Three Dimensional Graphics Algorithms on the Micro-Grain Array Processor-II

Benjamin Bishop *Yan Zhang* *Kevin Acken*
Mary Jane Irwin *Robert M. Owens*

Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802

Abstract

Three-dimensional graphics processors play a central role in many modern computer systems due to the large demand for entertainment, visualization tools, architectural CAD tools, and physical modeling. But high performance graphics subsystems are just beginning to reach the commercial workstation and personal computer markets. They also remain high priced and often lacking the performance needed for graphics-intensive applications. The root of the problem lies in the complex graphics algorithms that require large amounts of object data to be manipulated with high throughput, such as what is required for ray tracing. This has resulted in complex, real-time 3D graphics to be limited to high-end systems. In this paper, we present a library of 3D graphics algorithms that have been mapped to the Micro-Grain Array Processor (MGAP), an inexpensive and versatile SIMD processing board capable of fitting in a typical workstation. Our results show that the MGAP can produce comparable data throughput as more costly graphics subsystems, while maintaining the flexibility of being a general purpose parallel machine.

1: Introduction

The increasing demand for three dimensional graphics on low end workstations and personal computers has brought on the need for inexpensive architectures capable of meeting the high throughput requirements of a graphics subsystem. In order to support real-time 3D graphics, the subsystem will need to implement the transformation and rasterizations stages of the graphics pipeline [8] in hardware. In the transformation stage, the vertices defining the objects to be displayed are transformed to screen pixel coordinates, typically using a matrix stack of affine transformations within a homogeneous space. The rasterization stage receives the transformed vertices, and converts them into pixels in the frame buffer either by spanning the polygons or by ray tracing. The transformation and rasterization stages can be expected to operate on millions of polygons per second in order to obtain real-time performance within complex graphics applications. This has lead graphics subsystems to include parallel Multiple-Instruction Multiple-Data (MIMD) and Single-Instruction Multiple-Data (SIMD) architectures within their designs. MIMD architectures have the advantage of high processor utilization, but require complex control and inter-processor routing. SIMD architectures have lower processor utilization, but can contain many more processors due to simplier interconnect and control. In addition, SIMD architectures generally take up less area, making them well suited for workstation environments, and are less expensive than MIMD machines. Pixel-Planes [4] and PixelFlow, designed at the University of North Carolina Chapel Hill, demonstrated the effectiveness of using an SIMD architecture for graphics rendering.

In this paper, we present the mapping of transformation and rasterization algorithms to the Micro-Grain Array Processor (MGAP). The MGAP is capable of producing the high throughput required for 3D graphics applications, while still being a small and inexpensive workstation board. The MGAP also has the advantage of being able to be dynamically reconfigured, allowing the same array to operate at different stages of the graphics pipeline. Clark [2] showed the advantages of using one architecture for multiple stages of a graphics pipeline with the *Geometry Engine* processor. In section 2, the architecture and history of the MGAP will be described. Section 3, 4 and 5 will present the mappings of the transformation, rasterization, and ray tracing algorithms respectively, and section 4 will summarize the results and conclude the paper.

2: The Micro-Grain Array Processor

The Micro-Grain Array Processor (MGAP) family of array processors is a SIMD array of fine-grain processors developed by the Microsystems Design Lab at Penn State University [7]. The MGAP-II is a workstation coprocessor board holding a 32 custom designed chip array containing a total of 49,152 processing elements(PEs), along with data staging memory and control logic for the array. Each PE contains a 32 bit local memory along with two functional MUXes, each capable of performing any three-to-one boolean function every 20 nanoseconds. Each PE is connected in a nearest neighbor fashion to its north, south, east, and west neighboring PEs. Figure 1 shows a block diagram of one PE. The unique and powerful feature of the MGAP architecture is that each PE can be individually configured to receive data from any of its neighbors independent of the configuration of any other processor. The PE configuration process is completely dynamic allowing the MGAP to be reconfigured on the fly. This *connection autonomy* allows PEs to be grouped into larger, "virtual" processors called *word* processors, which allows for arbitrary precision arithmetic and data flow on the array. The MGAP architecture has be shown to be highly effective at performing signal and image processing applications [1, 6]. Table 1 outlines the generations of the MGAP family.

Table 1: Comparison of the MGAP Family of Array Processors

	Date	Tech.	Chips	PEs	mem/PE	Clock	TeraOps
MGAP-I	Fall 1993	1.2 CMOS	32	16,384	16 bits	25 MHz	0.8
MGAP-II	Fall 1996	0.8 CMOS	32	49,152	32 bits	50 MHz	6.4
MGAP-III	?	MCM	?	262,144	64 bits	100 MHz	51.2

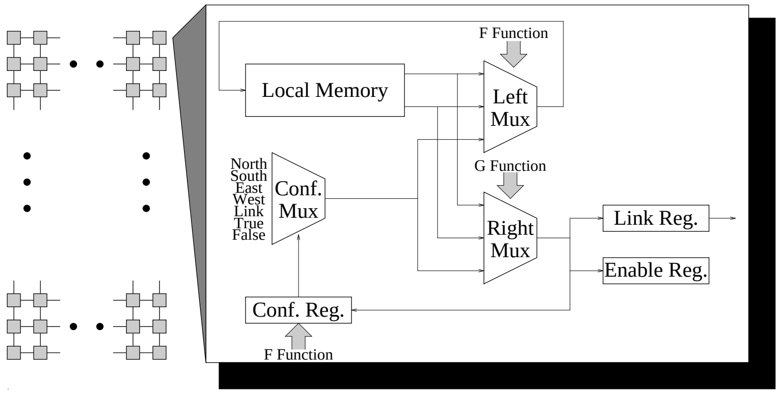


Figure 1: Processing Element of the MGAP-2

3: Transformation Algorithm

The transformation stage of a graphics pipeline is responsible for performing affine transformations on the vertices of the objects that are to be displayed. This operation requires the product of a 4 x 4 transformation matrix with the 4 x 1 vertices, which in itself is not expensive, but this may have to be repeated over hundreds of thousands of vertices per screen update. The standard affine transformations includes:

- Translating(T) an object in X, Y, or Z
- Scaling(S) the size of an object
- Rotating(R) an object about the X, Y or Z axis
- Projecting(P) an object to the screen

This means that the more detailed the object, the slower the transformation process. On the MGAP, the point vectors of an object can be distributed across different processors and thus the product of the transformation matrix and the vectors can be performed simultaneously on the array.

Each 3D vector will be represented by (x, y, z), where each coordinate has a precision of sixteen bits. Four MGAP-II PEs can be grouped into a word processor in a 2 by 2 fashion. The coordinates x, y and z are stored in three of the four PEs, and the fourth PE serves as temporary storage locations. The multiplication takes place in each PE and the summation of the products is generated in the fourth PE. In addition, four such word processors can be combined into a group of 16 PEs in a 4 by 4 fashion (Figure 2). The transformation matrix is broadcast to all the PEs. Each word processor will multiply one of the four rows of the transformation matrix with the vector. The group for four word processors will then produce the transformed point vector.

X	Y	X	Y
Z	T	Z	T
X	Y	X	Y
Z	T	Z	T

Figure 2: The transformation configuration of the MGAP-II PEs

The transformation matrix M consists of a 4 x 4 matrix of elements m_{ij}. Thus to transform each vertex, the following equation will need to be performed:

$$v_i = m_{i0} * x + m_{i1} * y + m_{i2} * z + m_{i3}, i = 0, 1, 2, 3 \tag{1}$$

Due to the limited local memory within each PE, the multiplications will be performed in a bit-pipelined fashion, producing one bit of partial product each cycle. The accumulation of these four bits will be calculated in the fourth PE, which serves as temporary storage locations. Figure 3 shows this multiply-accumulate operation, which would take four cycles per partial product bit add.

Each MGAP PE contains 32 bits of local memory, of which 16 bits are taken by the vertex coordinates. Since the transformation matrix also needs to be stored in PEs as well, the precision of the matrix elements was reduded to eight bits instead of a full sixteen bits. This allows all calculations to be performed in the 2 x 2 word processors. This choice will limit the range of the transformations, however higher precision transformations can always be performed by multiplying successive transformation matrices. An alternative solution is to configure word processors into 3 x 2 PEs. This method will make the tradeoff between the

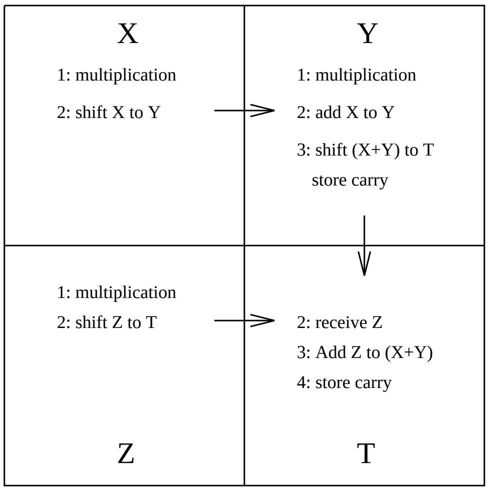


Figure 3: The bit-pipelined accumulate operation takes four cycles on the MGAP.

precision of the calculation and how many vertices can be operated on in parallel. Table 2 shows the instruction counts and estimated performance comparison of the two choices in transformed triangles per second. It can be seen that the choice of 2x2 word processors with 8-bit matrix elements makes more sense since it is more speed efficient. When 16-bit precision transformations are required, the disjoint triangles/sec will be cut by half and become 3.125 million, which is still compatible with the performance of the other option with slightly lower performance. However, on average, if assume half of the transformations need 16-bit precision, this choice will perform better than the other one.

Table 2: Vertex transformation on the MGAP using a low and high precision matrix

Operation		2 x 2 word proc. (cycles)		3 x 2 word proc. (cycles)
Shift Vectors on Array	(16*192)	3072	(16*192)	3072
Shift Matrix on Array	(8*192)	1536	(16*192)	3072
Multiply-Accumulate	(16*8*4)	512	(16*16*4)	1024
Shift Vectors off Array	(16*192)	3072	(16*192)	3072
Total Instructions		8192		10240
Estimated Time/Array		0.164 msec		0.205 msec
Vectors/Array	$(\frac{256}{4} * \frac{192}{4})$	3072	$(\frac{256}{4} * \frac{192}{6})$	2048
Vectors/Sec		18 million		9.9 million
Disjoint Triangles/Sec		6.25 million		3.33 million

4: Rasterization Polygon Spanning Algorithm

The goal of the rasterization stage is to convert the transformed vertices into pixels within the frame buffer. One such method that is popular in many commercial systems is to raster the polygons by *spanning* the vertices to determine which pixels are contained in each polygon. Pixel color and shading is also determined during this spanning process. Hidden surface removal can be accomplished through the use of a z-buffer, which determines the closest viewable polygon in each pixel location. On a SIMD array, this can be accomplished by defining the array as pixel processors, and successively broadcasting the edges that define the polygon to the processors. Each processor determines if it lies to the inside or outside of the edge. If it lies outside the edge, the pixel processor disables itself knowing that it is not part of the polygon. After all of the edges of a polygon are broadcast, the enbaled processors represents the pixels within the polygon. Next, z-depth and color can be determined for each enabled pixel based on its coordinates. Hidden surface removal is accomplished by separate registers in each processor that keeps track of the current closest polygon to the viewer.

The MGAP array will be configured to form word processors that correspond to pixel processors. Each word processor contains four PEs and holds the (x, y) coordinates of the pixel location that it represents. These coordinates are stored as sixteen bit integer values. Each word processor also keeps track of its current pixel value in the form of an eight bit RGB color and an eight bit z-buffer depth value. Figure 4 shows this configuration. This allows for a 128 x 96 pixel block to be processed on the MGAP-II array at one time. Larger graphics can be generated by breaking up the image into 128 x 96 pixel blocks and operating on each block separately, swapping the results off the array after each block is completed. The MGAP-III would allow for a 256 x 256 pixel block to be processed on the array.

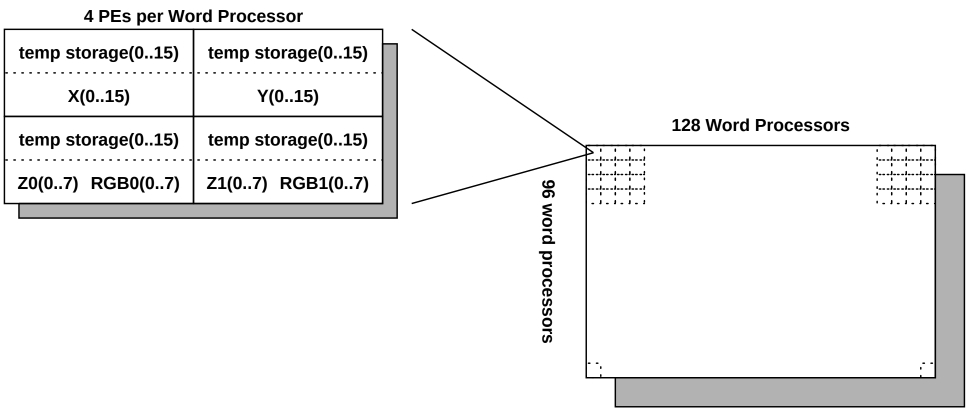
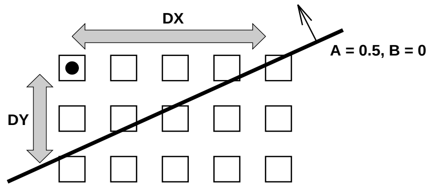


Figure 4: PE organization for spanning polygons during rendering stage.

In the MGAP algorithm, the pixel processors are first initialized with a sixteen bit coordinate of their pixel location. Next each edge of the transformed polygon is projected to two-space and parameterized as an eight bit slope an intercept. These values are broadcast at the bit level to the pixel processors, and each processor determines if it is over or under the edge. Next, depending on the normal of the edge, all the pixels that are either over or under the edge are disabled. Precalculated depth and color planes are then individually broadcast over the array, and the enabled processors determine their z-depth, and RGB color values. Finally, the calculated z-depth of each enabled pixel processor is compared with the current closest known pixel value. If the new pixel is closer, it either can replace the old values or, in the case of transparent objects, it can be blended with the old values. This algorithm is shown in figure 5.



For Each Polygon

- **For Each Edge**
  - **Broadcast A, B**
  - **Each Pixel Calculates DX, DY**
  - **If (DX > 0) or (DY > 0) DISABLE**
- **Broadcast Z-Depth Plane, Color Planes**
- **Each Enabled Pixel Calculate Z, RGB**
- **Compare with Closest Z-depth**

Figure 5: Algorithm for spanning polygons on MGAP.

Tbale 3 gives the estimated instruction count and performance. This method of using the MGAP as a pixel array has the advantage that the time required to span a polygon is independent of its size. The disadvantage of this design is that the array will generally see a low level of processor utilization, especially if the drawn objects are made of many small polygons.

Table 3: Triangle Rasterization on MGAP

Operation		cycles	
Calculate DX		136	
Calculate DY		136	
Disable External Pixels		2	
Cycles/Triangle	(3*274)	822	
Calculate Z-Depth		272	
Calculate RGB	(3*272)	816	
Compare Min and Store		40	
Cycles/Spanned Tri.		1950	
Estimated Time		0.039	msec
Triangle/Sec		25641	

5: Rasterization Ray Tracing Algorithm

A simple ray tracing algorithm can be efficiently implemented on the MGAP-II through the use of a simple, highly parallel algorithm with scaled integer arithmetic. The algorithm involves finding intersection information for a series of rays for a fixed number of faces. For each ray, the algorithm looks for the closest face in which the intersection is some maximum distance from the center of the face. When this is determined, the intensity for a given ray can be determined using an approximation of the diffused light model [3]. This model assumes that incoming light is reflected evenly at all angles, which gives the appearance of a rough surface. This intensity information can then be used to update the viewing area on the host machine.

In order to map the algorithm efficiently to the MGAP-II, a number of implementation choices are required. The algorithm dictates that a large number of registers are required, for example, the dot product consumes at least 7 registers (not including temporary values). Since a large number of registers are desired in each computational unit, each operand is represented as a row of processors 16 bits wide(Fig. 5). This yields 32 available 16 bit integer registers for each computational unit. It would speed computation if these numbers were represented in a redundant number system, however, this would consume more storage area and defeat the goal of a large register set.

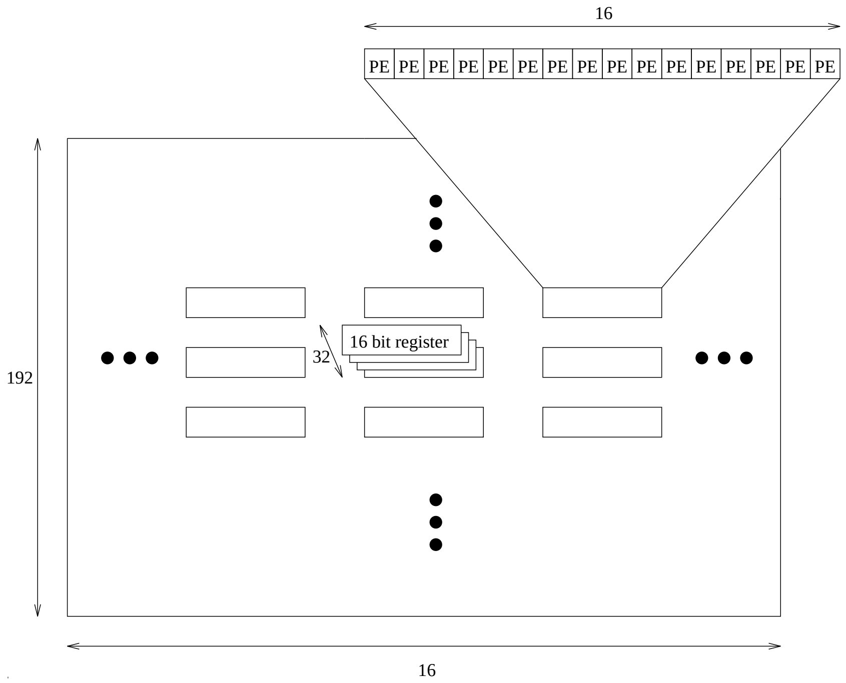


Figure 6: PE organization for ray tracing algorithm on the MGAP-II.

Due to the nature of this architecture, it is advantageous to create parallel pipelines of computational units, such that each pipeline contains the same face information. A different series of light rays are then passed through each pipeline. This is done to avoid any across-pipeline compare that would be necessary at the end of the pipeline if the same series of rays were passed over several pipelines containing different faces.

In each stage of the pipeline, a number of computations are performed. First the ray is checked to determine if it intersects the given face. This is done by finding the intersection of the ray and the face plane then adding the differences of each element of the intersection and face center vectors. If this sum is less than some maximum distance approximation then the intersection lies in the face. Note that this distance approximation yields a non-circular face shape. Secondly, the distance from the viewer to the intersection is compared with the closest distance so far. If this intersection is closer than the best so far, information for the current face is later passed to the next computational unit. Next, the intensity for the current ray is computed using a diffused light model. Lastly, if the current intersection is the nearest so far, the current intensity and intersection distance are passed along with the ray information, otherwise the previous results are passed.

The estimated performance of this ray tracing algorithm is given in table 4. Using this approach and without calculating lighting on the array a 320x200 image could be ray traced in approximately 18 second.

Table 4: Low-level ray tracing algorithm statistics

Operation	cycles
Add/Subtract	289
Multiplication	6096
Division	8418
Comparison	223

Table 5: Ray tracing overall statistics for 256 face object

Pixels/Sec	3608
Time for 320x200	18 sec

6: Performance Summary and Conclusion

In this paper, we presented mappings of 3D graphics algorithms to the MGAP-II. The results obtains are summarized in table 6.

Within the transformation stage, we found that configuring groups of four processors with lower matrix precision proved to be more efficient than grouping six processors with

Table 6: Performance Summary of Graphics Algorithms on the MGAP-II

stage	algorithm	performance	
Transformations	Low Precision	6.25M	triangles/sec
	High Precision	3.33M	triangles/sec
Rasterization	Spanning	25K	triangles/sec
Rasterization	Ray Tracing	18 sec	for a 320 x 200 scene

full precision. It was also found that configuring the MGAP-II as pixel processors for the rasterization stage resulted in a low utilization of processors, though we still received reasonable performance. The results also show that ray tracing algorithms can be adapted to general purpose SIMD machines like the MGAP-II. Future work includes using the MGAP to implement a full 3D graphics pipeline, as well as designing a complete set of rendering algorithms for the MGAP-II.

References

[1] R.S. Bajwa and R.M. Owens, "Image Processing with the MGAP:A Cost-Effective Solution," *Proc. of IPPS*, April 1993.

[2] J. Clark, "The Geometry Engine: A VLSI Geometry System for Graphics", *Proc. SIGGRAPH 82*, 1982, pp. 127-133.

[3] Foley, van Dam, Feiner, and Hughes, *Computer Graphics Principles and Practice*, Addison-Wesley, 1990.

[4] H. Fuchs, J. Poulton, A. Paeth, and A. Bell, "Developing Pixel-Planes, A Smart memory-Based Raster Graphics System", *Proc. 1982 Conference on Advanced Research in VLSI, MIT*, pp. 137-146.

[5] F.S. Hill, *Computer Graphics*, Macmillan Publishing Company, 1990.

[6] H.N. Kim, M.J. Irwin, and R.M. Owens, "MGAP Applications in Machine Perception", *Proceedings of Computer Architectures and Machine Perception*, Sept. 1995.

[7] M.J. Irwin, R.M. Owens, "A Micro-Grained VLSI Signal Processor", *Proceedings of ICASSP*, Mar. 1992, pp. 641-644.

[8] M. Kilgard, "Hardware for Accelerating OpenGL", *The X Journal*, Sept-Oct 1995, pp. 52-69.