A VLSI-ORIENTED ARCHITECTURE FOR REAL-TIME RASTER DISPLAY OF SHADED POLYGONS*

(Preliminary Report)

Donald Fussell
Bharat Deep Rathi
The University of Texas at Austin
Austin, Texas 78712

ABSTRACT

This paper describes the organization of a large-scale graphics hardware system which can produce color, shaded, anti-aliased, perspective images of complex three-dimensional scenes in real time. By complex scenes we mean those consisting of at least 25,000 polygons. In contrast, existing high-performance raster systems of this type can handle only 1000 to 4000 polygons. This level of complexity is attainable with reasonable cost and reliability only if large parts of the system can be implemented as custom VLSI chips. In particular, it is possible to replace a traditional frame buffer with a device which stores polygons rather than points and performs scanout, shading, and anti-aliasing on them. With the addition of other special-purpose chips which perform transformations, clipping, perspective projection, and lighting calculations, such a "polygon buffer" forms the core of a parallel, pipelined organization which achieves the desired level of performance.

KEYWORDS: Graphics hardware, raster display, real-time, VLSI

1.0 INTRODUCTION

Recent advances in the availability of design techniques and fabrication facilities for VLSI circuits have created new opportunities for designers to produce special-purpose architectures. Computer graphics is one of the application areas which has benefitted most from this situation to date. Using the custom VLSI design techniques introduced by Mead and Conway [11], a number of researchers in recent years have developed new hardware systems whose function is the high-speed implementation of algorithms essential to the production of high-quality images by computer [1] [2] [3] [7] [8] [14] [17]. These new approaches complement the work of designers of graphics hardware systems who did not have access to the technology of integrated circuit design and who therefore could not regard as feasible methods which may now may not only be possible, but in some cases even preferable.

The goal of this work is to design a hardware system which can produce color, shaded, anti-aliased, perspective images of complex three-dimensional scenes in real time. By complex scenes we mean those consisting of at least 25,000 polygons. In contrast, existing high-performance raster systems of this type can handle only 1000 to 4000 polygons in real time, so our requirements include about an order of magnitude increase in the complexity of the scene. In order to achieve increased performance cost-effectively, it will be necessary to make use of massive parallelism of operations and of pipelining techniques, while carefully designing the system to consist of only a few basic module types and thus only a few chip types which can be used in large numbers to achieve the desired performance at a low cost. In this sense, the architecture we are proposing is a radical one in that it would have been impractical at the level of complexity we wish to obtain without the use of special-purpose chips.

The reduction of this complexity into a simple, regular structure suitable for VLSI implementation is a task which must proceed in a top-down fashion as the system design progresses. This paper is a preliminary report of an ongoing project, and thus only the higher levels of the design are described here.

*This work supported in part by NSF Grant MCS-8109489.

2.0 SYSTEM OVERVIEW

The functions which must be performed by a graphics system in order to produce a realistic image of a three-dimensional polygonal model are by now well understood. Figure 1 gives a pictorial representation of the operations which must be performed in transforming a database into an image on the screen. We may envision data as being pipelined through the various operations from the model data structure onto the display. The two ends of the pipeline represent the scene in object coordinates (the model) and image coordinates respectively. In the former case, the representation is more compact, due both to hierarchical encoding of the data and to the fact that at even the lowest levels of the hierarchy the geometric primitives used are polygons or higher-order parametric surfaces. The representation of the scene in image coordinates must ultimately be reduced to points, which are the lowest-level and least compact geometric primitives available for representation of complex scenes.

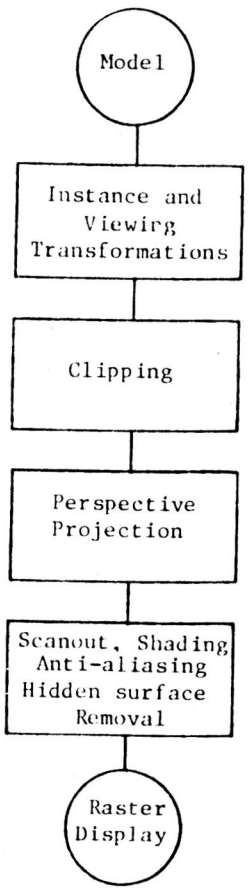

Figure 1

Image generation process

We can partition the operations being performed during the image synthesis process into two general classes. The first is a transformation from a model in object coordinate space to an image coordinate space description and the second is the reduction of a structured collection of high-level modeling primitives to a set of low-level output primitives, which for the case of raster displays are points. Instance transformations and viewing transformations are devoted primarily to the first task of coordinate system transformation (although instance transformations also serve to decode the structure of the model to form a simple collection of modeling primitives), while scanout, shading, and anti-aliasing are involved in the conversion from modeling to output primitives. Clipping may be considered to fit into the first category since it is peformed on high-level primitives as part of the transformation to an appropriate image-space representation. Hidden-surface removal can be considered an independent operation since it can be performed in conjunction with either process, although for the implementations we are interested in, it is intimately related to the primitive reduction process.

Hardware approaches to high-performance image generation have handled these two classes of operations independently, and the architecture we propose will follow this tradition. Both high-performance raster and vector display systems have long employed special-purpose array processing modules for doing the matrix-vector multiplications required for coordinate transformation as well as hardware clipping devices. The advent of VLSI technology has made possible the creation of special-purpose chips to perform the same operations at a lower hardware cost and with greater reliability. The Geometry Engine of Clark handles both of these functions with a set of 12 virtually identical MOS chips in current technology, which could be reduced to a single chip in the near future [1]. This system can currently process approximately 1000 polygons in real time, and with the projected reductions in size and concomitant increases in speed, this should increase to about 4000.

The second class of operations involving primitive reduction has been somewhat more difficult to handle at real-time rates and has therefore contributed more to the complexity and expense of real-time image generation systems. This is easily seen when the cost of a typical real-time vector display is compared with that of a raster display of comparable performance, since the primary distinction between the two in terms of hardware requirements arises as a result of the absence of most operations of this type. Most VLSI-oriented approaches to solving this

portion of the image synthesis problem have begun with the idea of enhancing the capabilities of a frame buffer in order to allow it to tackle problems of this class [2] [7] [8] [17]. The work of Cohen [3] is a notable exception to this.

The architecture proposed here is based on the same idea, to take advantage of the opportunity to create custom chips in order to add processing capability to the previously passive memory function of the frame buffer. If we assume that each element of a frame buffer need not be merely a memory cell, that we have the freedom to make the unit more complex, endowing it with the capability to implement for itself some operations which must be done for image generation in parallel with all the other cells of the frame buffer, then the question arises as to whether it might not be better to make each unit a higher-level primitive than a mere point. This is the approach we take, which is distinct from those noted above, in which the primitives remain points but the points are provided with added processing power. In our system, the "frame buffer" consists of a collection of polygons rather than points. For simplicity and uniformity of implementation, the polygons are required to be triangles. The advantages of this requirement outweigh its limitations. Since any polygon can be easily triangulated, any scene described by a collection of polygons can be easily transformed into a description consisting only of triangles. In the process, any "non-planar polygons" which may have existed in the original scene description are removed. Moreover, procedures which are used for automatic generation of scene descriptions from real-world input data typically generate only triangles [5]. Finally, Gouraud shading, the most suitable smooth-shading technique for fast hardware implementation, produces no shading anomalies on triangles.

It makes sense to call this collection of triangle processing elements a frame buffer because it performs an analogous function in the graphics system to that of a traditional frame buffer in that it serves as a medium for storing the scene description after it has been transformed into image space. Of course, no reduction of high-level to low-level primitives has been performed before the image space scene description is stored, so this function must be performed by the frame buffer itself. From the point of view of the refresh controller for the raster display, the "triangle buffer" appears to be a frame buffer in that the controller outputs addresses to it on an address bus and receives in return a pixel's color value on a data bus. Each triangle processor performs a scanout of the triangle it contains, incrementally determining the color and Z coordinate value for each pixel. These are fed through arbitration logic which determines which pixel is closest to the observer and returns the color of that pixel on the data bus. At the cost of added complexity, the arbitration logic can also be used to perform anti-aliasing, resulting in a filtered pixel color value being returned on the data bus. Thus the triangle buffer performs all operations involved in the reduction of high-level to low-level primitives.

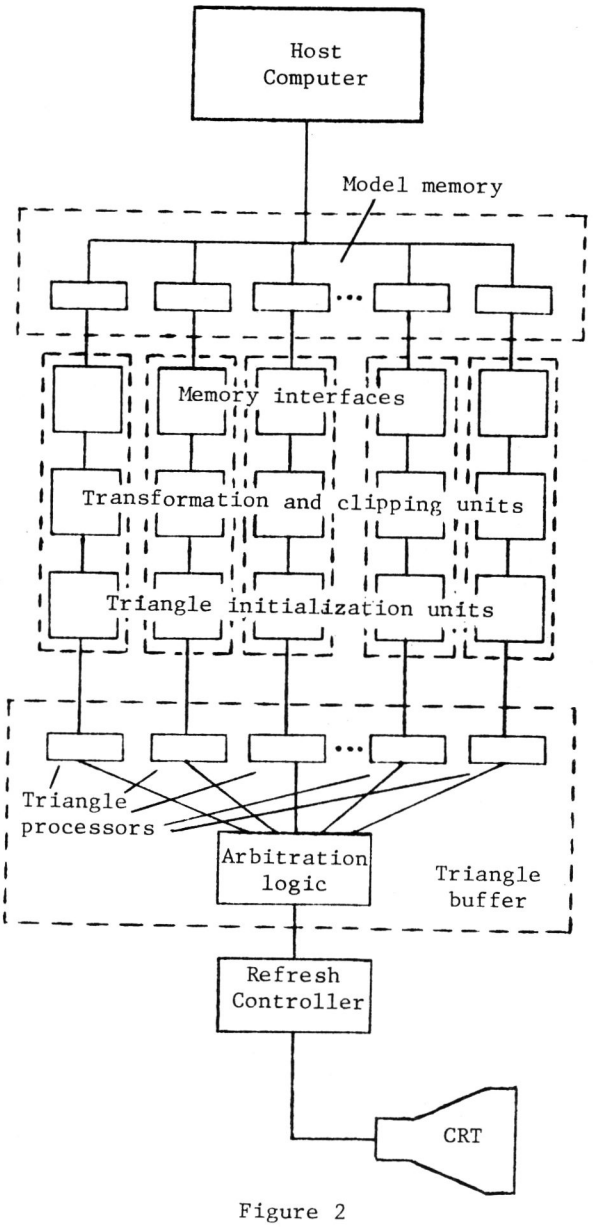

Figure 2
System organization

The remainder of the system implements the coordinate transformations, clipping, perspective division, and shading calculations for the vertices. It also contains a dual-ported memory for storing the object-space scene description.

3.0 SYSTEM ORGANIZATION

In this section we describe in more detail the organization of the major portions of the system. The overall organization is depicted in Figure 2. Our discussion will begin with a consideration of the triangle buffer, following which the modeling memory and transformation pipeline are described.

3.1 The Triangle Buffer

The triangle buffer consists of an array of triangle processing units, each of which performs scanout and smooth-shading of the triangle it contains. A triangle processor could be considered a smart, dual-ported memory "cell". It is connected on the one hand to a bus over which it receives information from a triangle initialization unit, and on the other hand to a comparator tree onto which it outputs color and Z coordinate values. It is also connected to an address bus through which the CRT refresh controller indicates the X-Y address of the current pixel. Since the data bus is write-only from the transformation and clipping unit into the triangle buffer, no corresponding address bus is required. When polygon data is output to the triangle buffer, it is simply accepted by the first free triangle processor available. The advantage of this scheme is that it simplifies the interface to the transformation and clipping units. The disadvantage is that it requires every polygon in the scene to be rebroadcast to the triangle buffer at every refresh cycle. The overall collection of triangle processors in the buffer is partitioned into a set of "slices", each containing the number of triangles which can be processed in real time by a single transformation, clipping, and triangle initialization pipeline. In our system this number is estimated to be about 1000.

3.1.1 The Triangle Processor -

Each triangle processor consists of 20 registers, with associated addition, comparison, and control logic, as shown in Figure 3. Each processor performs a simple scanout of the triangle, based on an identification of the

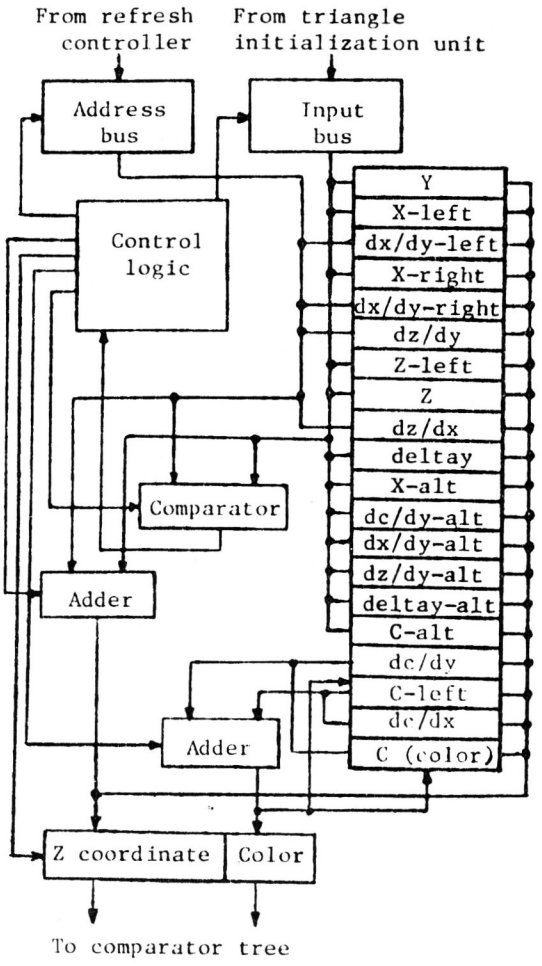

Figure 3

Block diagram of triangle processor

initial left and right boundary edges and the third "alternate" edge by the triangle initialization module (see Figure 4). The processor monitors the address bus until a Y address equal to the value stored in the Y register is sent. If then monitors the X address until it reaches a value equal to X-left, at which time it outputs the initial color and Z coordinate. For each subsequent pixel, the color is incremented by dc/dx and the Z coordinate value by dz/dx and the new values are output. When the X address reaches X-right, the output is halted and the next scanline is prepared by adding dx/dy-left, dx/dy-right, dz/dy and dc/dy to X-left, X-right, Z-left, and C-left respectively. The values of Z and C are then set equal to Z-left and C-left and the value of deltay is decremented by 1. This process continues scanline by scanline until deltay

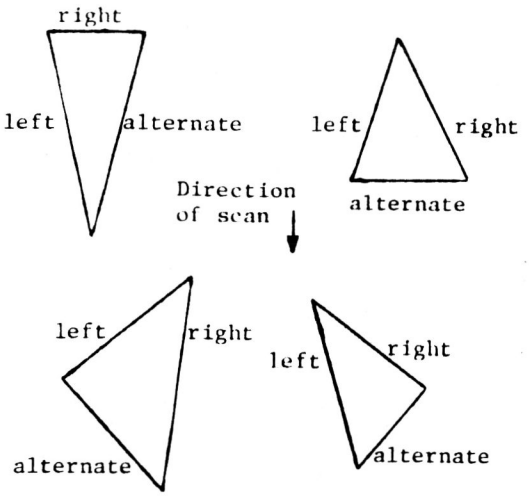

Figure 4

Assignments of triangle edges

reaches 0. At this point, X-alt is compared to X-left and X-right to determine whether the left left or right edge should be replaced by the alternate edge. The value of dx/dy-alt, is then copied into dx/dy-left or dx/dy-right as appropriate. In the former case, the values of dz/dy-alt and dc/dy-alt are copied into dz/dy and dc/dy respectively. deltay is set to the value of deltay-alt, and the processing continues until deltay once again reaches zero. At this point output is disabled, and the triangle processor enables itself for input of a new triangle. Note that this scanout procedure for triangles is similar to that of [12] and [15]. The use of trapezoids in which the top and bottom edges are horizontal, as done in [10] and [16], would allow a reduction of 5 registers and some simplification of the control. However, this would require the polygons in the scene model to be subdivided into such trapezoids each time they are transformed. By maintaining the scene model as a set of triangles, we totally avoid the need for polygon subdivision at each refresh cycle. All operations of the triangle processors are fixed-point. Operations on colors are assumed to be in RGB space, with the addition unit actually performing parallel operations on each of the three components. The use of separate, fast adder units for color and Z coordinate calculations assures that new values of each can be generated at the required bandwidth.

3.1.2 Arbitration Logic -

The triangle processors resemble the object processors proposed by Cohen [3] and Weinberg [16]. A significant difference in our approach is that we do not serialize the operation of these processors using a linear array of comparators. Instead, we use a binary tree of comparators, through which the depth and color values output by the triangle processors are pipelined. As a result, all triangle processors are equidistant from the endpoint of the pipeline and thus can operate on the same pixel simultaneously. This simplifies the synchronization and control of the system, and indeed allows the triangle buffer to resemble a frame buffer in its function. The complexity and number of comparators required is the same in both cases. Their operation is straightforward; they compare the two input Z values, and the closer of the two to the observer is output to the next pipeline stage along with its associated color. The root of the comparator tree outputs the color of the visible portion of the scene at the current pixel to the CRT refresh controller.

If anti-aliasing is desired, it is possible in principle to employ the method proposed by Weinberg [16], modified to work on a binary tree rather than a linear array of comparators. This method involves scanout of polygons at subpixel resolution so that the degree of coverage of the current pixel by each triangle can be determined. The comparators then maintain a list of partially visible polygon sections for the current pixel, sorted by Z. The last comparator outputs the list to a filter processor which calculates the appropriate color for the pixel. This algorithm requires that each comparator be able to merge two sorted input lists, which is no more complex as a pipelined algorithm for a binary tree than for a linear array. However, in addition to the added complexity of the comparator unit, this approach has the serious drawback that it builds a list which can grow as it passes through the comparator pipeline. Thus more values must be passed through later stages of the pipeline than through earlier ones, requiring that the bandwidth of the pipeline increase with proximity to the output end, or else that synchronous operation of the system be fatally disrupted. This drawback holds for a linear array of comparators as well as for a binary tree. As a result, we have chosen not to use this approach. For simplicity, the current system design does not include anti-aliasing, although we are investigating methods which can be used without unduly complicating the operation of the triangle buffer.

3.2 Modeling, Transformation, And Clipping

As mentioned in an earlier section, the object space representation of the scene is maintained in a separate, dual-ported memory. This memory contains a set of triangles, each of which consists of a color and a sequence of vertices described in world coordinates. In addition, each vertex is associated with a normal vector, which is the average of the normals to all polygons adjacent to the vertex, in order to perform smooth-shading. All coordinates are stored as floating-point numbers.

This memory is attached to the system bus of a host computer on the one hand, and to a set of transformation processors on the other. The host computer may be a general-purpose machine or a dedicated graphics processor. It is responsible for all manipulations of the scene database, for interacting with the user and for execution of all application software. If a hierarchically-structured model is used as a source database, this processor must decode the structure for use by the display system. It is also responsible for controlling the motion of the observer and of the objects in the scene. This involves specifying transformations to be performed on designated sets of polygons in the scene description. The transformations are encoded in the form of standard 4x4 homogeneous matrices. Only a single such transformation is performed by the transformation processing units on a triangle, so all concatenation of instance and viewing transformations must be done by the host processor.

The organization of the model memory and its interface to the transformation processors is shown in Figure 5. Each transformation unit communicates over a local bus with only as many polygons as it can process in real time, along with a set of transformation matrices which allow the set of polygons to be segmented into disjoint sets of objects. Each such matrix is associated with a pair of registers in which the identifiers of the first and last polygon in the corresponding object are stored. Thus an object is a contiguous sequence of polygons on the local bus. If a transformation pipeline can process 1000 triangles in real time, then about 23,000 words of memory will be accessible on each local bus if we allow 10 independent objects. In this way large numbers of triangles can be processed in parallel slices of 1000 triangles.

The transformation pipeline consists of an interface to a local model memory bus, a matrix-vector multiplier, a sequence of clipping units, and a triangle initialization unit, as depicted in Figure 5. The interface functions to access the object memory, load matrices into the

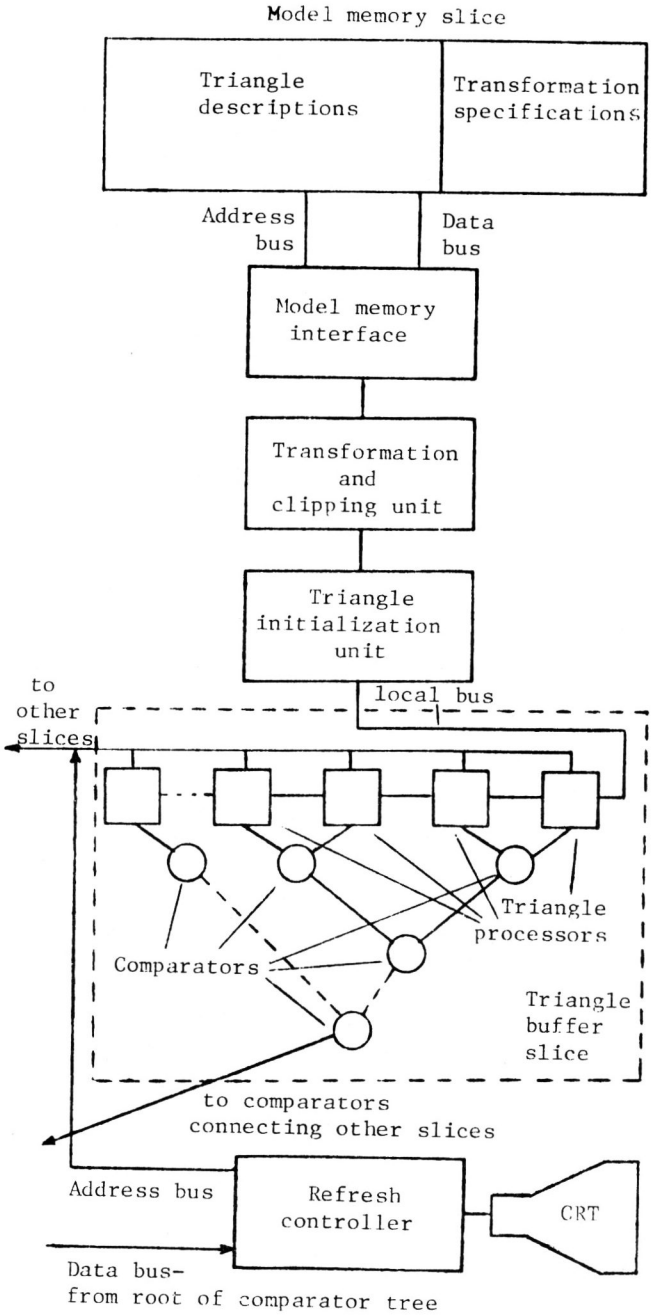

Figure 5

Detail of system slice

transformation processor, load clipping parameters into the clipping unit, and feed polygons into the pipeline. The transformation unit does matrix-vector multiplication, either as a systolic linear array [4] or as done by Clark [1]. The clipping pipeline consists of six identical units, each of which clips a polygon against a plane of the truncated viewing pyramid. The triangle initialization unit is primarily a division/subtraction engine which calculates the color for each vertex, performs perspective division, sorts polygon vertices by Y and then X, and then calculates the parameters of the triangle to be loaded into the triangle buffer. It also converts the floating-point values used by the transformation and clipping processors to fixed-point format for use by the triangle processors. Note that each triangle initialization unit communicates with only the triangle processors in a slice of the triangle buffer via a local bus analogous to that used to interface with the object memory.

4.0 VLSI IMPLEMENTATION

The system described here is a very large-scale project which requires the design of a number of special-purpose units. The dual-ported object memory is available commercially, but the other parts of the system must be custom designed. As stated before, the goal of the system is to allow real-time display of scenes consisting of at least 25,000 triangles. This implies 25,000 triangle processors and an equal number of comparators, and an estimated 25 transformation pipelines, along with approximately 575,000 words of object memory. To build such a system at a reasonable cost it will be necessary to make use of all the capabilities of today's VLSI design and fabrication technology. If we optimistically assume one micron feature sizes, we can reasonably estimate that two chips will suffice for each transformation pipeline and that perhaps 32 triangle processors and an equal number of comparators will fit on a large chip. With these assumptions, implementing a 32,000 triangle system will require 1000 triangle processor chips, 1000 simple comparator chips, 64 transformation pipeline chips, and 736,000 words of dual-ported memory. At a cost of $100 per chip, we obtain $264,000 for special-purpose chips, so we can generously estimate a $500,000 cost of goods for the system and thus a selling price in the neighborhood of $1 million. This is comparable to the cost of existing systems which provide an order of magnitude less performance.

The design of this type of system is certainly a non-trivial task, particularly in a university environment. We view this overall organization as a rich source of design projects which can be implemented and tested independently, with the object of consolidating them into a small-scale working prototype. The comparator unit, the simplest independent subsystem, has been designed and is currently being prepared for fabrication. The design of the triangle processor is currently underway, and the various parts of the transformation pipeline will be undertaken subsequently.

5.0 CONCLUSION

We have described a high-performance hardware organization suitable for real-time display of complex three-diminsional scenes. It is primarily intended to provide an order of magnitude increase in the capability of high-performance systems used for such applications as flight simulation. These goals are distinct from those of most other researchers who have applied the opportunities afforded by the availability of custom-designed integrated circuits to the design of graphics display hardware. The complexity of the system makes it more ambitious and its realization a more distant prospect than these other design efforts. Nonetheless, it illustrates at yet another level the exciting prospects made available to the field of computer graphics by recent advances in VLSI technology.

Acknowledgements

The authors would like to thank Sanjay Deshpande for his contributions to the development of this organization.

References

[1] Clark, J.H., A VLSI Geometry Processor for Graphics, Computer, July, 1980.

[2] Clark, J.H. and M.R. Hannah, Distributed Processing in a High-Performance Smart Image Memory, LAMBDA, vol.1, no.3, 1980.

[3] Cohen, D. and S. Demetrescu, A VLSI Approach to Computer-Generated Imagery, Technical Report, USC, 1979.

[4] Foster, M.J. and H.T. Kung, The Design of Special-Purpose VLSI Chips, Computer, January 1980.

[5] Fuchs, H., Z.M. Kedem, and S.P. Uselton, Optimal Surface Reconstruction from Planar Contours, Communications of the ACM, vol. 20, no. 10, October 1977, 693-702.

[6] Fuchs, H. and B. Johnson, An Expandable Multiprocessor Architecture for Video Graphics, Proceedings of the Sixth Annual ACM-IEEE Symposium on Computer Architecture, April 1979.

[7] Fuchs, H., J. Poulton, A. Paeth, and A. Bell, Developing Pixel-Planes, a Smart Memory-Based Raster Graphics System, Proceedings, Conference on Advanced Research in VLSI, Massachusetts Institute of Technology, January 1982, 137-146.

[8] Gupta, S., R. Sproull, and I. Sutherland, A VLSI Architecture for Updating Raster-Scan Displays, Computer Graphics, vol. 15, no. 3, August 1981, 71-78.

[9] Kaplan, M. and D.P. Greenberg, Parallel Processing Techniques for Hidden Surface Algorithms, Computer Graphics, vol. 13, no.2, August 1979, 300-307.

[10] Jackson, J.H., Dynamic Scan-Converted Images with a Frame Buffer Display Device, Computer Graphics, vol. 14, no. 3, July 1980, 163-169.

[11] Mead, C. and L. Conway, Introduction to VLSI Systems, Reading, Addison-Wesley, 1980.

[12] Myers, A.J., An Efficient Visible Surface Algorithm, Technical Report, Computer Graphics Research Group, Ohio State University, July 1975.

[13] Parke, F.I., Simulation and Expected Performance of Multiple Processor Z-Buffer Systems, Computer Graphics, vol. 14, no. 3, July 1980, 48-56.

[14] Roman, G. and T. Kimura, A VLSI Architecture for Real-Time Color Display of Three-Dimensional Objects, Proceedings of the Delaware Valley Microprocessor Conference, April 1979, 113-118.

[15] Romney, G.W., Computer Assisted Assembly and Rendering of Solids, Technical Report 4-20, Department of Computer Science, The University of Utah, 1970.

[16] Weinberg, R., Parallel Processing Image Synthesis and Anti-Aliasing, Computer Graphics, vol. 15, no. 3, 55-62.

[17] Whitted, T., Hardware Enhanced 3-D Raster Display Systems, Canadian Man-Computer Communications Conference, June 1981.