A VLSI Based Real-Time Hidden Surface Elimination Display System

Stefan G. Demetrescu

Computer Science Department
California Institute of Technology

4090-TR-80

A VLSI Based Real-Time Hidden Surface Elimination Display System

Thesis by

Stefan G. Demetrescu

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science in Computer Science

California Institute of Technology

Pasadena, California

4090-TR-80

1980

(Submitted May 19, 1980)

ABSTRACT

This thesis describes a novel approach to the problem of generating dynamic TV raster displays for real-time simulation (such as for visual flight simulation). In particular, the most time consuming part of generating such displays, the hidden surface elimination, is performed using many identical custom VLSI processors. Each processor is assigned a surface and, for each pixel, all processors compete to decide which object is visible.

It is found that this approach leads to a practical system which is conceptually and practically simple, expandible, and reliable.

1.0 INTRODUCTION

Numerous new applications of computer graphics have resulted in increasing demand for real-time graphics; that is, the generation of computer synthesized pictures of many (e.g., 2000) 3-dimensional objects which appear to move smoothly in time (i.e., a different frame every 1/10 to 1/60 of a second) and where all the computation is done at the time of the display. Such graphics have been successfully implemented for moderate cost in line drawing systems utilizing random scan black and white CRT displays (e.g., Evans and Sutherland Picture System). While giving the user a sensation of smooth movement, they lack the reality of true objects because line drawings lack solidity, color, shading, and hidden line elimination.

These disadvantages have been overcome by the use of moderately priced frame buffer systems which typically consist of a 512x512 resolution raster color TV monitor connected to a large memory (typically 512x512x8 = 2M bits) and a medium sized general purpose processor. While giving a much greater sense of reality through the use of color, hidden surface elimination and shading, these systems lack real-time capability. That is, they can only produce new pictures every few seconds.

Raster systems have been built which are both real-time and give a sense of reality (through use of special purpose hardware attached to a general purpose processor) but these systems are typically very expensive (>$1M) and thus have very limited applications (e.g., in visual flight simulation for pilot training).

The purpose of this study is to design a system which can provide all the advantages of a real-time raster system at a significantly lower cost while keeping the system simple, expandible, and reliable (highly testable). This is realized through the use of highly parallel processing using identical low cost custom LSI elements to implement a major portion of the system.

2.0 OVERVIEW OF THE APPROACH

2.1 The Problem To Be Solved

The problem is to represent a dynamic view of a 3-D set of objects on a raster CRT screen where both the view and the objects are chosen by the user of the system arbitrarily (within system imposed limits). One can visualize this as if the 3-D objects actually existed and were being imaged from some moving location (the viewing location) by a TV camera.

The problem then can be reduced to deciding what is to be seen at each pixel (picture element) of the screen (Figure 1a) at each instant of time. It is first necessary to determine the location occupied by the projection of the 3-D objects on the screen (Figure 1b). Next, it must be determined which of the many objects which are potentially visible at each pixel are actually visible (i.e., closest to the observer) (Figure 1c).

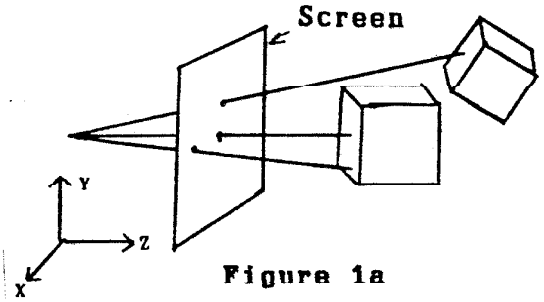


Figure 1a

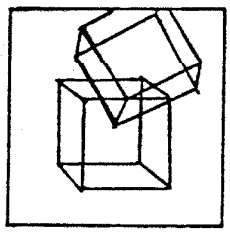


Figure 1b

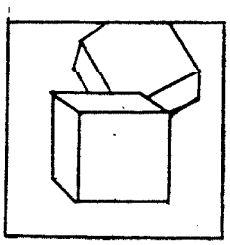


Figure 1c

2.2 A General System Architecture

Clearly a simple system architecture suggests itself (Figure 2). First, a data base contains the complete description of all the 3-D objects which make up the simulated space. For each view, the data base manager choses those objects which are going to potentially participate in the formation of the view and passes them to the geometry processor. This processor calculates which pixels on the screen each object can potentially occupy (that is, if it is not obscured by others). This process is easily accomplished through the use of well known geometric transformations (e.g., rotation, translation, and perspective projection).

Next, the hidden surface eliminator determines which one of the many objects which are potentially visible at each pixel is actually visible by chosing the one which is closest to the observer. This set of pixels is then passed (typically) to a post processor which enhances the picture

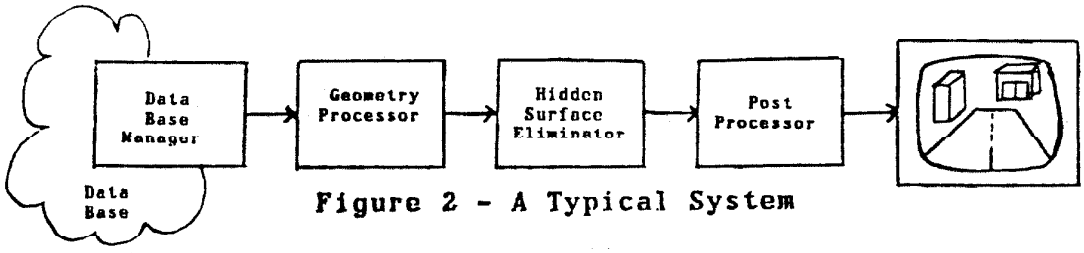


Figure 2 - A Typical System

by eliminating undesirable artifacts such as "staircasing" and aliasing. Finally, the picture is sent to a display screen (typically a TV monitor). Most if not all existing systems which perform this function have a system architecture similar to the one outlined here.

2.3 The Hidden Surface Eliminator

The data base manager and the geometry processor computations grow linearly with the number of objects (as a result of the fact that each object can be considered independently of the others). However, the hidden surface eliminator must consider interactions between objects. Thus, the computations grow typically as n^2 and at best as n*log(n) (by sorting the objects). This observation leads author to concentrate attention on this bottleneck.

Conceptually, one can approach the hidden surface problem in dual ways. The first is an object serial, pixel parallel method. That is, the objects are considered in sequence and placed in their appropriate locations in a frame buffer. Whenever many objects fall on the same space, it is necessary to resolve the dispute using depth (Z) coordinate information. Thus, each pixel can be thought of as possessing a processing unit which considers the objects in sequence. Consequently, if more objects are to be processed per frame, the processing speed must be increased. If, on the other hand, more pixels are to be processed per frame, then more processing units must be added.

The second approach is pixel serial, object parallel. Each object is interrogated at each pixel to determine whether it is a potential candidate to appear there. If so, it is decided which object is closest. Then, the pixel takes on the color of that object. Thus, each polygon can be thought of as possessing a processing unit which considers the pixels in sequence. Consequently, if more pixels are to be processed per frame, the processing speed must be increased. If, on the other hand, more objects are to be processed per frame, then more processing units must be added.

Table 1 illustrates the duality discussed above. Most existing systems utilize some combination of the above approaches.

Comparison of Dual Approaches to the Hidden Surface Problem

	First Approach	Second Approach
One Unit Per:	Pixel	Object
Objects:	Sequential	Parallel
Pixels:	Parallel	Sequential
If More Objects:	Faster Units	More Units
If More Pixels:	More Units	Faster Units

Table 1

2.4 Chosing The Obvious

Clearly, the approach chosen by a designer is dependent upon the technology in which the function will be implemented. Since the author wishes to explore the possible advantages of a VLSI approach to this problem, VLSI is the technology chosen. VLSI implementations exhibit their best performance when they are very regular and repetitive (see also section 4.1). This fact, combined with the desirability of generating pixels serially as needed for the TV scan (so that no frame buffer is necessary) points to an obvious implementation of the second approach outlined above. That is, for each pixel all objects potentially visible there compete among themselves. The object closest to the observer "wins" and is displayed.

The most obvious implementation appears to be optimal. Consider assigning each surface of each object to a special purpose processor each frame. This processor consists of a surface processor and a comparator processor (see figure 3a). Each surface processor generates, for each pixel, the color/intensity for that pixel (I) and the distance from the observer (Z) independently of all other processors. The comparator accepts 2 sets of (Z,I) pairs, and outputs the (Z,I) pair with the Z closest to the observer. Thus, for each pixel, the (Z,I) of the front surface is available at the output of the last comparator.

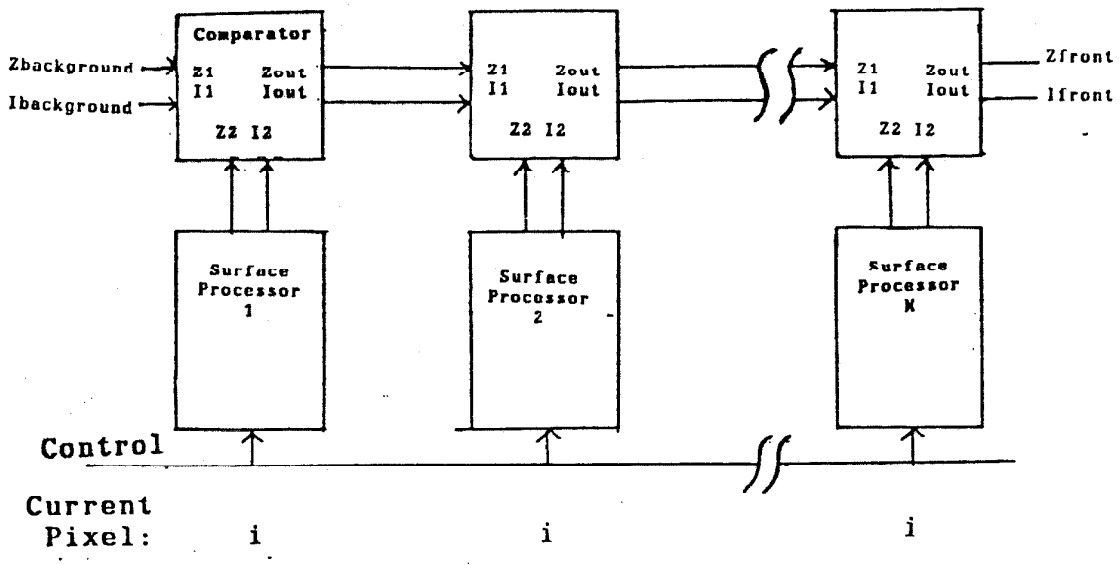


Figure 3a - Competing Surface Processors

2.5 Making The Obvious Work

This approach necessitates M (M = number of surfaces) serial comparisons for each pixel. If one is to compute pixels "on the fly" it is necessary to generate one every 100 ns (for a 512x512 raster updated at 30 Hz). This rather severely limits the number of surfaces. However, it is possible to use the well known pipeline concept in order to skew the operation of the processors (figure 3b).

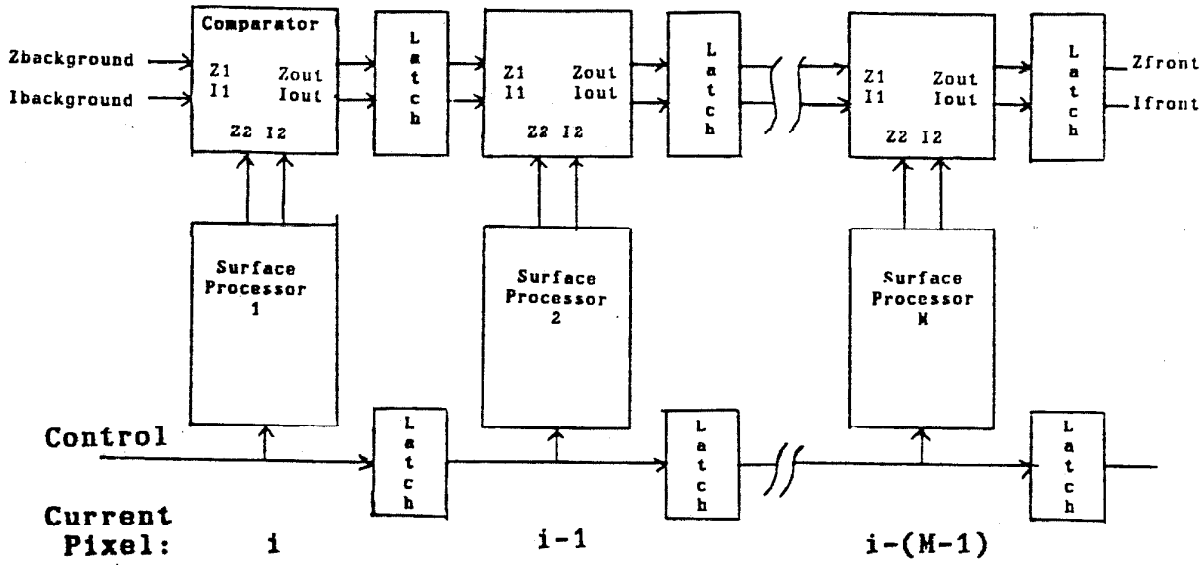


Figure 3b - Pipelined Competing Surface Processors

Thus, only one comparison is performed by each comparator during each pixel cycle. The pixels travel through the pipeline at one surface comparison per cycle. Thus, it takes each pixel M cycles to traverse the pipeline. However, new pixels exit the pipeline each cycle after the first one has exited. Clearly, all surface processors operate on different pixels at each instant of time due to the skewing effect of the pipeline. This necessitates pipelining all control signals (such as "reset for a new picture").

3.0 DETAILED ANALYSIS OF A PROCESSOR

Up to this point the nature of the surfaces which constitute the 3-D objects has not been essential. In order to actually implement a surface processor it is necessary to limit this nature. It has been found that limiting the surfaces to convex planar polygons allows an especially simple implementation for the processor described above. This does not imply that other shapes are not possible. However, the present discussion will limit itself as stated.

For each pixel, each surface processor must determine two things: 1) if the polygon is potentially visible, and 2) if it is visible, the value of Z (the distance from the observer) and I (the intensity and color of the surface).

It is assumed that all 3-D objects are closed (implying that only one side of any surface making up the object is ever visible) and that all polygons making up the surface are defined by an ordered list of their vertices which proceed in a counterclockwise direction when viewed from the outside (see figure 4). For example, the top of the cube shown is defined as (P1, P2, P3, P8). Note how back facing polygons (which can't possibly be visible because they are known to be hidden by the front facing surfaces of the object) project their vertices onto the screen in a clockwise sense whereas front facing polygons (which are seen unless obscured by other front facing polygons) project their vertices onto the screen in a counterclockwise sense.

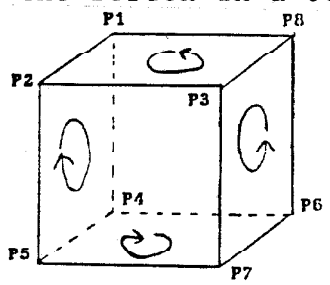


Figure 4 - A Typical Closed Object

Thus, a polygon is said to be potentially visible if it is front facing (that is, if its vertices project onto the screen in a counterclockwise sense) and if it projects onto the visible part of the screen.

It is also assumed that the geometry processor (in figure 2) has moved the 3-D objects into such a position that it is only necessary to perform a perspective transformation in order to obtain the screen coordinates of the polygon vertices and has clipped any objects which fall outside the active screen area when they are projected on it. For the purposes of the following discussion, the following (homogeneous coordinate) perspective transform is

assumed:

$$\begin{bmatrix} Xs \\ Ys \\ Zs \\ 1 \end{bmatrix} = \begin{bmatrix} K*X/Z \\ K*Y/Z \\ K/Z \\ 1 \end{bmatrix} = \begin{bmatrix} K*X \\ K*Y \\ K \\ Z \end{bmatrix} = \begin{bmatrix} K & 0 & 0 & 0 \\ 0 & K & 0 & 0 \\ 0 & 0 & 0 & K \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where Xs, Ys are the pixel coordinates on the screen and Zs is some measure (not a true distance) of the distance from the screen. Since this transformation is linear, it maps straight lines in (X,Y,Z) space to straight lines in (Xs,Ys,Zs) space. A detailed discussion of homogeneous transformations is available in "Principles of Interactive Computer Graphics" by Newman and Sproull, McGraw-Hill, 1973.

3.1 Polygon Interior Determination

Consider the quadrilateral shown in figure 5a. Its projection can be seen in figure 5b. Now, define lines L1 through L4 as shown on the Xs-Ys plane so that they pass through the appropriate projections of the vertices. These lines can each be described as follows:

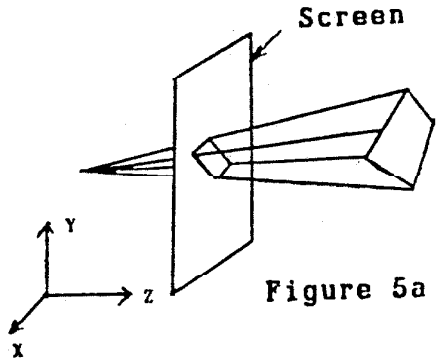


Figure 5a

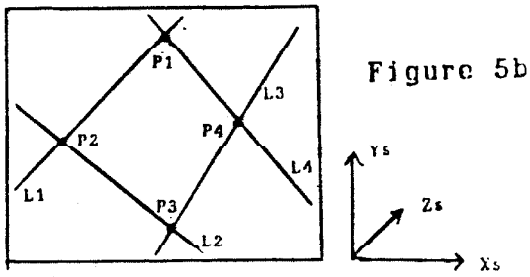


Figure 5b

for Li: $Fi(Xs,Ys) = Ai*Xs + Bi*Ys + K*Ci = 0$

(Xs,Ys) = pixel coordinate

The coefficients A1, B1, C1 (for example) can be calculated from the coordinates of the points through which the corresponding line (L1) passes (in this case P1 and P2). It is assumed that the points P1, P2, P3, P4 proceed in a counterclockwise direction as projected on the screen (since the clockwise direction polygons have been already eliminated because they are never visible). From elementary algebra it is known that L1 is described by:

$$(Y1s-Y2s)*Xs + (X2s-X1s)*Ys + X1s*Y2s-X2s*Y1s = 0$$

but, $Xs = K*X/Z$ $Ys = K*Y/Z$ thus,

$$(Y1*Z2-Y2*Z1)*Xs+(X2*Z1-X1*Z2)*Ys+(X1*Y2-X2*Y1)*K = 0$$

consequently: $F1(Xs,Ys) = A1*Xs + B1*Ys + K*C1 = 0$

where: $A1=Y1*Z2-Y2*Z1 \quad B1=X2*Z1-X1*Z2 \quad C1=X1*Y2-X2*Y1$

It is clear that the value of Fi is positive on the polygon side of the line and negative on the other side. Thus, the condition for a pixel to be inside the polygon bounded by the lines Li is:

$$\text{for all } i: \quad Fi(Xs,Ys) \geq 0$$

3.2 Zs Coordinate Calculation

Due to the linearity of the perspective transform used, the planar polygons in (X,Y,Z) space remain planar in (Xs,Ys,Ys) space. Thus, one can clearly relate Zs to Xs and Ys as follows:

$$Zs(Xs,Ys) = Az*Xs + Bz*Ys + K*Cz$$

where Az, Bz, Cz can be easily determined as shown:

$$Z*Zs = Az*Z*Xs + Bz*Z*Ys + Cz*Z*K$$

$$\text{but } Zs = K/Z \quad Xs = K*X/Z \quad Ys = K*Y/Z$$

$$\text{thus:} \quad K = Az*K*X + Bz*K*Y + Cz*Z*K$$

$$1 = Az*X + Bz*Y + Cz*Z$$

Thus, one can find Az, Bz, Cz by using 3 of the many vertices of the polygon as follows:

$$\begin{bmatrix} X1 & Y1 & Z1 \\ X2 & Y2 & Z2 \\ X3 & Y3 & Z3 \end{bmatrix} \begin{bmatrix} Az \\ Bz \\ Cz \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} Az \\ Bz \\ Cz \end{bmatrix} = \begin{bmatrix} X1 & Y1 & Z1 \\ X2 & Y2 & Z2 \\ X3 & Y3 & Z3 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Zs is some measure of distance from the observer for each pixel (Xs,Ys) and can thus be compared with the Zs of all the other polygons at this pixel in order to determine which is in front. Since Zs = K/Z, the largest Zs is closest to the screen.

3.3 I Calculation

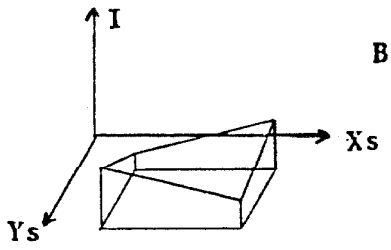
If the 3-D object being modelled is actually made of planar polygons then the intensity across the surface of the polygon is constant. However, objects being modelled often contain curved surfaces which are approximated by planar polygons. In order to avoid the faceted appearance produced by this approximation, a method known as Gouraud shading is used.

In general, the intensity of a perfect diffuser (a perfectly diffusing surface) is proportional to the cosine of the angle between the normal of the surface and the direction of the light source (assumed to be at infinity).

The normal (and hence the cosine of the angle) at the interface between two polygons has a discontinuity which causes the faceted effect. This effect can be avoided by avoiding the step function of intensity at the polygon interface.

This can be achieved by averaging the values of the normals at each vertex between polygons to acquire an average value of intensity at each such vertex. Intensities inside the polygon can then be calculated by (somehow) interpolating these values.

An easy way to visualize this process is to plot the (Xs,Ys) coordinates versus the intensity. The intensity at each vertex is well determined (using the averaging process described).



Boundaries of X,Y,I surface

Figure 6

The most convenient method to assign intensity values along the polygon edges is to interpolate linearly. This method allows independent handling of polygons since they are automatically matched in intensity at the boundaries. Then, the problem is to find a surface in (Xs,Ys,I) space which is continuous and meets the boundary conditions at the polygon edges. In general there are many such continuous surfaces. The one which appears to be optimal is the one which has the minimum surface area (this is also known as the soap bubble problem, i.e., the shape which a soap bubble will form on a wire mesh representing the (Xs,Ys,I) contour).

Unfortunately, it is difficult to solve the minimum area problem for arbitrary (non-planar) polygons in (Xs,Ys,I) space. A good compromise is to divide each polygon into triangles and then to solve the minimum area for each. This, of course, is trivial since the solution for a triangle is always a plane in (Xs,Ys,I) space and hence can always be expressed as:

$$I = AI*Xs + BI*Ys + K*CI$$

where AI, BI, CI can easily be determined as shown:

$$Z*I = AI*Z*Xs + BI*Z*Ys + CI*Z*K$$

$$\text{thus,}\quad Z*I = K*(AI*X + BI*Y + CI*Z)$$

using the 3 triangle vertices one acquires:

$$K\cdot\begin{bmatrix} X1 & Y1 & Z1 \\ X2 & Y2 & Z2 \\ X3 & Y3 & Z3 \end{bmatrix}\begin{bmatrix} Az \\ Bz \\ Cz \end{bmatrix} = \begin{bmatrix} Z1*I1 \\ Z2*I2 \\ Z3*I3 \end{bmatrix} \qquad \begin{bmatrix} Az \\ Bz \\ Cz \end{bmatrix} = \begin{bmatrix} X1 & Y1 & Z1 \\ X2 & Y2 & Z2 \\ X3 & Y3 & Z3 \end{bmatrix}^{-1}\begin{bmatrix} Z1*I1 \\ Z2*I2 \\ Z3*I3 \end{bmatrix}\cdot\frac{1}{K}$$

This scheme generates surfaces which are continuous but not smooth (i.e., which have a discontinuous first derivative). The coefficients AI, BI, CI can be easily calculated from the coordinates and intensities at the vertices.

4.0 IMPLEMENTATION OF A PROCESSOR

4.1 Why A VLSI Implementation?

The design described above is clearly ideal for VLSI. VLSI implementations are optimal when the number of different functional blocks (and chip types) is kept low while the functional blocks which are necessary are used in high volume in order to lower the unit cost. Only two functional blocks are necessary to implement the hidden surface elimination system: the surface processor and the comparator. There are two ways to achieve high volume. The first, taken by most designs, is to use the functional block in many systems a few times per system. The second is to use the function many times in each system thus drastically reducing the number of systems necessary to make the VLSI implementation economical. The second approach clearly matches the requirements of this design: one processor is required for each polygon in the image. Thus, one system may contain 10,000 processors or more.

Another significant feature of VLSI is that the number of gates per chip is expected to grow by an order of magnitude every few years. This typically implies that the issue of communication and interconnection of functions on the chip becomes very important. Designs must be made to scale appropriately without incurring disproportionate increases in the ratio of interconnect to logic area. The design described here easily scales up with no difficulty due to the purely local nature of communication between processors. That is, each processor only communicates with its predecessor and successor. It is also noteworthy that, independent of the number of processors in each package, the number of pins on the package remains constant.

4.2 Specifications Chosen For This Implementation

For this implementation, the author has chosen a 512x512 raster size. Each pixel can take on one of 256 levels of color/intensity (typically used in conjunction with a color map at the end of the pipeline). The Zs coordinate passed to the comparators is 16 bits. These specifications are chosen because they are adequate for many commercial applications.

The majority of polygons in typical computer generated images are convex and consist of 3 or 4 edges. Consequently, this implementation allows a maximum of 4 edges per convex polygon. If a polygon with more than 4 edges, or a concave polygon must be displayed, it can be broken into many 3 or 4 edge convex polygons.

4.3 The Running Sum Evaluator (RSE)

As indicated in section 3, all the values necessary for the operation of the surface processor (to obtain the values of F1, F2, F3, F4, Zs, I) are of the form:

$$G(Xs,Ys) = A*Xs + B*Ys + C$$

It would first appear that two multiplications and three additions are necessary in order to calculate G(Xs,Ys) at each pixel. However, pixels are evaluated in scan line order. Thus, one can calculate the pixel at G(Xs+1,Ys) as follows:

$$G(Xs+1,Ys) = A*(Xs+1)+B*Ys+C = A*Xs+B*Ys+C+A = G(Xs,Ys)+A$$

Thus, the value of G at pixel (Xs+1,Ys) can be calculated from the value of G at (Xs,Ys) by one addition.

4.3.1 Single Field Operation -

If the frame is to be scanned in sequential line order (line 1, line 2, line 3, etc.) then whenever a new line is begun the following holds:

$$G(Xmin,Ys-1) = A*Xmin+B*Ys+C-B = G(Xmax,Ys)+B'$$

$$\text{where} \quad B' = -B+A*(Xmin-Xmax)$$

and where Xmin = value of Xs at the left of the screen, Xmax = value of Xs at the right of the screen. Thus, the value of G at the start of a new line (Xmin,Ys-1) can be calculated from the previous value of G at the end of the last line (Xmax,Ys) by one addition.

At the beginning of each frame G is set equal to its value at the upper left pixel:

$$G(Xmin,Ymax) = C' \quad \text{where} \quad C' = A*Xmin + B*Ymax + C$$

4.3.2 Double Field Operation -

For simplicity, the previous discussion has not concerned itself with the issue of interlacing. Standard TV monitors scan the frame at 30 Hz in two passes by first scanning the even numbered lines (even field) and then the odd numbered lines (odd field). A vertical retrace precedes each field. Interlacing is performed so as to reduce

flicker. If the system of processors is to generate frames in the sequence in which they are required, it must first generate the pixels on the even scan lines, and then the ones on the odd lines.

In this situation, when the end of the line is reached, one must skip down two lines as follows:

$$G(Xmin,Ys-2) = A*Xmin + B*Ys + C - 2*B = G(Xmax,Ys) + B'$$

$$B' = -2*B + A*(Xmin-Xmax)$$

Thus, the value of G at the start of the new line (Xmin,Ys-2) can be calculated from the previous value of G at the end of the previously scanned line (Xmax,Ys) by one addition.

At the beginning of the even field it is necessary to set G equal to its value at the start of the first even line:

$$G(Xmin,Ymax) = C' \quad \text{if Ymax falls on an even line}$$

$$\text{where } C' = A*Xmin + B*Ymax + C$$

$$G(Xmin,Ymax-1) = C' \quad \text{if Ymax falls on an odd line}$$

$$\text{where } C' = A*Xmin + B*Ymax + C - B$$

However, at the beginning of the odd field it is necessary to set G equal to its value at the start of the first odd line. Thus, C' must be redefined as follows:

$$G(Xmin,Ymax-1) = C' \quad \text{if Ymax falls on an even line}$$

$$\text{where } C' = A*Xmin + B*Ymax + C - B$$

$$G(Xmin,Ymax) = C' \quad \text{if Ymax falls on an odd line}$$

$$\text{where } C' = A*Xmin + B*Ymax + C$$

Thus, it is necessary to change the value of C' between frames as well as between fields.

4.3.3 RSE Implementation -

This analysis clearly suggests the hardware implementation for the evaluation of G at successive pixel locations depicted in figure 7. This circuit is called a Running Sum Evaluator (RSE). When a New-Frame signal occurs, G is initialized to its value at the first pixel

(i.e., C′ is loaded into the accumulator G). As the scan proceeds across the screen, A is added to the accumulator in order to calculate the new value of G. When a New-Line signal occurs, B′ is added to G as shown.

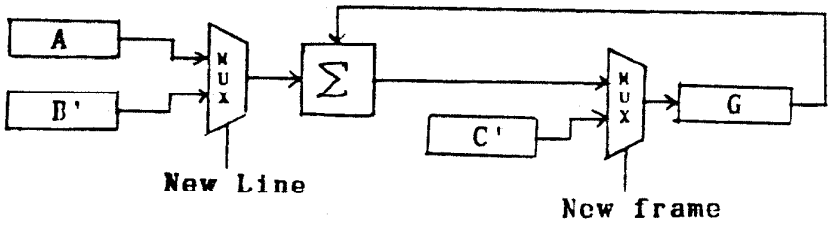


Figure 7 - A Running Sum Evaluator (RSE)

4.4 Simplified Block Diagram Of The Processor

Figure 8 shows a simplified block diagram for a processor. Note that the 18 coefficients (A, B′, C′ for each of the 6 RSEs) must be somehow loaded into the processor for each new view to be displayed. They are not calculated inside the RSEs (see section 5).

As the pixel (Xs,Ys) is latched in, all the RSEs calculate their respective values for the current pixel. If all Fi > 0 then the quadrilateral is potentially visible at this pixel. If Zs is closer to the observer than the closest Zs found by previous processors and the quadrilateral is potentially visible, then the locally computed values of Zs and I are passed on to the processors following this one. Otherwise, the incoming Zs and I are passed on.

Note how the control signals for the RSEs (New-Frame and New-Line) travel down the pipeline with the pixel with which they belong.

4.5 Accuracy Determination

At this point it is necessary to determine the accuracy necessary for the calculation of the functions G (i.e., F1, F2, F3, F4, Zs, I). This analysis determines the number of bits necessary in the data paths of the respective RSEs.

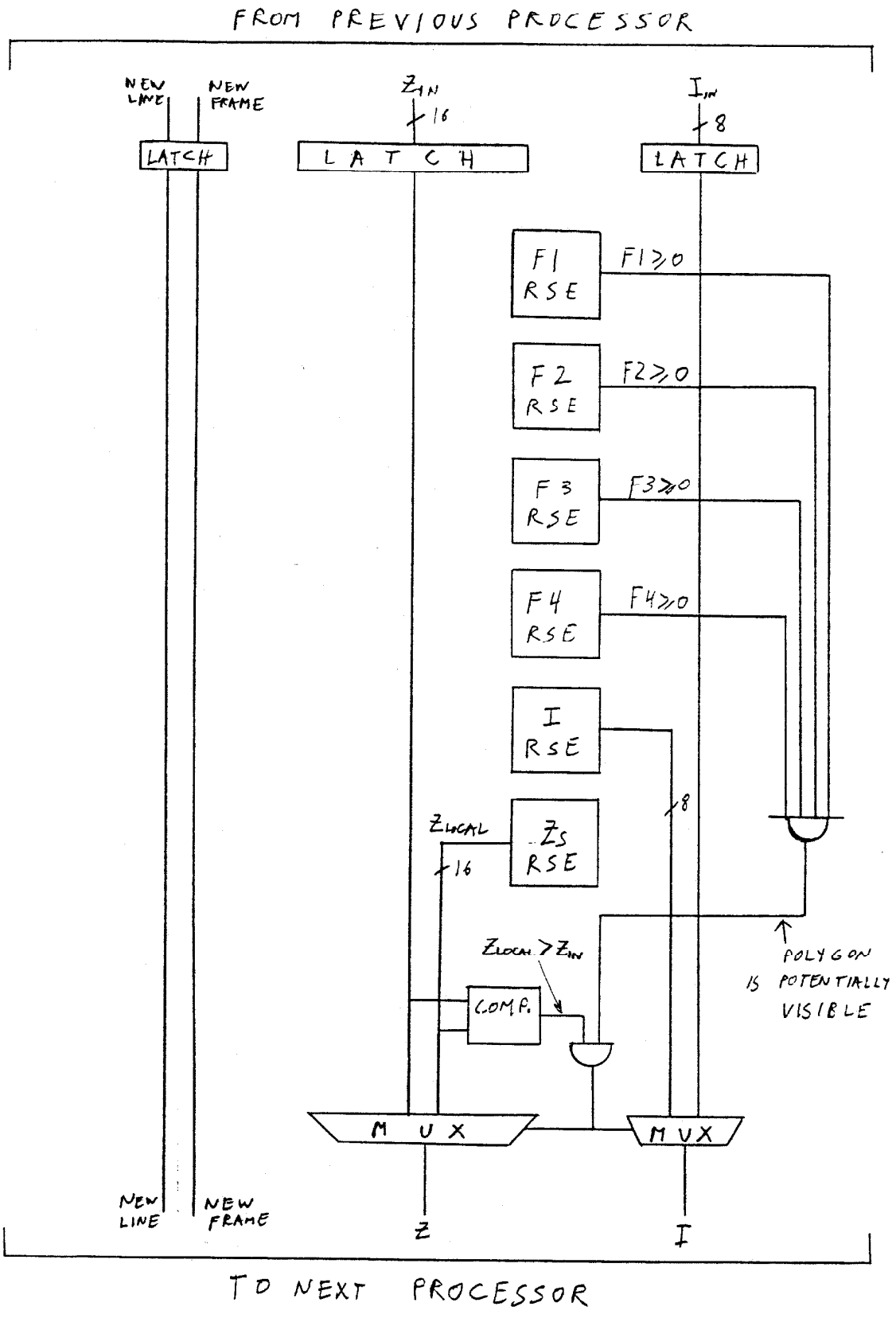


Figure 8 - Simplified Block Diagram of a Processor

4.5.1 Accuracy For F1, F2, F3, F4 -

As described in section 3.1, line L1 can be defined as follows:

$$f1(Xs,Ys) = a1*Xs + b1*Ys + K*c1 = 0$$

$$a1 = Y1s-Y2s \quad b1 = X2s-X1s \quad K*c1 = X1s*Y2s-X2s*Y1s$$

$$\text{thus: } b1' = X1s-X2s+(Y1s-Y2s)*(Xmin-Xmax)$$

$$\text{and } c1' = (Y1s-Y2s)*Xmin+(X2s-X1s)*Ymax+(X1s*Y2s-X2s*Y1s)$$

Note that (A1,B1,C1) = Z1*Z2*(a1,b1,c1) thus, F1 = Z1*Z2*f1. But, Zi $\geq$ 0 so the sign of f1 = sign of F1. Thus, one can use the values a1, b1', c1' as coefficients to the F1 RSE since only the sign of F1 is of use.

If X1s, Y1s and X2s, Y2s are known as 9 bit signed integers (-256 to 255) then it is clear that ai requires 10 bits, bi' requires 19 bits, and ci' requires 20 bits. Since

$$fi(Xs,Ys) = ai*Xs + bi*Ys + ci$$

the accumulator and adder both require 20 bits. However, as section 3.1 indicates, Xs and Ys are never calculated explicitly. Rather, the values Ai, Bi', Ci' can be derived directly from the 3-D coordinates X, Y, Z without the need for any divisions. This, however, generates values which can be considered to be non integral. Without analyzing this problem in detail, let it suffice to say that all coefficients are expanded to 24 bits thus allowing for any roundoff errors.

4.5.2 Accuracy For Zs -

In section 3.2 it was found that:

$$Zs(Xs,Ys) = Az*Xs + Bz*Ys + K*Cz$$

Since Z has been chosen to be limited to the range 0 to 2^16-1, it is clear that 16 bits of "integer" in the accumulator and adder are sufficient. Similarly, since the maximum number of times Az and Bz' are added to the accumulator is 512, the maximum error introduced by the inaccuracy in Az and Bz' is 512*E(Az)+512*E(Bz') (E(x) = difference between truncated x and actual x value). If 8 fractional bits are allowed then:

$$E(Az) = +/- 2^{-9} \qquad E(Bz') = +/- 2^{-9}$$

$$\text{thus: } E(Zs) = +/- \ 512*(E(Az)+E(Bz')) = +/- \ 2$$

Consequently, 24 bits are required to store each of Zs, Az, Bz', and CZ'. It is noteworthy that the value of Zs is only valid when the pixel being considered is within the area of the polygon. When not inside it is easily demonstrated that:

$$Zs(Xs,Ys) = Zs.true(Xs,Ys) \bmod 2\hat{}16$$

where Zs.true is the value of Zs defined on the infinite plane described by the polygon vertices. As a consequence, the following is also true:

$$Cz' = Cz.true' \bmod 2\hat{}16 = Zs.true(Xmin,Ymax) \bmod 2\hat{}16$$

$$Bz' = Bz.true' \bmod 2\hat{}16 = (-Bz+Az*(Xmin-Xmax)) \bmod 2\hat{}16$$

4.5.3 Accuracy For I -

In section 3.3 it was found that:

$$I(Xs,Ys) = AI*Xs + BI*Ys + K*CI$$

Since I is limited to the range 0 to 255 it is clear that 8 bits of "integer" in the accumulator and adder are sufficient. Similarly, since the maximum number of times AI and BI' are added to the accumulator is 512, the maximum error introduced by the inaccuracy in AI and BI' is 512*E(AI)+512*E(BI'). If 10 fractional bits are allowed then:

$$E(AI) = +/- \ 2\hat{}{-11} \qquad E(BI') = +/- \ 2\hat{}{-11}$$

$$\text{thus: } E(I) = +/- \ 512*(E(AI)+E(BI')) = +/- \ .5$$

Consequently, 18 bits are required to store each of I, AI, BI', and CI'. It is noteworthy that the value of I is only valid when the pixel being considered is within the area of the polygon. When not inside it is easily demonstrated that:

$$I(Xs,Ys) = Itrue(Xs,Ys) \bmod 2\hat{}8$$

where Itrue is the value of I defined on the infinite plane described by the polygon vertices in (Xs,Ys,I) space. As a consequence, the following is also true:

$$CI' = CItrue' \bmod 2\hat{}8 = Itrue(Xmin,Ymax) \bmod 2\hat{}8$$

$$BI' = BItrue' \bmod 2^8 = (-BI+AI*(Xmin-Xmax)) \bmod 2^8$$

4.6 Detailed Block Diagram

Figure 9 depicts a detailed block diagram of the processor. It is essentially the same as figure 8 but with more implementation details added. Note that now the processor contains 4 stages of latching on the pipeline path rather than 1. This is done so as to reduce the the amount of serial computation necessary in each pipeline step. With this arrangement 4 cycles are required for a pixel to pass through each processor. Nevertheless, a new pixel emerges from the processor each cycle.

4.6.1 Pipeline Operation -

In the following discussion, a pixel is followed through the processor as it passes through each of the 4 pipeline stages. While a pixel is in stage A (see figure 9) its values (Zin,Iin) are not affected. However, the RSEs calculate whether the polygon to which they are assigned is potentially visible (the Out Bus in figure 9), and what the Zlocal and Ilocal are if it is visible. This information is passed along with the incoming pixel to stage B.

In stage B the Zlocal and Zin are compared to determine which is closer to the screen. This information is combined with the decision of the Out Bus and is sent to the next pipeline stage.

In stage C the multiplexer choses to either pass on the incoming Zin, Iin (if this pixel is not inside the polygon assigned to this processor, or if the polygon is behind the incoming pixel) or to substitute its own Zlocal and Ilocal. The new pair of Z, I is passed to the next pipeline stage.

In stage D the (Z,I) pair propagates from one processor to the next. No processing is performed in order to allow the pad drivers/receivers time to operate.

Note how the control lines New-Frame (indicating that the RSEs should initialize to their respective C') and New-Line (indicating that the RSEs should add B' instead of A) travel along with the pixel with which they belong.

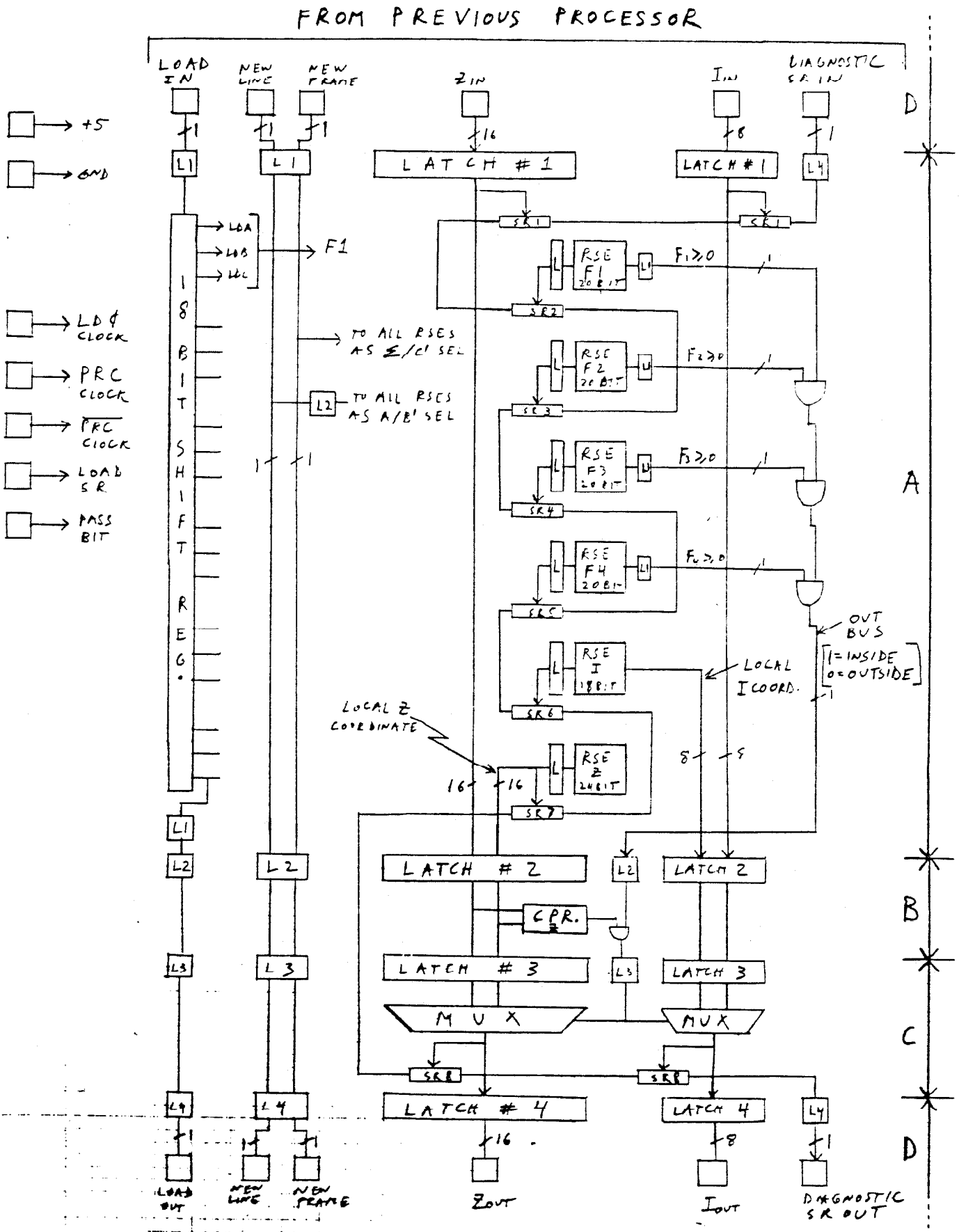


Figure 8 - Block Diagram of a Processor

4.6.2 Loading Of Coefficients -

It is necessary to somehow load the 18 coefficients (A, B´, C´ for each of F1, F2, F3, F4, Zs, I) representing a quadrilateral into the RSEs every time a new frame is to be displayed (typically this occurs every frame, i.e., at 30 Hz). Since a 24 bit data path already exists through all of the processors (16 bits for Zs and 8 bits for I) and since all of the coefficients are 24 bits or less (not accidentally) it is desirable to devise a means by which this data path can be used for the loading of coefficients.

This function is accomplished through the use of the Pass and Load-In/Out signals described below. Referring to the left part of figure 10, one can see how pixels travel from one processor to the next. The path of each pixel through the pipeline of processors is drawn versus time, and each leaves a diagonal track as shown. The start of the pipeline is shown at the top of the figure where it is indicated that background intensity and Zs depth are inserted into the pipeline.

Consider now the last pixel of a frame (indicated in the figure). The first pixel of the next frame does not follow it immediately because it is necessary to allow the TV monitor to perform a vertical retrace (typically 1.2 ms). Thus, after the last pixel exits the pipeline, the Pass bit is set. This bit forces the multiplexer control to always pass through the incoming Zs and I (now, however, these data lines are just a 24 bit word not related to Zs or I). The pipeline becomes essentially a long shift register.

At this time the Load-In bit is sent into the pipeline along with the first coefficient (A) of the first RSE (F1) of the first processor. It is followed in the next cycle by the B´ coefficient, etc. until all 18 coefficients of the first processor are inserted into the pipeline. These are followed by the coefficients of the second processor, etc. until all coefficients of all of the processors are inserted into the pipeline.

As shown in figure 9, the Load bit does not travel along with the pixel with which it was inserted. Instead of spending 1 cycle time in section A of the pipeline in each processor it spends 18 cycles there. This is indicated in figure 10 as a flat double line (i.e., the load bit is remaining in the same processor). During each cycle that the load bit spends in section A it enables the loading of the appropriate coefficient from the 24 data lines. This can be visualized in figure 10 as the intersection of the coefficient tracks (diagonals) with the flat double line representing the location of the load bit. After the load bit has loaded all of the RSE coefficients in one processor,

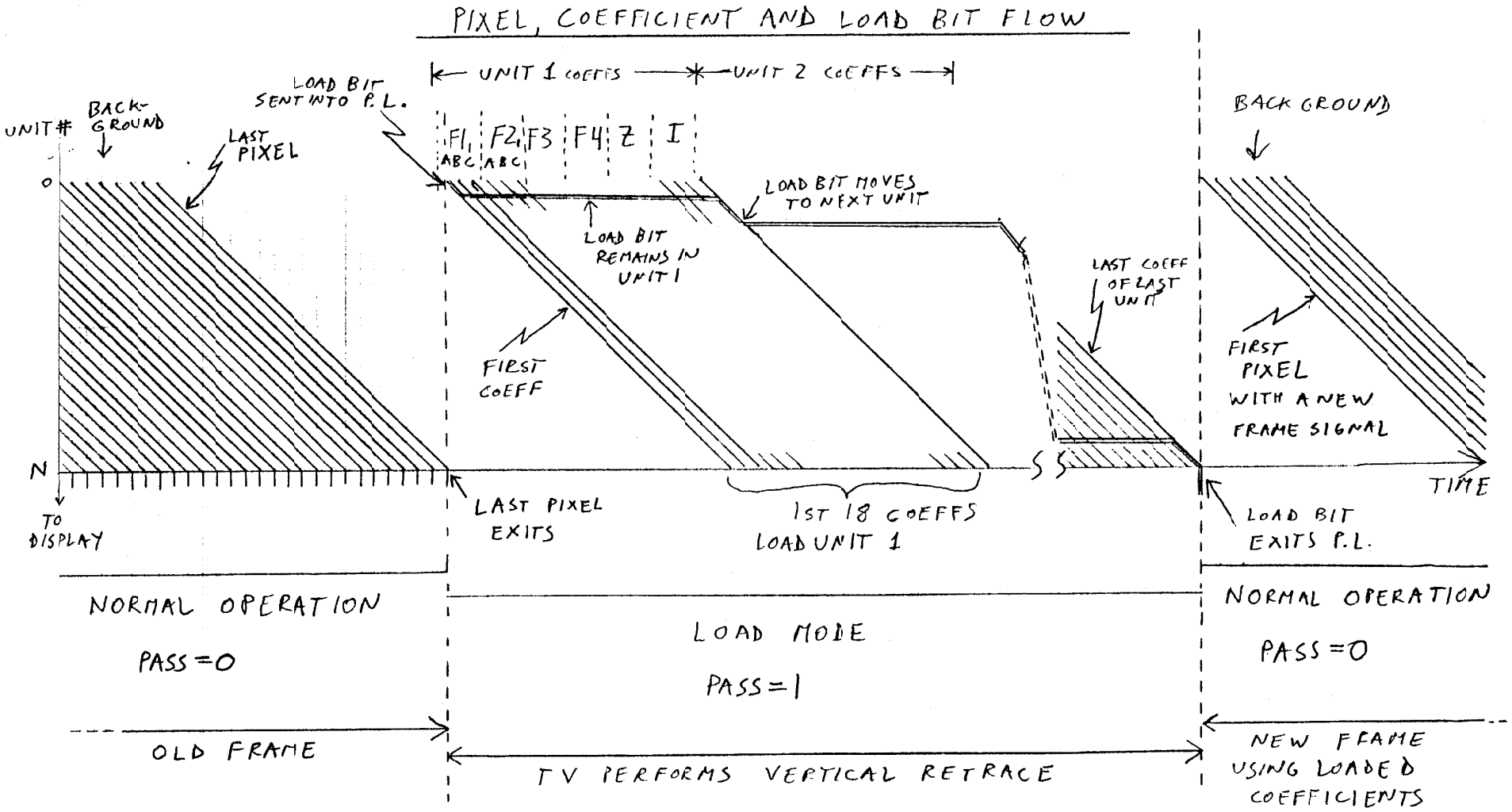

PIXEL, COEFFICIENT AND LOAD BIT FLOW
UNIT 1 COEFFS
UNIT 2 COEFFS
UNIT #
BACK-GROUND
LAST PIXEL
LOAD BIT SENT INTO P.L.
F1 ABC
F2 ABC
F3
F4
Z
I
LOAD BIT MOVES TO NEXT UNIT
LOAD BIT REMAINS IN UNIT 1
FIRST COEFF
LAST COEFF OF LAST UNIT
BACK GROUND
FIRST PIXEL WITH A NEW FRAME SIGNAL
0
N
TO DISPLAY
LAST PIXEL EXITS
1ST 18 COEFFS LOAD UNIT 1
LOAD BIT EXITS P.L.
TIME
NORMAL OPERATION
PASS=0
LOAD MODE
PASS=1
NORMAL OPERATION
PASS=0
OLD FRAME
TV PERFORMS VERTICAL RETRACE
NEW FRAME USING LOADED COEFFICIENTS

FIG. 10

it travels with the first coefficient of the next processor through the pipeline (shown by the short diagonal double line) to the A section of that processor where it again remains for 18 cycles.

Eventually, the Load bit exits the pipeline. At this point the pipeline idles until the vertical retrace of the TV monitor is complete (this implies that the loading of the coefficients can not take longer than the TV retrace time). When the retrace is complete, the Pass bit is disabled thus again enabling the multiplexer logic. The background color of the first pixel of the new frame is then inserted into the pipeline along with a New-Frame signal indicating to all the RSEs (as the pixel and control signal reaches them) that their accumulator should be initialized to their respective C'.

The exact format of each coefficient as it must be loaded on the Zs and I data lines is given in figure 11.

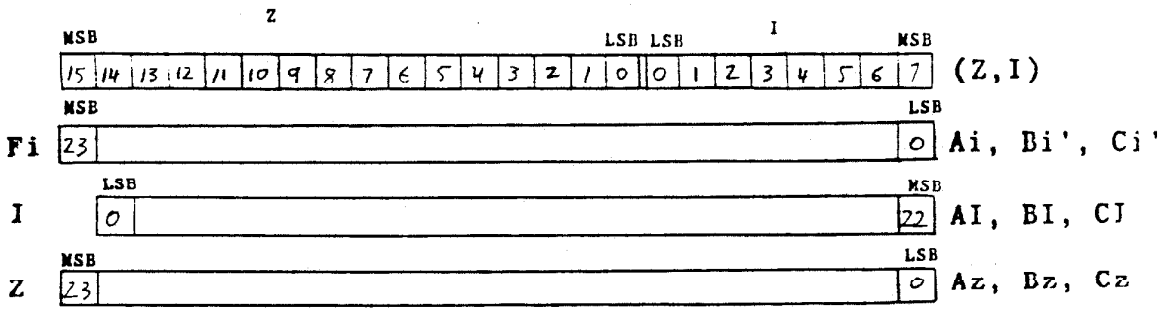


Figure 11 - Coefficient Format

4.7 Testing Of The Processor

It is imperative that the processor be easily and completely testable. This is necessary for three reasons. First, during debugging of the design it is important to be able to isolate errors so that they can be corrected. Second, after manufacture it is desirable to isolate bad parts quickly. Third, after the processor has been inserted into the system there must be a method to test each processor automatically. This becomes especially important as the number of processors grows (e.g., 10K) since if the

pipeline fails, manual isolation of the bad processor is not practical.

When the processor is being tested alone, it is feasible to load it with variously shaped polygons while feeding in a black background at infinity and to compare the generated output with the predicted one. This procedure tests the ability of the Fi RSEs to distinguish the inside of the polygon, and the ability of the Zs and I RSEs to perform the correct linear interpolation. The A, B´,and C´ coefficients can be chosen so as to exercise worst case conditions (long carries) for adders.

After that portion of the processor is known to work, the comparator and the multiplexers can be tested by varying the incoming Zs. Many more tests are possible in this vain.

However, this type of testing is usually not practical after the processor is installed in a large system. This method also gives very limited information about the nature of the failure since the majority of internal values (i.e., the values of the 6 RSE accumulators) are not directly accessible.

These difficulties can be easily overcome though the use of a diagnostic shift register (figure 9). When the Load Shift Register signal is asserted, the shift register is loaded with the values of all RSE registers, the input latch value and the multiplexer output. This information can be shifted out at the pixel rate (one bit per pixel).

All RSE registers can be directly loaded using the normal load protocol and the shift register allows access to almost every storage element of the chip. As a result, each small processing element can be tested independently of the others with only a small amount of functioning circuitry required. This allows fast and complete testing of the entire chip and the ability to pinpoint the location of any errors.

When the processors are used in a system, all the diagnostic shift registers can be connected together. Whenever the pipeline fails, it can be quickly and automatically loaded with a test pattern. Then successive "snap shots" can be taken using the diagnostic shift register. This method provides a much beter opportunity to detect a malfunctioning processor automatically since the only requirement is that the diagnostic shift register work.

4.8 The Layout

The author has implemented a processor as described above using NMOS technology. Figure 12 depicts the layout of the processor chip. Note the close correspondence to the block diagram in figure 9. The 18 bit shift register of the Load bit is distributed on the left of the RSEs (labeled "Load SR"). There are 3 stages of this shift register in each section, corresponding to the 3 coefficients to be loaded in each RSE. The section on the left of the comparator (labeled "Mux Dr") performs the logic and latching necessary to decide which pair of Zs, I is to be sent to the next processor. The latching for the New-Frame and New-Line signals is performed on the periphery of the chip next to the pads.

4.8.1 The Central Array -

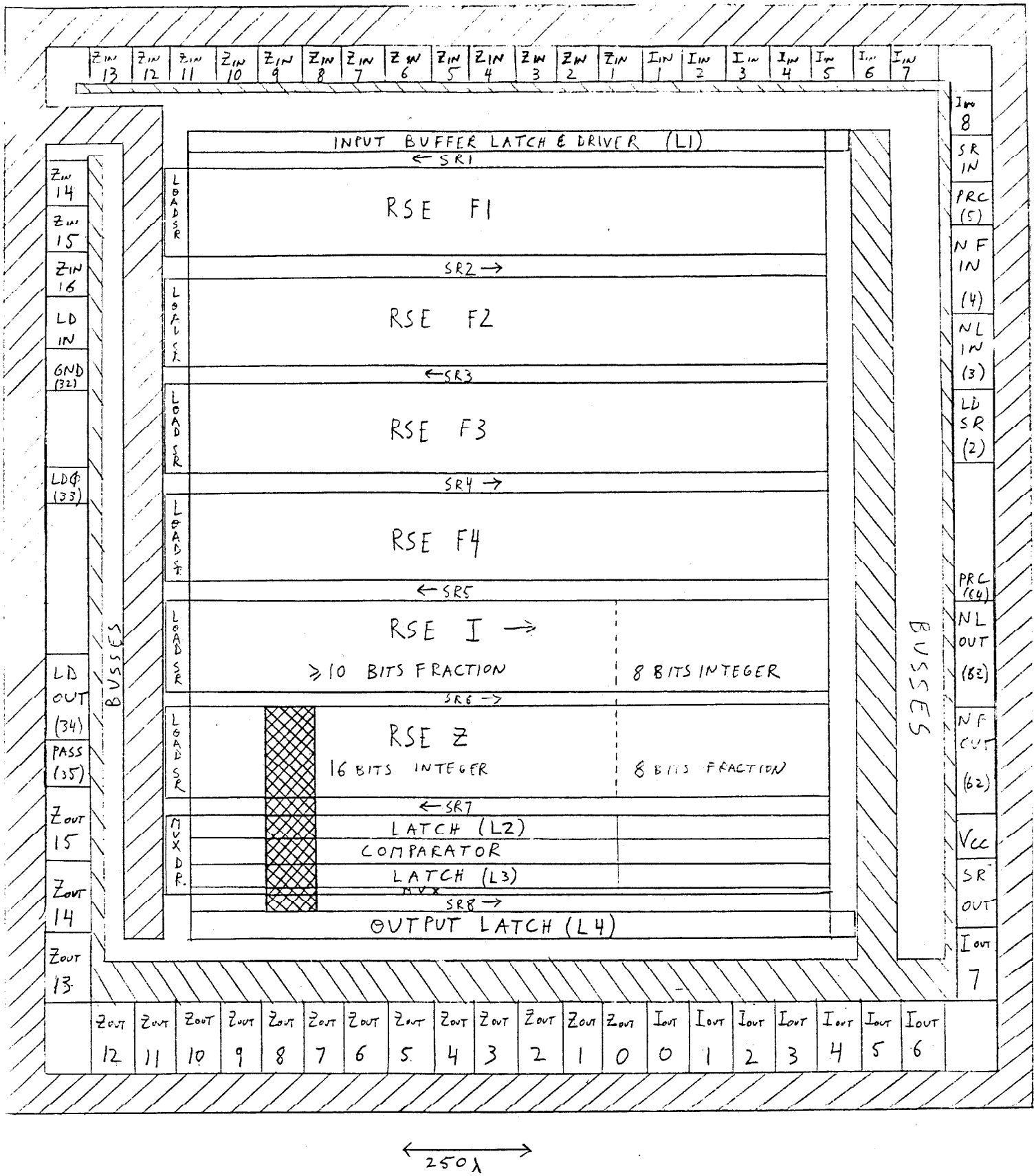
The central array is designed using a bit slice architecture. Data busses run vertically while power, clocks and control signals run horizontally. A more detailed example of the crosshatched portion of figure 12 is provided in figure 13. Power and clock signals are not shown for clarity. Some control signals (e.g., Load A, Load B′, Load C′, Select A, Select B′) are generated on the left or right edges of the RSE, whereas others (e.g., power lines, clocks, Load SR) connect to vertical busses on the left and right of the array.

Five of the six RSEs have their MSB (most significant bit) on the left. However, it is necessary to align the integer portion of the I register (8 bits) so that it falls in those bit slice locations corresponding to the fractional part of the Z register. Thus, the integer part of Zs occupies the left 16 bits while the integer part of I occupies the right 8 bits. For this reason, the MSB of the I register is to the right.

4.8.2 The Pad Arrangement -

Due to the fact that the processor has 63 pins it is especially important that careful thought be given to the pad placement (translating into pin assignments) so as to allow effective layout of a P.C. (printed circuit) board containing many such processors.

Figure 12 labels some of the pin numbers corresponding to the pad locations (pin numbers are in parentheses) for reference. Figure 14 shows a typical processor package and



2639 X 2057 (λ)

Figure 12 - Processor Layout

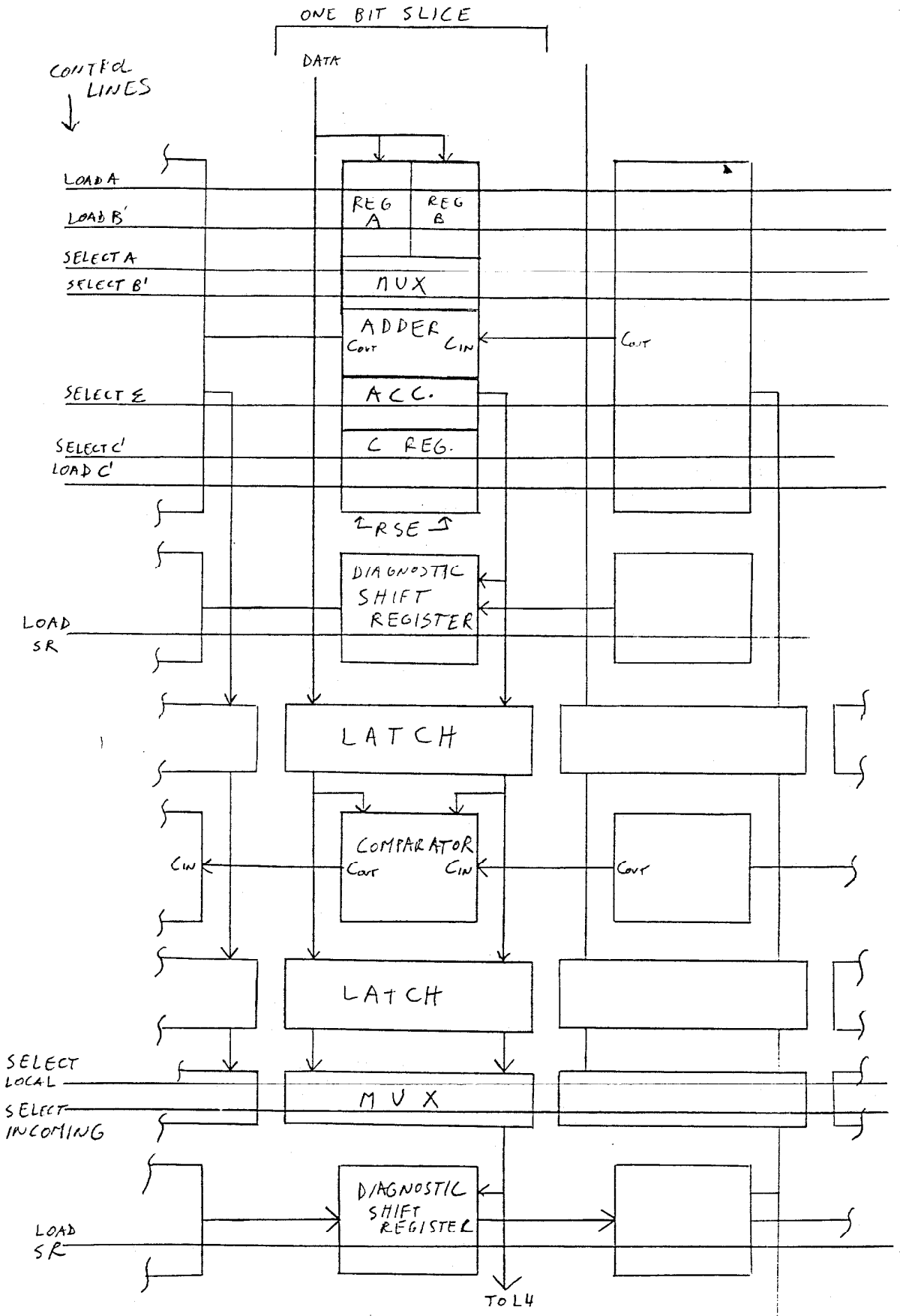


Figure 13 - A Typical Bit Slice

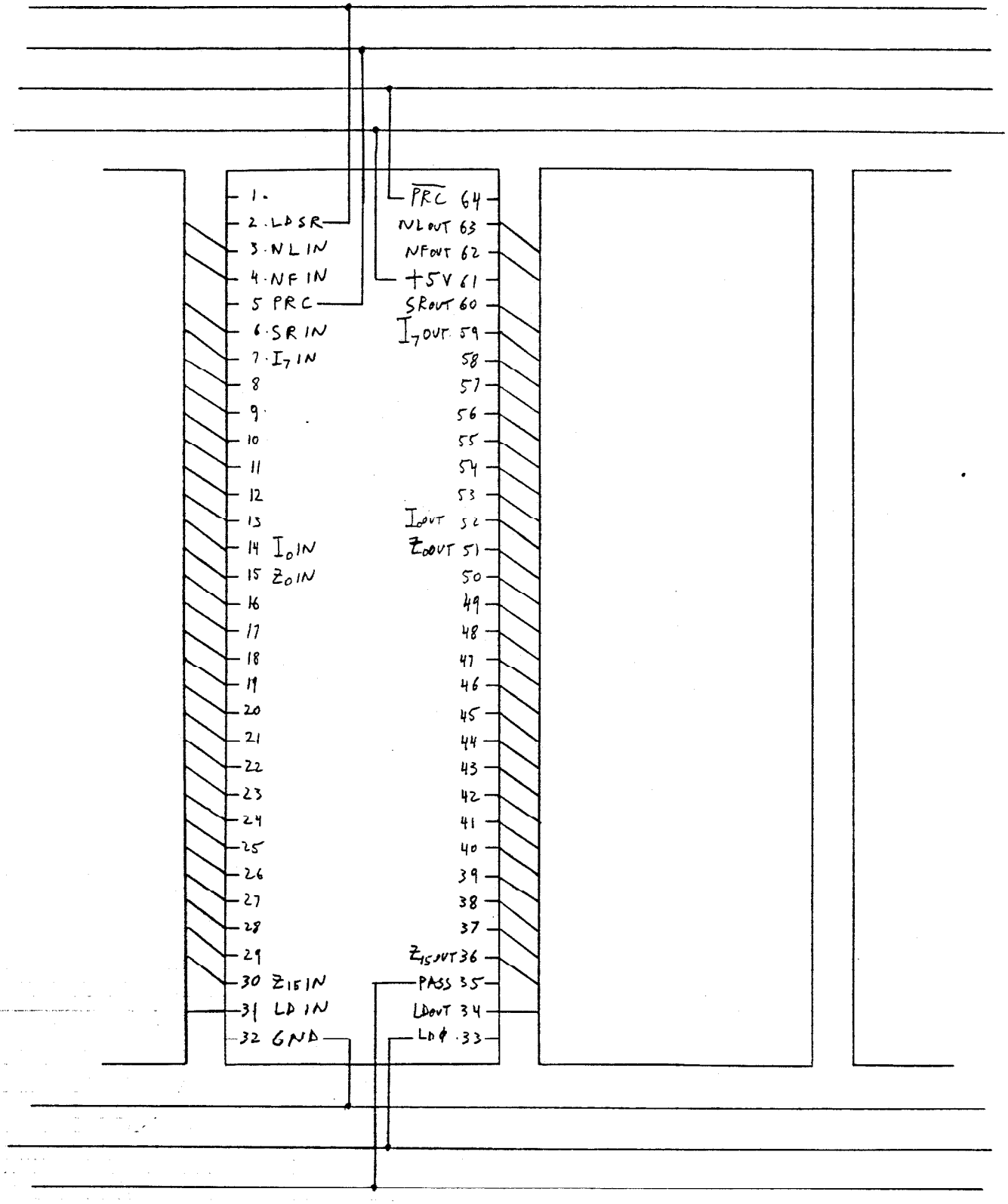


Figure 14 - Processor Interconnection

its required interconnect to the adjacent processors. Note how this pin placement allows close packing of processors on a P.C. board.

4.9 Timing

Figure 15 shows the details of the processor timing. This processor utilizes two non-overlapping clocks (Prc and Prc-bar) and an auxiliary clock (Ld-phi). All input data lines (Load, New-Frame, New-Line, Zs, I, Diagnostic Shift Register) are latched on the falling edge of Prc-bar. All output lines begin to change state on the rising edge of Prc and are guaranteed valid at the following falling edge of Prc-bar.

As indicated, the New-Frame signal must be inserted into the pipeline one cycle before the first pixel of the new frame. Similarly, the New-Line signal must be inserted two cycles before the first pixel of the new line is inserted into the pipeline.

The Load bit is also expected to be inserted one cycle before the first coefficient enters the pipeline. The load bit must only be high during this cycle and not in any following cycles.

Note that the discussions of previous sections assumed that the delays actually required for New-Frame, New-Line and Load were zero in the interest of simplicity. However, as indicated here, they are not zero.

The Load Shift Register signal is actually a clock and thus must be identical to the Prc-bar clock when it is desired to load the shift register.

4.10 Significant Implementation Limitations

4.10.1 Accuracy Of Zs And I -

One significant limitation of this processor is its Z and I word size. In many applications 16 bits of Zs depth are not sufficient, especially when one considers the fact that

$$Zs = K/Z$$

Thus, the resolution of Zs becomes very poor as Z becomes large. Secondly, 8 bits of intensity are not sufficient if one wishes to allow a wide range of intensity values for

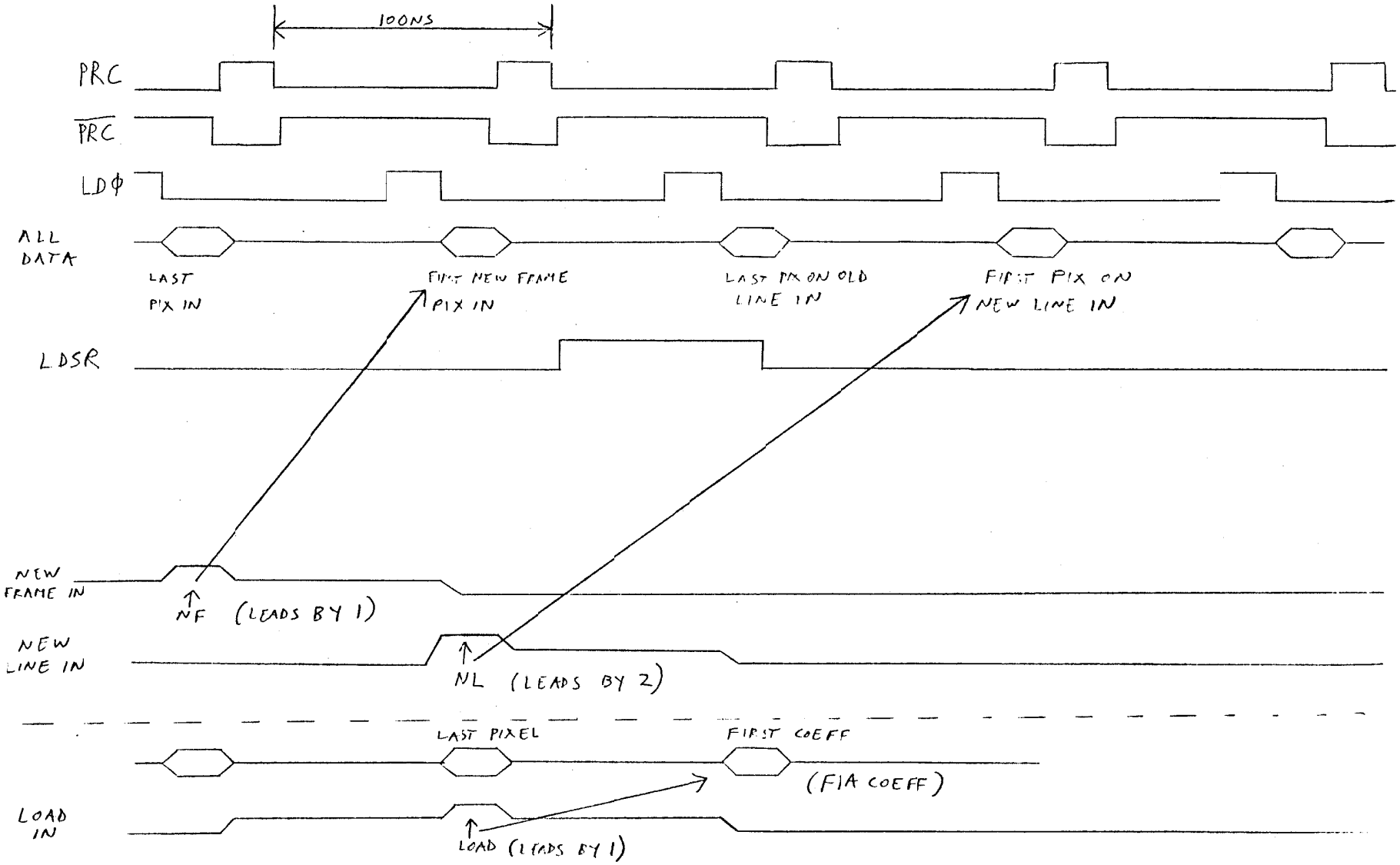


Figure 15 - Timing

each of many colors.

These two limitations can be solved if the data path width is increased appropriately. This, however increases the worst case carry propagation (since the accuracy is larger) and consequently increases the cycle time. However, the cycle time can be decreased by implementing more pipeline delays in each processor. This allows additions (and comparisons) to be performed in many stages thus lowering the worst case carry delay (see also section 5.2).

4.10.2 Raster Size -

A second limitation is the resolution of the raster. The present 512x512 raster is too coarse for some applications where small details are important. In order to solve this problem, it is necessary to both decrease the cycle time (as indicated above) and increase the accuracy of the RSEs (since the errors are now multiplied by a bigger raster size). See also section 5.2.

4.10.3 Coefficient Loading -

Another limitation of this design is well documented in figure 10. As can be seen, in principle, there is no reason why the first coefficient can not follow the last pixel immediately into the pipeline. The present design can not do this because the Pass bit is not pipelined and consequently the pipeline must be clear of all pixels before they are all (simultaneously) turned into a shift register for the coefficients. This causes some wasted time during the loading of coefficients.

This problem can be easily overcome by adding logic on each chip which effectively sets the Pass bit for that processor when it first encounters the Load Bit and resets it whenever it encounters a New-Frame signal. This has the effect of pipelining the pass bit without the need for 2 extra signals (Pass-In and Pass-Out).

4.10.4 Number Of Pins -

The large number of pins required for this implementation is undesirable because it leads to a large package size. This, in turn, causes a low packing density on P.C. boards. This problem will become less significant as the number of processors per chip increases. However, it

is possible to multiplex the 24 data lines on less pins.

For example, one could transmit the 24 bits as 3 sets of 8. Note that this implies that each set must be transmitted in 1/3 of the pixel cycle time. Thus, strong pad drivers (which use a large amount of power) are required.

5.0 ANALYSIS OF A SYSTEM OF PROCESSORS

Figure 16 depicts a block diagram of the hidden surface eliminator shown in figure 2. Its main component is a pipeline of the processors described above. The coefficient calculator provides an interface between the geometry processor and the pipeline.

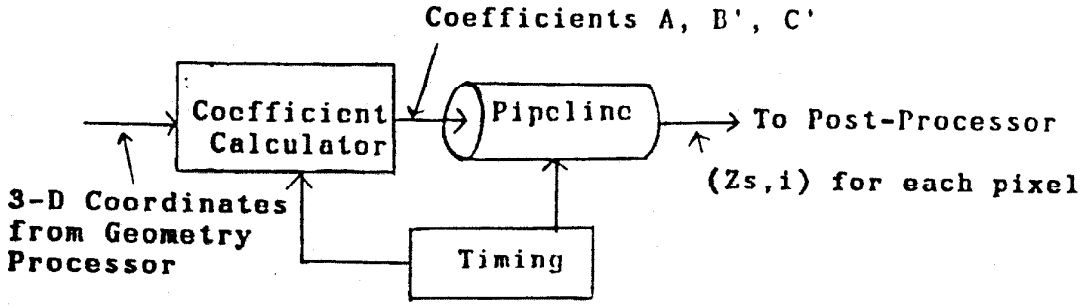


Figure 16 - Hidden Surface Eliminator

The geometry processor provides the coefficient calculator with a new set of 3-D coordinates of vertices of the polygons which are potentially visible for each new frame. The coefficient calculator then (optionally) stores these values and operates on them to obtain the coefficients for the processors using the formulas provided in section 3.

Each processor (i.e., each quadrilateral) requires 6 triplets of coefficients (triplets of (A,B´,C´) for the 6 RSEs (F1, F2, F3, F4, Zs, I)) each frame (or field, if in double field operation). The coefficient calculator has (at least) one full field time to calculate the values of the coefficients. These coefficients can be stored and when the pipeline is ready to accept the coefficients, they can be quickly loaded into the pipeline (one every 100 ns). The calculator can then begin to process the next field.

Referring to section 3, note that the coefficient calculator can calculate all the necessary coefficients from the 3-D coordinates of the polygon vertices without ever having to perform the perspective transformation explicitly. In fact, none of the calculations are data dependent and the only divisions required occur in the inversion of the matrix required for the calculation of the Z and I coefficients. Thus, the coefficient calculator can be an extremely simple processor.

If the TV monitor is being used in double field operation (see section 4.3.2) the coefficient calculator must load the coefficients into the pipeline before each field. However, it is not necessary to recalculate the A or

B′ coefficients. The C′ coefficient is different but only by the addition or subtraction of B.

Note that while the geometry processor and the coefficient calculator have been conceptually separated in this discussion, they can in fact be combined into one entity since they perform very similar kinds of arithmetic operations.

5.1 Simultaneous Coefficient Loading

A difficulty arises when the number of processors becomes large. This is due to the fact that the vertical retrace time of a typical TV monitor (NTSC standard) is 1.2 ms. Thus, if 100 ns are required for the loading of each coefficient, only (roughly) 600 processors can be loaded during vertical retrace.

A simple solution to this difficulty is to separate the pipeline of processors into sections of 600 (figure 17). During normal operation, the sections are connected end to end. However, after the last pixel has gone all the way through one of these sections, it can begin to be loaded. Thus, in effect, the pipeline is disconnected into sections for purposes of loading and, when completed, it is joined back together. Due to the fact that now all sections can be loaded simultaneously, the loading time is no longer a limitation on the number of processors in a system.

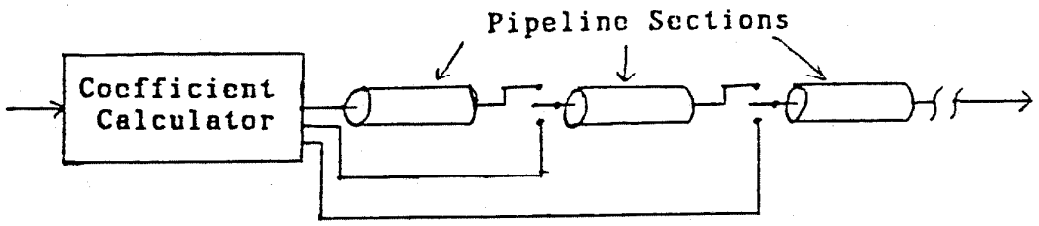


Figure 17 - Sectionated Pipeline

A long pipeline does, however, increase the delay between the time that a pixel is inserted into the pipeline and the time that it emerges from the other end. This problem may become critical in certain applications (e.g., flight simulation) where it is important for the image not to lag far behind. One solution to this problem is to create a tree structure an example of which is shown in figure 18. The comparator units shown operate identically to the ones which are incorporated inside each processor. That is, they pass through the (Zs,I) pair which is closer to the observer. These comparators must, unfortunately, be

made separately since, even though the early discussion of the processors separates the surface processor from the comparator, the actual implementation incorporates the comparator next to each surface processor. This decision was made in view of the large number of pins which would be required for the implementation of a comparator (i.e., 3*(8+16) = 72 pins).

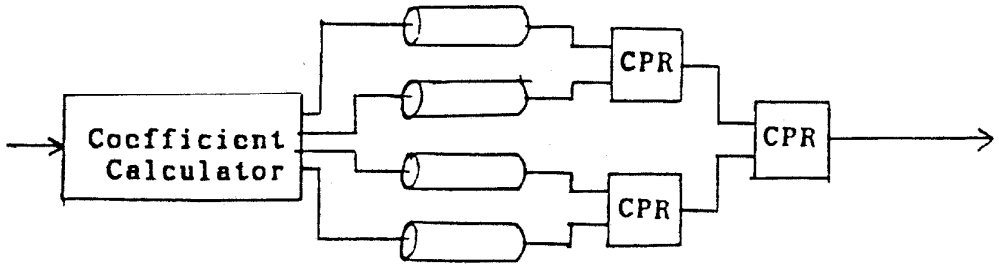


Figure 18 - Pipeline Tree

A pixel enters all 4 pipelines (in this example) at the same time (assuming that all pipelines are the same length). The pixels from the individual pipelines then compete to decide which one of the 4 candidate pixels is the closest. Note that now the pipeline delay has been cut almost by a factor of 4 (since all 4 pipelines work in parallel).

5.2 Parallel Pipeline Operation

As might be expected, it is possible to operate multiple pipelines in parallel in order to decrease the effective pixel cycle time. Assume that it is desired to divide the effective cycle time by N (where N divides the number of pixels on a line evenly). Then, N identical pipelines can be operated in an interleaved manner. That is, each pipeline is caused to generate every N'th pixel each cycle time. Thus, since there are N pipelines, N pixels are available each cycle time thus decreasing the effective cycle time by N.

Clearly, in this arrangement, N processors are assigned to each quadrilateral (one processor in each pipeline). The A and B' coefficients for these RSEs are the same in each processor. However, C' (the initial RSE value) is different to indicate the fact that each processor starts at a different pixel.

6.0 CONCEPT EXTENSIONS

The significant problem of aliasing is not addressed by this method. This problem is due to the fact that the 3-D image which is to be represented is not, in general, band limited in spatial frequency. This approach to hidden surface elimination basically samples the contents of the image at the center of each pixel. Thus, as the well known sampling theorem states, aliasing occurs since the image being sampled contains spatial frequencies greater than half the sampling frequency. This effect derives its name from the fact that under-sampling results in high frequencies being aliased (or wraped around) as low frequencies. This results in disturbing Moire patterns.

Thus, a clear need exists for eliminating or ameliorating this problem in order to make the resulting image more appealing (less annoying) to the viewer.

While planar approximations to curved surfaces are adequate for some purposes (accompanied by Gouraud shading, of course) it is still desirable to be able to describe curved surfaces as such. Thus, a processor which fits into the same pipeline described above but which can describe a non-planar surface is desirable.

Finally, many applications which require very realistic depictions of simulated environments need textured surfaces. A processor which produces a textured I value would be highly desirable in such applications.

7.0 CONCLUSION

This thesis has outlined the design of a low cost VLSI-based dynamic raster display system which provides a sense of reality through the use of real-time hidden surface elimination and shading.

A comparatively simple and ordered approach utilizing highly parallel pipelined processing provides a practical solution when implemented in current VLSI technology. In fact, VLSI has proven to be uniquely appropriate for this approach. The advances in this technology promise to make this approach even more appealing in the future.

A complete display system which makes use of the display system described above has been analyzed. This system is simple, easily expandible and highly reliable (due to the highly testable nature of the individual processors).

8.0 ACKNOWLEDGEMENTS

The author wishes to acknowledge the aid of the following people in the development and implementation of this concept: Dr. James Blinn, Dr. Ivan Sutherland, Dr. Danny Cohen, Dr. Charles Seitz.