

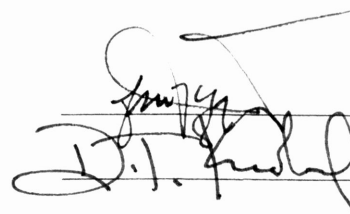
Application of a Systolic Array in a Geometry Engine for Computer Graphics

Derek T. Spears

Undergraduate Honors Fellows Program, 1989-90

Texas A&M University

Electrical Engineering Department



Fellows Advisor
Honors Director

I. ABSTRACT

The increase in size and complexity of scientific and engineering datasets has dictated the need for efficient Computer Graphics systems for visualization. Such graphics systems require high speed transformation of vertices and high speed rendering. A high performance transformation processor is proposed here for inclusion in a total graphics system. The processor will incorporate a systolic architecture and extensive pipelining to achieve high throughput.

TABLE OF CONTENTS

Chapter	Page
I. ABSTRACT	i
II. INTRODUCTION	1
2.1 Graphics Fundamentals	1
2.2 Computer Architectures	4
2.3 Systolic Arrays	6
III. DESIGN APPROACH	
3.1 Criteria	9
3.2 Systolic Array Schedule and Optimizations	10
3.3 Design	12
3.4 Internal MAC pipelining	14
3.5 Clipping	18
3.6 Rendering	23
IV. RESULTS	24
V. CONCLUSIONS AND DISCUSSION	26

I. INTRODUCTION

2.1 Graphics Fundamentals

Graphics processing can be divided into three major stages, transformation and clip, rendering, and display. In the transformation stage, the points which comprise a polygon are transformed in 3-space by a 4x4 matrix-vector multiplication. This matrix operation allows any polygon to be translated, rotated or scaled, thus allowing the user to reposition it anywhere in 3-space. The rendering stage of the processor shades the polygon based on lighting and color models and passes this data to the Display stage which contains video memory and drawing hardware which allows the polygon to be viewed on the screen. The focus of this paper is the optimization of the transformation and clip stage.

The transformation of polygons, as stated earlier, is controlled by a 4x4 transformation matrix. Translation, Rotation, and scaling operations are able to be performed with specified matrices [1]. This matrix acts upon the x, y and z coordinates of the vertices which compose the polygon. The general format is shown in figure 1. This matrix applies to all points in three-space. In addition to points, polygons require surface normals, used in lighting calculations, to be transformed also. Since only direction information is important for normals, only rotation matrices are interesting. The rotation matrix is only 3x3 and therefore normal transformations can be limited to a 3x3 matrix.

More than one transformation matrix is typically used to orient an object onto a screen. In addition the Global Transformation Matrix, the model must also have the View Transformation Matrix applied. This matrix correctly places the model within a specified view volume. If the model has a hierarchical structure as in PHIGS (Programmers Hierarchical, Interac-

$$[xyz1] = [xyz1] \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & 1 \end{bmatrix}$$

General Matrix Representation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

Translation Matrix

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling Matrix

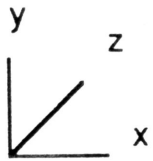
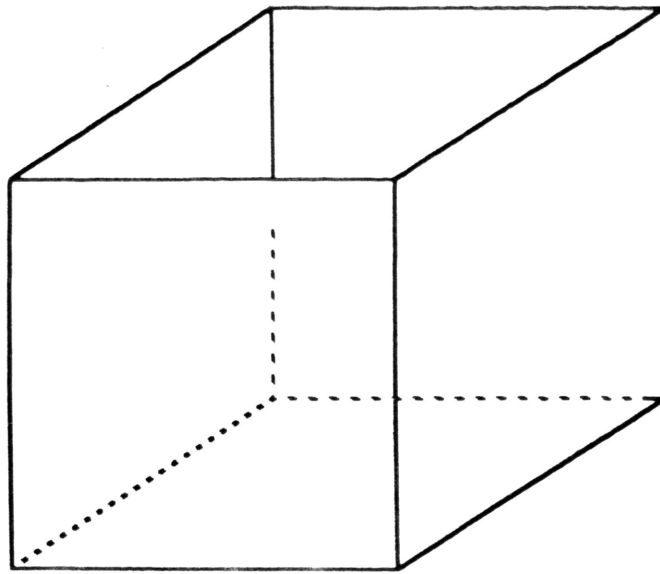
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

X, Y, and Z Rotation Matrices

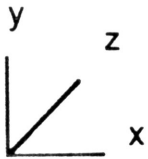
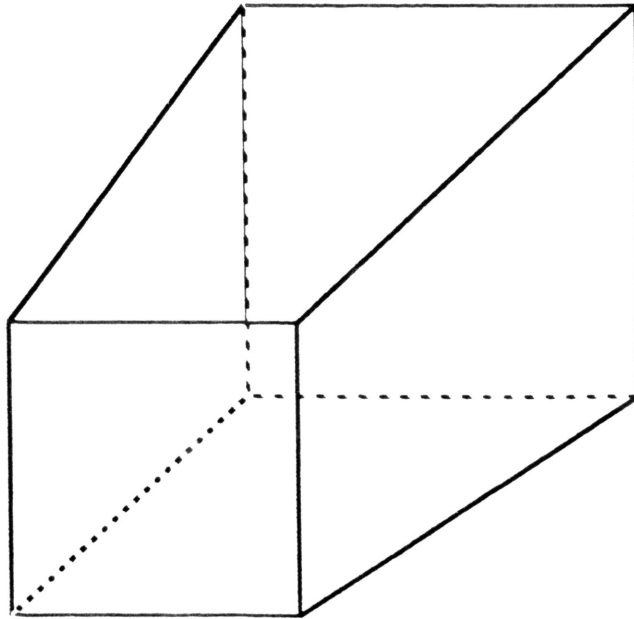
Fig. 1 Transformation Matrices

tive Graphics System) [2], different branches of the hierarchy may have Local Transformation Matrices. To find the correct transformation matrix for each individual polygon, the respective matrices must be multiplied together. Since matrix multiplication is not associative, the concatenation order specified by the application must be followed.

Clipping occurs when the polygon falls outside of a defined view volume.



Parallel Projection



Perspective Projection

Fig. 2 View Volumes

Two types of projections are used in polygon viewing (Figure 2). The parallel projection defines a unit cube starting at the origin 0,0,0 and extending in the positive unit axis directions. Perspective projection defines a view volume with a back plane at $z = 1$, front plane at $z = z_{min}$, and sides at $y = \pm z, x = \pm z$ [3]. The particular viewing environment used is defined by the user application. If a polygon falls completely outside of the viewing volume, it is rejected from being rendered and displayed. If it falls completely within the viewing volume, it is accepted without modification and passed on to the rendering and display stages. If the polygon straddles any of the viewing boundaries, it will have to be clipped according to the intersection equations and the new points passed on to the remaining stages.

2.2 Computer Architectures

The design of computer graphics systems has closely followed the path of the development of supercomputing systems. In both worlds, the following three types of architectures exist. The earliest of such architectures is the scalar architecture (SISD, or Single Instruction Single Data). Scalar processors are the simplest design in which a single processor performs one operation at a time in a serial manner. In this type of system, the amount of time to complete a task is the number of operations multiplied by the amount of time per each operation.

An extension of the single, scalar processor is multiple or parallel processors (MIMD, or Multiple Instruction Multiple Data) [3]. If the problem can be divided up into many independent tasks, then a speedup roughly proportional to the number of processors used can be obtained. In real life, dependencies between processes do occur (i.e. Task 2 depends on the answer currently being computed in Task 1) and the performance increase drops accordingly. Such data dependencies require synchronization between pro-

processors to maintain the integrity of data. If the tasks are truly parallel, the time to completion is simply the number of operations multiplied by the time per operation divided by the number of processors. When data dependencies do exist, performance tends to be less than this linear relationship due to wait states invoked by synchronization overhead. This non-linear performance degradation is known as Amdahl's Law, which states that the speedup, $S(n)$, over a scalar processor to be

$$S(n) = \frac{n}{1 + (n - 1)f}$$

where f is the fraction of non-parallelizable tasks and n is the number of processors [5]. Graphics Transformations have been shown to be inherently parallel due to the fact that each polygon is independent, in that the order of processing polygon transformations is not important [11].

A third type of high performance architecture is the vector processor (SIMD or Single Instruction, Multiple Data) [4]. Vector processors implement a more fine-grained form of parallelism in which each processor contains multiple functional units, and where each functional unit is an adder, multiplier or other type of simple operation. These functional units can be chained together in a pipe so that the answer from the first functional unit can be used by the second in its computation. In example, the vector triadic equation

$$X_i = A_i \times B_i + C_i$$

could be solved by performing the operation $D_i = A_i \times B_i$ in the multiply functional unit and the accumulate operation $X_{i-1} = D_{i-1} + C_{i-1}$ in the adder functional unit during the same clock cycle. In this way, one operation per cycle can be realized, providing the vector pipe can be filled. However, the time to solution for one element is still two cycles with added overhead for storage

in the pipe. Again, the problem of dividing our task into smaller, independent tasks is presented. As in MIMD architectures, performance degradation occurs as the fraction of serial code increases.

2.3 Systolic Arrays

An efficient pipelined approach to the matrix-vector multiplication problem is the Systolic Array. The Systolic Array is a very specialized vector pipeline in which each of the processors is a very specialized cell, such as a multiply-add cell. According to Kung,

A systolic system is a network of processors which rhythmically compute and pass data through the system. Physiologists use the word "systole" to refer to the rhythmically recurrent contraction of the heart and arteries which pulse blood through the body. In a systolic computing system, the function of a processor is analogous to that of the heart. Every processor regularly pumps data in and out, each time performing some short computation, so that a regular flow of data is kept up in the network [6].

Figure 3 shows a linearly connected array appropriate for the matrix-vector problem [7]. Each processor is multiply-add cell. With intelligent scheduling, this approach can produce one result per cycle.

The systolic array has several advantages over conventional approaches to matrix operations. The systolic array is very simple in that it has no μ -control unit. This reduces operational overhead and increases performance. This simplicity of the processing elements (PE's) also reduces area occupied by the processor and makes it possible for an array to fit on a single chip. The inherent regularity of the systolic array makes for simple design and implementation. A processing element needs only to be designed once and then replicated and connected to create the array. With these benefits in

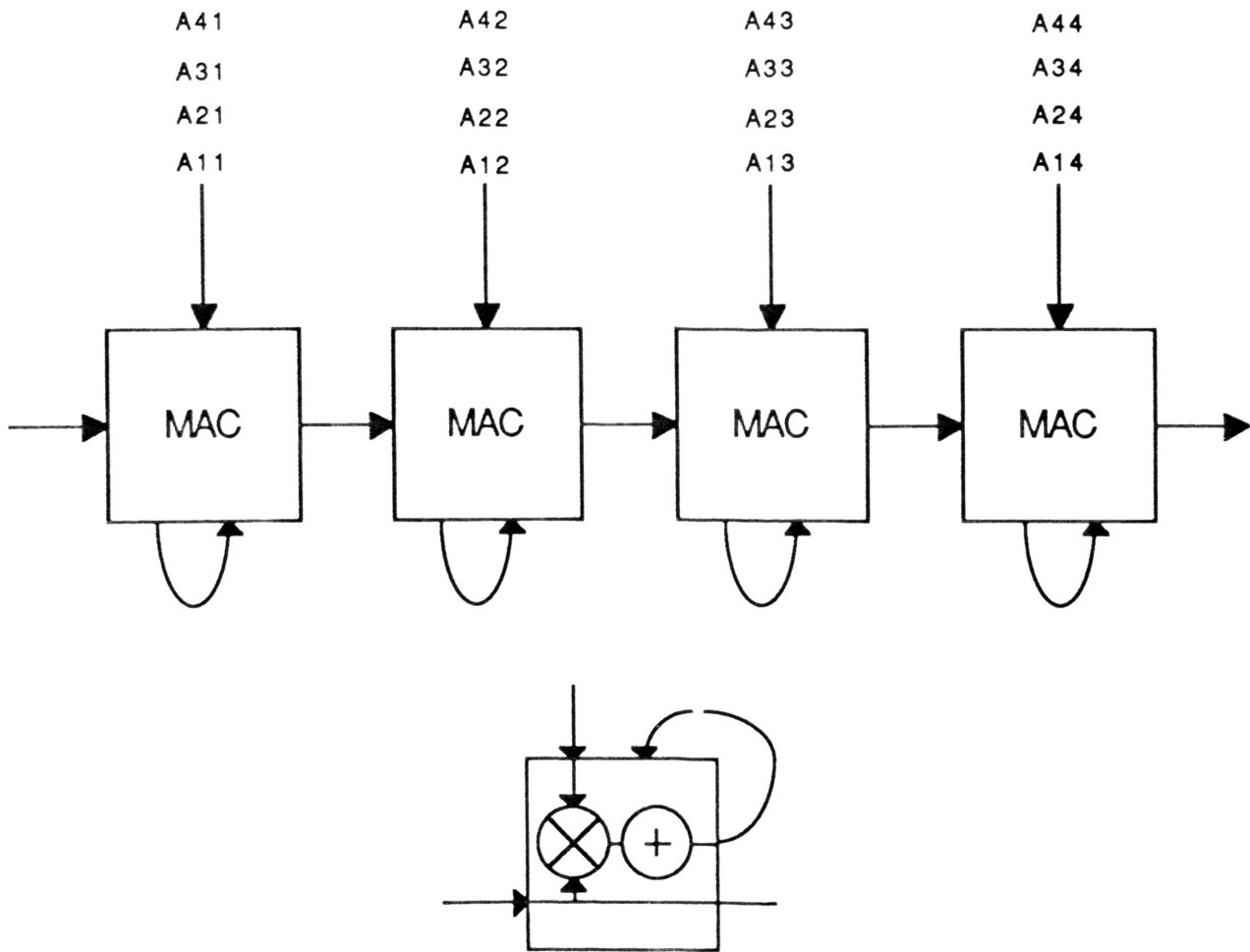


Fig. 3. Linearly Connected Array

mind, it seems a logical choice to adapt a systolic array to the specific problem in computer graphic's of geometrical transformations.

DESIGN APPROACH

3.1 Criteria

In order to begin design of the systolic based transformation engine, several design criteria must be established.

- The processor must be able to sustain a high transformation rate. Targeted performance is on the order of 30 million transformations per second (10 million triangles per second), or roughly an order of magnitude increase over current systems.
- The processor should be compact and regular enough to be easily implemented in silicon using current technology. 3 micron CMOS will be used for comparison purposes.
- The processor must be easily scalable to a parallel system.

Typical computer graphics processors handle their transformations in 32 bit floating point arithmetic. The need for this much precision stems from problems such as scaling objects out of existence due to lack of precision. According to the IEE-754 floating point standard, numbers between 1.18×10^{-38} to 3.40×10^{38} can be represented [8]. This provides the necessary precision and the IEEE-754 standard is a logical choice for the base numeric representation due to its acceptance in the general computer marketplace.

Due to size and complexity restraints, previous systolic systems have relied on fixed point numbers. Floating point calculations have been handled by off-chip processors [9]. This methodology does not meet the requirements stated, in that we will not be able to fit the entire processor on a single chip. However, advances in solid state technology have made it feasible to implement floating point arithmetic on a small enough scale so that several functional units could reasonably fit on a chip. A design proposed by

Jeyadevan implements the IEEE-754 floating point standard in a multiply-add configuration. This cell was chosen to be the basic multiply-add cell (MAC) for this design because of the wide acceptance of the IEEE-754 representation and the need for floating point accuracy.

3.2 Systolic Array Schedule and Optimizations

The incorporation of this MAC into a systolic pipeline requires a schedule for I/O operations. A schedule for linear arrays shown by Kung [7] allows one result to be generated each cycle (Figure 4). This particular schedule will reverse the order of the input vector to z, y, x for reasons which will be discussed concerning clipping. In optimizing this schedule for the general transformation matrix, it can be seen that the last column of the general matrix is always composed of three zeros and a one (Figure 1). The purpose of this column is to preserve the one in the last element of the vector. Since this one is a constant, it can be held internally and not actually loaded via the input stream. This will allow us to do away with the last column of the transformation matrix. Therefore, we have eliminated four multiplications and three additions. In the actual design, the one in the vector is hardwired into the last processing element. In further optimizations, complexity of polygons is restricted to the simplest case, the triangle. Complex surfaces can be tessellated from triangles so limiting the processor to this primitive will not prove too restrictive [10].

The data provided to this linear array will come from a specialized memory commonly referred to as a Display List (DL). The DL keeps track of polygons and their affiliated transformation matrices. The transformation matrices are expected to be stored after concatenation so that they may be used as the correct transformation matrix for the triangle. These matrices may be pre-computed by a host processor. This approach will remain efficient

T_5				a_{41}
<hr/>				
T_4			a_{31}	a_{42}
<hr/>				
T_3		a_{21}	a_{32}	a_{43}
<hr/>				
T_2	a_{11}	a_{22}	a_{33}	
<hr/>				
T_1	a_{12}	a_{23}		
<hr/>				
T_0	a_{13}			
	PE1	PE2	PE3	PE4
	(x)	(y)	(z)	(1)

Fig.4. Optimized Schedule for Transformation Processor Systolic Array

as long as the number of triangles remains much greater than the number of independent transformation matrices. In real-world applications this tends to be true if the triangles do not all move separately. If a faster matrix composition is desired, matrices could be pre-multiplied with orthogonal systolic array on the Display List Controller.

As previously mentioned, the vector normals need to be transformed with the same matrix as each of the associated points. However, translation of the normal vectors is not desirable. Upon examination of the transformation matrix, it can be seen that all translations occur in the last row of the transformation matrix. The last row can therefore be deleted along with the

associated one in the last element of the vector. The new representation is element per element similar to the original matrix.

$$[xyz] = [xyz] \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Fig. 5. Vector Normal Transform Matrix

This transformation can either be handled by a separate, parallel three cell systolic array, or use the same array as the vertices. Since chip size is a concern, normals are transformed in the same array as the associated points. This implementation also reduces I/O bandwidth since now we are only loading three values for the input vector per cycle instead of 6 values.

Since a four element array is still being used, the fourth PE will be passed a zero as the matrix multiplicand in order to keep from translating the vector normal. The net result is that the vector normal follow the vertices in a steady, uninterrupted stream. An alternate approach that was considered released vector normal values after the third PE. This would make the first value of the vector normal become available on the output stream at the same time as the final value of the vertice, thus doubling the necessary output bandwidth. The net savings in calculation time would be 17% in such a design. The chosen design orders data linearly in the form

$z_{vert}, y_{vert}, x_{vert}, z_{norm}, y_{norm}, z_{norm}$

3.3 Design

The overall design of the array is shown in Figure 6. Three of these pipelines are run in parallel to provide the three vertices necessary for a triangle. Since each of these vertices uses the same transformation matrix, only one bus is needed for the matrix input stream. A simple command bus uses only three bits to determine whether each data input is an x, y or z coordinate, whether the data is a point or normal vector, or whether the

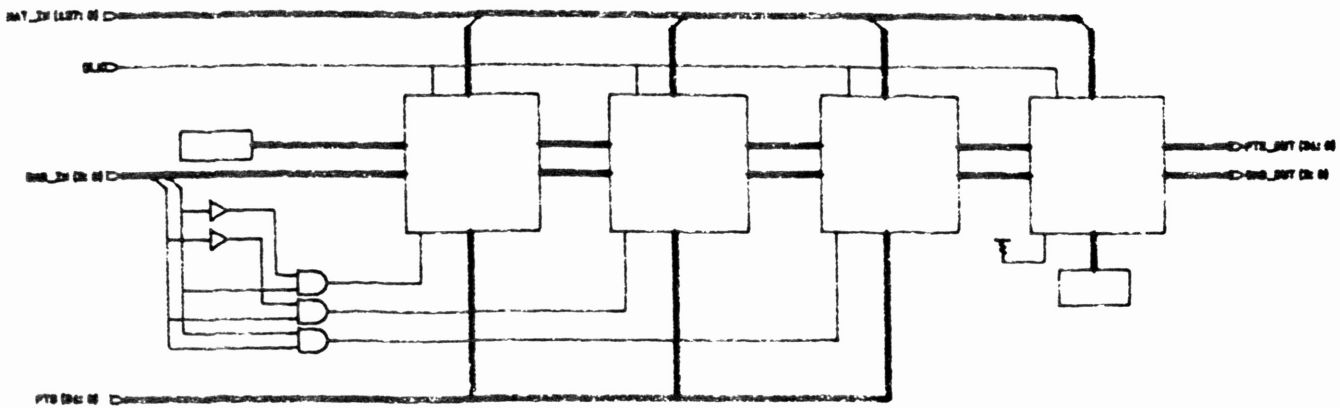


Fig. 6 Transformation₁₃ Processor Architecture

data is invalid.

bit 2	bit 1	bit 0	Command
0	0	0	Data Invalid
1	0	0	Data Invalid
0	0	1	X point
1	0	1	X normal vector
0	1	0	Y point
1	1	0	Y normal vector
0	1	1	Z point
1	1	1	Z normal vector

Table 1 Commands

The command stream is propagated via latches through each processing element so that the later stages of the processor are able to recognize the data. A hardwired zero is fed into the accumulator of the first processing element to clear the accumulator and a hardwired one is fed into the data multiplier of the last processing element in accordance with the transformation matrix.

3.4 Internal MAC Pipelining

Pipelining in this processor can be expanded by adding pipeline stages within the MAC. This means that the MAC itself is subdivided into several stages each having a fairly constant and consistent propagation delay. Care must be taken in adding more stages to the pipeline in order to prevent excessive growth in area of the cell due to latches involved in the pipeline. Also, the design must find the optimum size of each pipeline unit to maximize total latency time as well as clock cycle time. As shown in Figure 7, total latency time from start to finish increases due to latch load delays with each additional pipeline segment. Additional latency penalties can be incurred

Pipeline Statistics

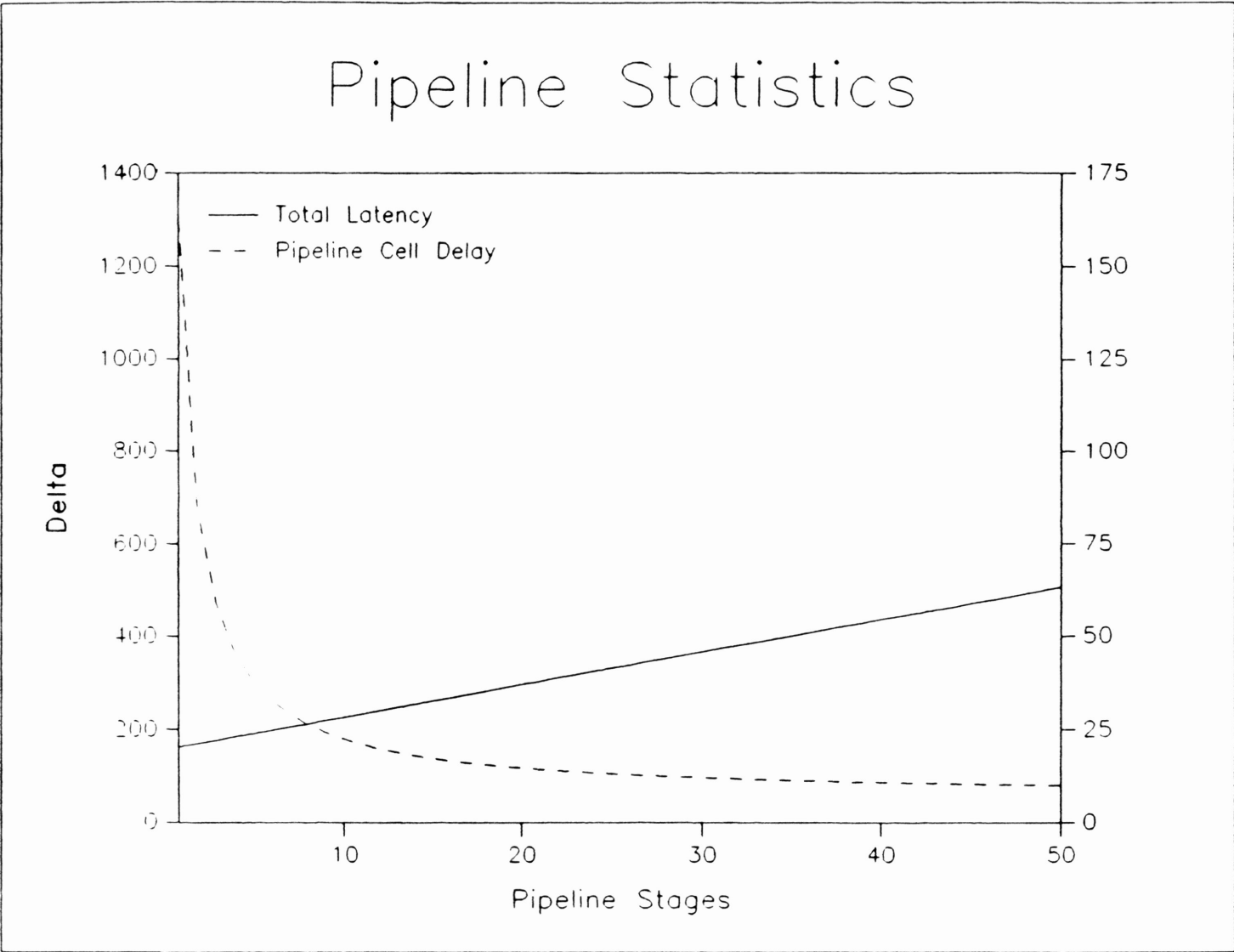


Fig.7 Pipeline Effects on Latency and Segment Delay

by having inconsistent pipeline segment delays. For instance, a unit with a delay time of 20 is broken down into three stages with delay times 10, 6 and 4. The maximum delay is 10 dictating that the clock cycle must be at least 10. Therefore, stage 2 has 4 units of unused time and stage three has 6 units of unused time. This contributes to an additional 10 units of latency time incurred by mismatched pipeline delays. Also, delay time in each discrete pipeline stage tends to become dominated by the latch, whose delay time is constant, as the amount of pipeline segment delay time decreases. This curve shows no appreciable benefit after approximately 15 pipeline stages for the MAC.

Selection of the optimum pipelining scheme began with finding the smallest discrete functional unit within the MAC. Delay times are chosen in accordance to Hwang's model. The 8 bit subtractor has largest delay time of any block at $24 \Delta_T$. The subdivisions were chosen to keep the individual delays as close to but less than $24 \Delta_T$ as possible. Other considerations included minimizing the amount of I/O paths to keep the latch count low. The subdivisions are marked on the MAC overview in Figure 8. The Braun Array multiplier unit is capable of being pipelined between every row of adders as seen in Figure 9 (hatched boxes represent latches). However, since the chosen pipeline segment delay is $24 \Delta_T$ and the Full Adder delay is $6 \Delta_T$, 4 rows of Full Adders can fit in one pipeline segment. This gives five stages of 4 Full adders and one stage of 2 Full Adders. The final stage is comprised of Carry Look-ahead adders to prevent carry-around delays and is contained in the segment with the 2 Full Adders. Since the Braun Array is the only element in Level one with a delay greater than $24\Delta_T$, the output busses of the XOR and 8 bit CLA Adder must contain five additional levels of latches in stage one to keep the data in the same level of the pipe as the Braun Array The

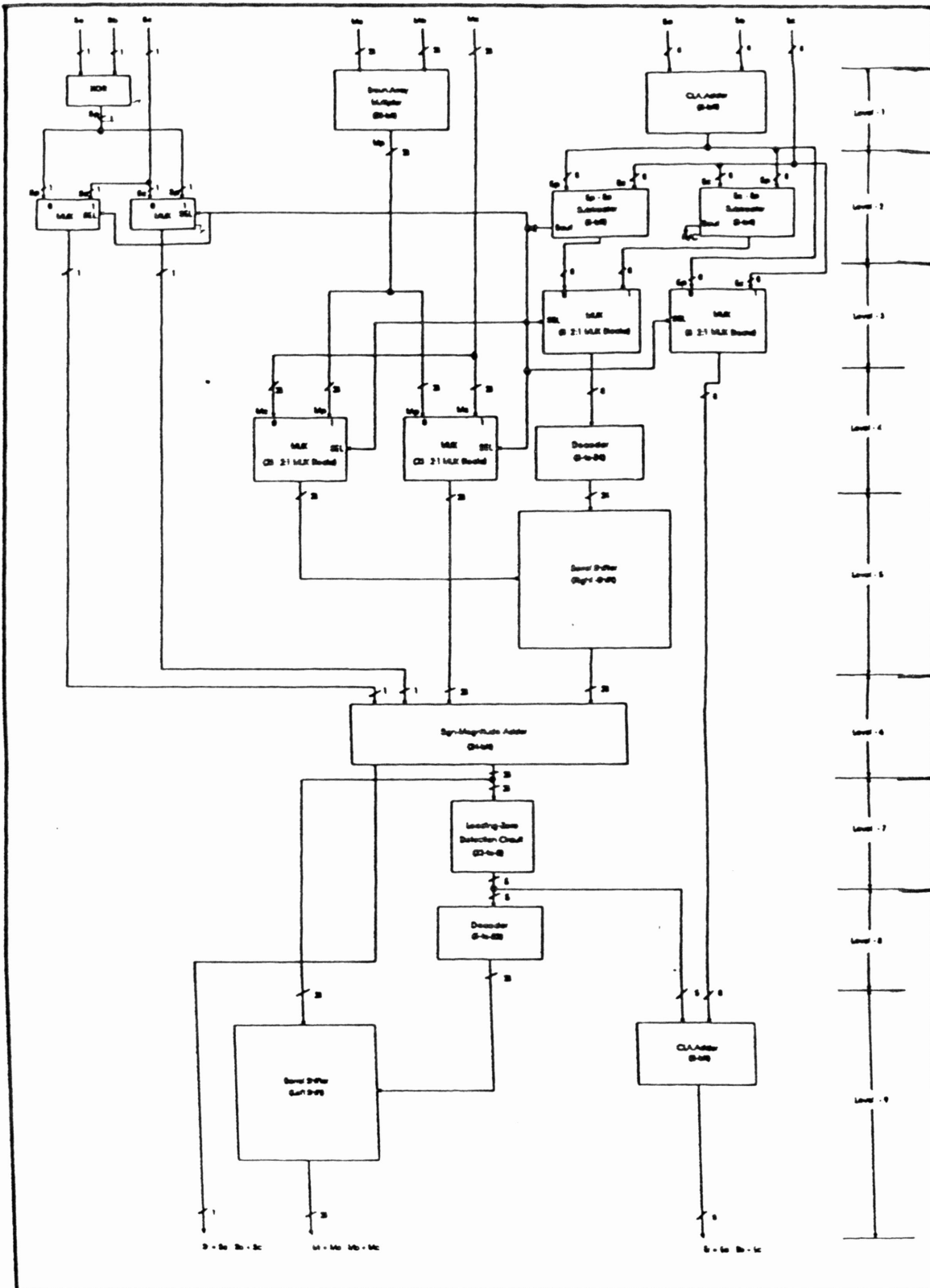


Fig.8 MAC Flow Chart

remaining subdivisions occur between levels 1 and 2, levels 2 and 3, levels 6 and 7 and levels 7 and 8.

3.5 Clipping

The decision to clip against a view volume is data-dependent in that it requires x , y and z to decide whether clipping must be done. Since the systolic schedule being used produces only one of the values per cycle, these values must be accumulated. Due to the nature of the clipping criteria (Figure 10), intelligent ordering of data can eliminate wait-states. It can be seen that if two compares are to be made per clock cycle (greater than and less than for each value of the vertice), the z value of the vertice is the only value that is not data dependent in either parallel or perspective projection. Thus, if data enters the clipping decision unit in the order z , y , x and the compare unit delay time is less than $24 \Delta T$, one compare can be made per cycle. In the perspective case, z is stored and compared with z_{min} , then y with z in the next cycle, and finally x with z .

To determine a trivial acceptance, we can do a NOR of all the conditions. If all three points are trivially accepted, then the entire triangle may be passed without clipping. A bit per bit logical AND of a line will determine a trivial rejection of that line. If one line is rejected, the remaining two need to be clipped. If two lines are rejected, the remaining line needs to be clipped. A rejection of all line allows the triangle to be completely disregarded. The design of such a clipping criteria unit is shown in Figure 11.

If clipping is necessary, then the following parameterized equations can be used to solve the intersection of the line and the plane boundary.

$$x_v = (x_2 - x_1)t + x_1$$

$$y_v = (y_2 - y_1)t + y_1$$

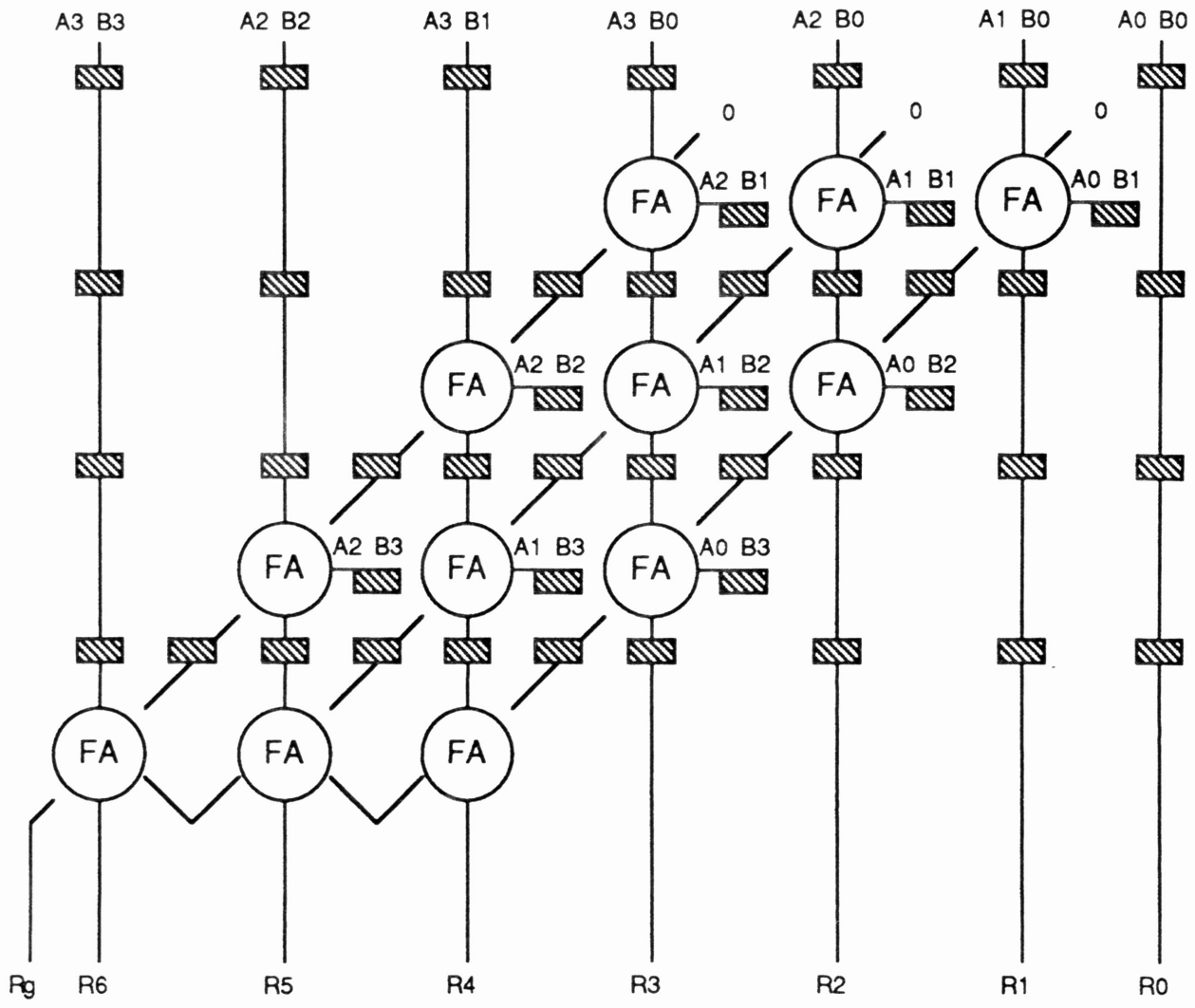


Fig.9. Pipelining in a Braun Array

1. Above View Volume	$y_v > 1$
2. Below View Volume	$y_v < 0$
3. Right of View Volume	$x_v > 1$
4. Left of View Volume	$x_v < 0$
5. Behind View Volume	$z_v > 1$
6. In Front of View Volume	$z_v < 0$

Parallel Projection

1. Above View Volume	$y_v > z_v$
2. Below View Volume	$y_v < -z_v$
3. Right of View Volume	$x_v > z_v$
4. Left of View Volume	$x_v < -z_v$
5. Behind View Volume	$z_v > 1$
6. In Front of View Volume	$z_v < z_{min}$

Perspective Projection

Fig. 10. Clipping Criteria

$$z_v = (z_2 - z_1)t + z_1$$

In example, the clipping equations for the $y = 1$ in a parallel projection would be derived by substituting 1 for y_v and solving for t .

$$t = \frac{(1 - y_1)}{(x_2 - x_1)}$$

This value of t is then substituted in the above equations for x_v and z_v .

$$x_v = \frac{(1 - y_1)(x_2 - x_1)}{(y_2 - y_1)} + x_1$$

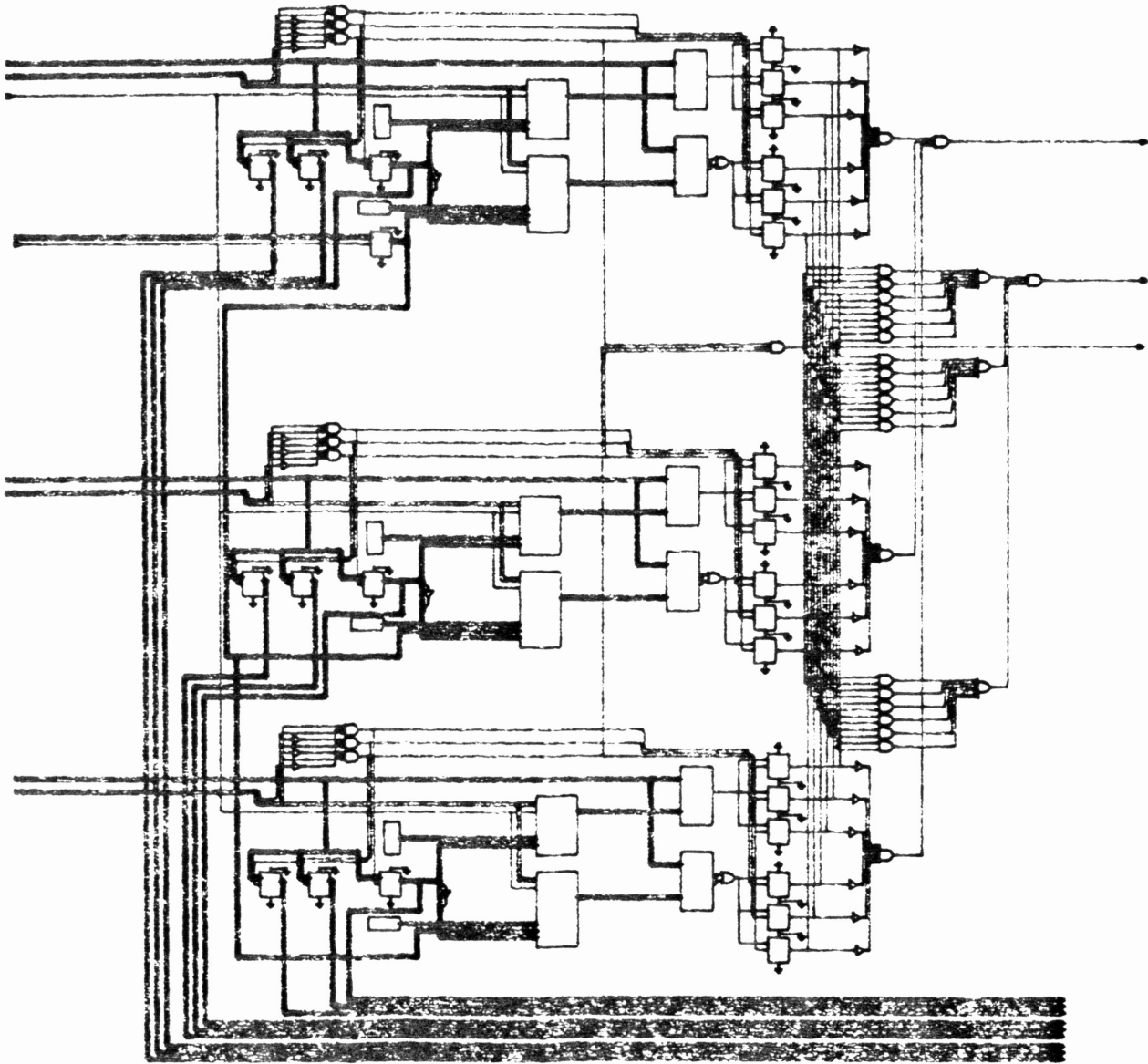


Fig. 11. Clipping Criteria Unit

$$z_v = \frac{(1 - y_1)(z_2 - z_1)}{(y_2 - y_1)} + z_1$$

and for $y = z$ in a perspective projection

$$t = \frac{(z_1 - y_1)}{(y_2 - y_1) - (z_2 - z_1)}$$

$$x_v = \frac{(x_2 - x_1)(z_1 - y_1)}{(y_2 - y_1) - (z_2 - z_1)} + x_1$$

$$y_v = \frac{(y_2 - y_1)(z_1 - y_1)}{(y_2 - y_1) - (z_2 - z_1)} + y_1$$

These equations require four floating point addition/subtractions, one floating point multiply and one floating point divide for the parallel case and six floating point addition/subtractions, one floating point multiply and one floating point divide for the perspective case [3].

Two possibilities exist for the solution of these clipping equations. A general purpose floating point unit or a clipping pipeline could be implemented. The clipping pipeline would contain an adder/subtractor, a divider and a multiply adder chained together. The question of whether a pipeline is justified depends on whether or not enough data can be passed to the pipeline to keep it full. If the number of actual clipped polygons remains high and continuous, a clipping pipe would be efficient. In actuality, the number of polygons clipped is usually small compared to the number of total polygons and all of these clipping operations will not tend to happen frequently enough to fill a pipeline.

A general-purpose floating point unit could be used to compute the intersections. This unit would run slower than the transformation engine and would inhibit performance if clip intersections needed to be calculated. Since data would be leaving the clip calculations would now be delayed, triangles leaving the transform and clip system would not be in the same order that

they entered. This is the same problem faced by parallel graphics systems such as the Alliant GX4000 [11]. In order to keep data in order, the clipping unit could halt the transformation pipe when a triangle needs to be clipped. This would incur the clipping penalty to all triangles after the current one. A solution that is used in parallel systems is to associate a data dependency tag with each triangle. If rendering data (lighting models, etc...) for that triangle is different than the triangles that follow, the dependency tag could be used to trap this condition and then halt the transform pipe. Otherwise the triangle is re-inserted into the stream at a later time. This solution only incurs the penalty to the remaining triangles if it is necessary. This solution also has the added benefit of allowing easy extension of the single transform processor to multiple transform processors.

3.6 Rendering

Before rendering can be done, the clipped triangle must then be projected onto a 2D projection plane. This operation simply requires the stripping of the z component in parallel projections. In perspective projections, the following matrix is used [1].

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In order to accommodate the performance of a high speed transform processor, rendering be able to be done at near the same rate as transformation. The Triangle Processor/Normal Vector Shader is a pipelined rendering system capable of sustained 1 million shaded triangles per second. Use of several of these pipe in parallel leads to throughput rates of up to 5 million triangles per second.

IV. RESULTS

Two versions of the transformation processor were considered. The simpler version had no internal pipelining in the MAC. An improved version was also considered with the proposed internal pipelining scheme of the MAC.

The single pipelined version of the MAC has a total area of 7372 ΔA and a total delay time of 156 ΔT . The transformation pipeline utilizes twelve of these units. In addition, each unit contains 1, 3 bit latch for the command stream and 3, 32 bit latches for holding incoming data for the MAC. Each of these latches is a simple D type latch.

Instance	Segment Delay Δ_T	Total Latency Δ_T	Area Δ_A
MAC1 (no pipelining)	156	156	7372
MAC2 (11 stage pipeline)	24	264	12,352
D Latch	-	7	6

Table 2. Delay and Area Parameters

The approximate total area for the non-pipelined 3-vertice transformation processor

$$\Delta_A^{Total} = 12 \times \Delta_A^{MAC1} + 3 \times 32 \times 12 \times \Delta_A^{Latch} + 3 \times 12 \times \Delta_A^{Latch} = 95,592$$

To compute actual area, Δ_A is taken to be 25 x 25 microns. This gives an effective area of .6 cm^2 , which while easily fit on a 1 cm^2 chip. The number of triangles per second non-pipelined MAC if Δ_T is taken to be .75 ns:

$$\frac{1}{6} \frac{1}{(\Delta_T^{MAC1} + \Delta_T^{Latch}) \times .75 \times 10^{-9}}$$

$$\frac{1}{6} \frac{1}{(156 + 7)(.75 \times 10^{-9})} = 1.36 \text{million} - \text{triangles} - \text{per} - \text{second}$$

The memory bandwidth required to supply this system is

$$(1.36 \times 10^6 \frac{\text{triangles}}{\text{sec}}) (6 \frac{\text{vert} - \text{norms}}{\text{triangle}}) (8 \text{input} - \text{streams}) (4 \frac{\text{bytes}}{\text{stream}}) = 260 \frac{\text{Megabytes}}{\text{sec}}$$

The pipelined version of the MAC gives a total area of

$$\Delta_A^{Total} = 12 \times \Delta_A^{MAC2} + 3 \times 32 \times 12 \times \Delta_A^{Latch} + 3 \times 12 \times \Delta_A^{Latch} = 155,352$$

Actual area, sing Δ_A of 25 x 25 microns squared is .97 cm^2 . The pipelined MAC's clock cycle time calculation is based on pipeline segment delays

$$\frac{1}{6(\Delta_T^{MAC1} + \Delta_T^{Latch}) \times .75 \times 10^{-9}}$$

$$\frac{1}{6(24 + 7) \times .75 \times 10^{-9}} = 7.1 \text{million} - \text{triangles} - \text{per} - \text{second}$$

Total latency time has, however, increased from 156 to 264. This increase in latency is due in part to the delays caused by the latches themselves. The latches cause a 77 Δ_T increase in latency. The remains 79 Δ_T is due to "wasted time" in various pipeline segments due to not utilizing the entire 24 Δ_T available. The memory bandwidth required to supply this system is

$$(7.1 \times 10^6 \frac{\text{triangles}}{\text{sec}}) (6 \frac{\text{vert} - \text{norms}}{\text{triangle}} (\text{input} - \text{streams})) (4 \frac{\text{bytes}}{\text{stream}}) = 1.2 \frac{\text{Gigabytes}}{\text{sec}}$$

V. DISCUSSION AND CONCLUSIONS

The non-pipelined version of the geometry engine produces only 1 million triangles per second. Currently workstation technology is capable of providing the same performance. However, the systolic processor does fit on a chip whereas the geometry stages of other processors tend towards several floating point chips. With the extra room on the chip, a smaller normal vector transformation pipe could be installed composed of three parallel systolic arrays consisting of three PE's each. This would effectively double the speed of the processor.

The internally pipelined approach shows more promise for real performance benefits. Although slightly lower than design expectations, the 7 million triangles per second is healthy performance for a single chip processor. Because of its size, parallel methods such as those used in the GX4000 could be implemented with the systolic processor. More aggressive chip technology, such as 1 micron CMOS, could turn in even larger performance gains.

Unfortunately, due to the speed of the processor, the memory bandwidth requirements are extraordinarily high. Liberal use of interleaved DL memory systems could overcome this bottleneck. For instance, the memory bandwidth per input stream is only 170 megabytes per second. Partitioning DL memory off into separate regions for each vertice and each matrix row and use of separate busses for each of these regions would make the memory bandwidth problem more manageable.

The Triangle Processor/Normal Vector Shader would be an appropriate rendering architecture for this type of transformation engine. The triangle Processor could support the speed of the transforms and maintain high throughput. The problem of high speed clipping is still a bottleneck though. If large amounts of polygons required clipping, the throughput of the graphics

system could be reduced dramatically. More investigation needs to be done to alleviate this problem.

The systolic array geometry engine, with enough pipelining, provides performance that is close to the criteria established. Due to its compactness, it could show promise in a high speed graphics environment.

REFERENCES

- [1] J.D. Foley, A. Van Dam, Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Co., Reading, Ma. pp. 255-277, 1983.
- [2] S. S. Abi-Ezzi, A. J. Bunshaft, "An Implementers View of PHIGS", IEEE Computer Graphics and Applications, pp. 13-22, Feb. 1986.
- [3] J. D. Foley, A. Van Dam, Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Co., Reading, Ma. pp. 295-297, 1983.
- [4] J. P. Hayes, Computer Architecture and Organization, McGraw Hill Book Co., New York, New York, pp. 570-622, 1988.
- [5] J. P. Hayes, Computer Architecture and Organization, McGraw Hill Book Co., New York, New York, pp. 586, 1988.
- [6] H. T. Kung, C.E. Leiserson, "Systolic Arrays ," Sparse Matrix Symposium, pp. 256-282, 1978.
- [7] S. Y. Kung, VLSI Array Processors, Prentice Hall, Englewood Cliffs, New Jersey, pp. 138-139, 1988.
- [8] J. P. Hayes, Computer Architecture and Organization, McGraw Hill Book Co., New York, New York, pp. 196-198, 1988.
- [9] Y. Y. J. Leung, M. A. Shanblatt, "Systolic array simulation for quantification of speed/area parameters," Simulation, pp. 295-300, June 1985.
- [10] M. Deering, S. Winner, B. Schediwy, C. Duffy, N. Hunt, "The Triangle Processor and Normal Vector Shader," ACM Computer Graphics, pp. 21-30, 1988.
- [11] J. G. Torborg, "Parallel Processing Techniques Overcome Graphics Bottlenecks," Computer Technology review, Spring 1988.