's image buffer. After all the pixels covered by the current polygon are considered, another polygon is processed. After all the polygons have been processed, the Z buffer contains, at each pixel, the distance of the closest polygonal surface at that pixel, and the image buffer contains the intensity of this closest -- and thus visible -- surface.

SYSTEM ARCHITECTURE AND ALGORITHM STRUCTURE

The multi-processor system on which the algorithm is to be implemented is based on a variable number of processing elements and a variable number of memory units over which the image and Z buffers are distributed. The system architecture and implementation are described in detail in Fuchs and Johnson (1977). In this paper only those details necessary to understand the algorithm will be described.

The system (see Figure 1) consists of a number of processing elements -- usually a power of 2 -- and a number of memory units, usually some power of 2 multiple of the number of processing elements, some of which make up the image buffer, and others which make up the Z buffer. The spatial resolution desired for the image determines the number of memory units which are needed. The image buffer memory units are dual ported with alternate accesses allocated to the video scan generator which constantly displays onto a video monitor the current contents of the distributed image buffer. The remaining memory cycles can be utilized by the processing element connected to the memory unit.

Each processing element contains a CPU, local program and data store and is connected to some number of memory units. All the processing elements simultaneously receive data, in the form of polygon vertices, from the central broadcast controller which is the system's sole interface with the host computer. Upon receiving a polygon's definition, each processing element performs a Z buffer-like algorithm on the parts of the image which are covered by this polygon and are under the processing element's control. When a processing element has completed its calculations for the current polygon, it signals the central broadcaster by raising the value on a one-bit output "done" line. This line, controlled by an open-collector gate at each processing element, will only go high after all the processing elements have signalled "done." Since a polygon will generally occupy a slightly different number of pixels in the various processing elements' buffers, the processing completion times will be staggered. When the broadcast controller detects that the "done" line has gone positive, it will begin to transmit the vertices of the next polygon to all the processing elements. This over-all sequence is repeated until all the polygons have been processed and the entire image generated. (Slightly less overall waiting may occur between transmission periods if more than one polygon description is transmitted at one time. The polygons are then stored by each processing element in its local memory and processed in sequence, with the processing element signalling "done" only after completing all the polygons.)

Clearly the execution time of this system is dependent on the number of steps needed by the most burdened processing element for a particular polygon. Therefore the processing elements and their associated buffers have been interlaced according to the lowest bits of the X and Y image addresses. Figure 2 gives a sample organization of an 8-processor, 16-image-memory system and its image interlacing scheme. Since no two adjacent pixels are located in the same buffer and since no two buffers with adjacent pixels are assigned to the same processing element, it is easy to see that all polygons greater than one pixel in size will be processed by more than one processing element. In general, if a polygon encloses a square region of some modest size -- whose area is approximately the number of processing elements -- then all the processing elements will be guaranteed to participate in the polygon's pixel calculations.

The regularity of the interleaving pattern allows the system to retain the advantages of picture coherence discussed in the introduction. Thus a processing element can calculate its own next candidate pixel by simply updating its current pixel's X, Y and Z values with increments which are standard for the entire system.

This kind of system has the additional advantage that it doesn't fall into the usual rigid dichotomy of real-time or non-real-time implementations. Current systems tend to fall into one of these two categories because software implementations are usually much too slow for real-time (1/30 second) image generation and thus only very specialized hardware systems can perform image generation "on the fly." Even these systems, however, can only perform with such speed for scenes under a certain

level of complexity. Once the particular threshold is exceeded, the hardware cannot keep up with the video scan, and the resulting image degrades rather rapidly.

This distributed algorithm, since it generates the image into a buffer, never suffers from image degradation, it may simply take longer to complete a more complex scene than a less-complicated one. Further, if high speed image generation is desired, with the last completed image in one half of each memory unit constantly being accessed by the video scan generator, while the new image is created in the other half of the memory unit. A single system line could be utilized to switch between the two halves whenever the entire scene -- i.e., all the polygons -- were completed.

If the scene complexity were found to cause execution time to be longer than desired, the system could be reconfigured with additional processing elements. The new configuration, with fewer memory units attached to each processing element, would execute precisely the same algorithm, but taking less time to process each polygon, thus less time to process the entire scene. Alternately less expensive configurations could be constructed with fewer memory processing modules, with a resulting sacrifice in speed of execution and/or spacial resolution. Thus it may now be possible to have the same basic architecture, as well as identical software, and thus basic compatibility between the large, fast real-time systems and the desk-top terminals in offices and laboratories.

ACKNOWLEDGEMENTS

The author wishes to thank Brian Johnson for many helpful discussions, and David Rowe for implementing the initial designs of the memory and processing units.

REFERENCES

[1] Blinn, J. F., and Newell, M. E. Texture and reflection in computer generated images. Comm. ACM 19, 10 (October 1976), 542-546.

[2] Catmull, E. A. Computer display of curved surfaces. Proc. Conf. on Comptr. Graphics, Pattern Recognition, and Data Structure, May 1975, pp. 11-17 (IEEE Cat. No. 75CH0981-1C).

[3] Fuchs, H and Johnson, B. W. A multi-microprocessor system for video graphics. Tech. Rep. MMS-31, Mathematical Sciences, U. of Texas at Dallas, Richardson, Texas, September 1977.

[4] Sutherland, I. E., Sproull, R. F., and Schumaker, R. A. A characterization of ten hidden-surface algorithms. Computing Surveys 6, 1 (March 1974), 1-55.

[5] Watkins, G. S. A real-time visible surface algorithm.. Tech. Rep. UTEC-CSC-70-101, Dep. Comptr. Sci., U. of Utah, Salt Lake City, Utah, June 1970.

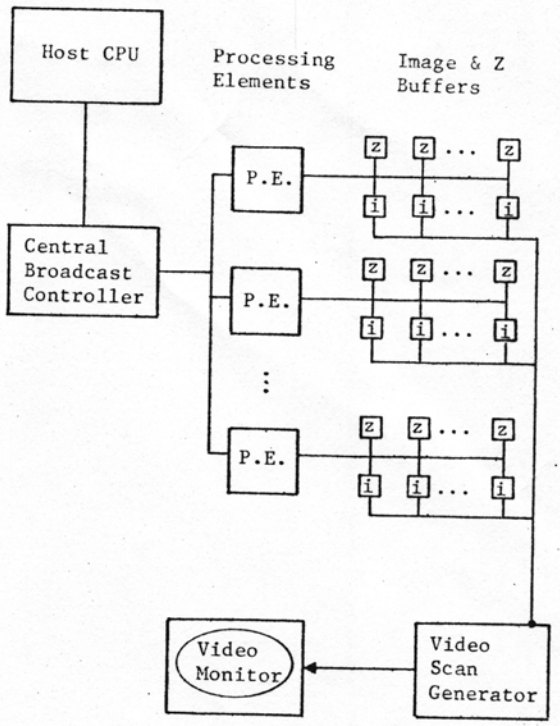

Figure 1: Distributed System Architecture

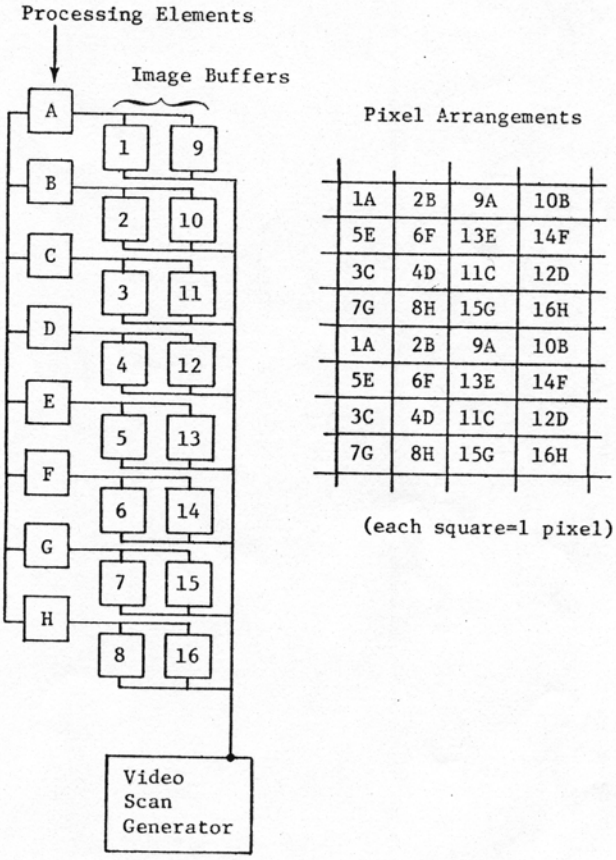

Figure 2: Image Interlacing