12

A VLSI Architecture for Image Composition

Christopher D. Shaw, Mark Green, and Jonathan Schaeffer

This paper describes a new parallel architecture for performing high-speed raster graphics. A central host broadcasts graphical objects to a number of identical graphics processors. Each graphics processor produces a raster depicting its graphical object on a transparent black background, and passes the raster to a leaf of a tree of VLSI processors called *Compositors*. Each Compositor combines a pair of rasters, performing anti-aliased hidden surface removal, and passes the composed raster to the next level of the tree. Appearing at the root of the tree is the final raster containing all objects at the correct depth with hidden surfaces removed.

This paper gives an outline of the algorithm by Duff that the Compositor will implement. The algorithm proves to be too complex for our implementation technology, so a modification of Duff's algorithm is introduced. The high-level design of the dataflow part of the VLSI chip which implements this modified algorithm is then presented, followed by performance simulations and conclusions.

1. Introduction

To date, research in the area of specialized 3-D computer graphics architectures has concentrated mainly in two areas:

1) Geometry Pipelines, wherein the geometric operations of rotation, transformation, scaling, projection and clipping are performed by a pipeline of multipliers. The operands of a geometry pipeline are the objects to be rendered, for example, vertices of polygons [3, 4]. Speedup over traditional general-purpose processors is limited by the number of multipliers in the pipeline. Of course, better VLSI technology will yield greater speedup but, in architectural terms, only simple duplication of the geometry pipeline will improve matters significantly.
2) Rendering processors, which take geometric descriptions of the picture to be rendered to draw raster images [9, 12]. Here, better VLSI will yield higher speeds, but simple duplication will not work without sacrifice. If two renderers draw pixels on the same raster, some means of collision avoidance must be developed.

The method of graphics parallelization that we propose breaks up the graphics production task by object. The modeling subtask in a host processor distributes graphical objects to a number of independent general-purpose Graphics Processors (*GP*). Each GP performs the geometry and rendering tasks on its own graphical object without communication with other GPs. Each GP creates a full coverage-enhanced raster which displays its graphical object on a transparent black background. Each GP could be as simple as a microprocessor or as complex as a geometry pipeline feeding a rendering processor.

One clear disadvantage of the object-level approach is the combination task that must be performed upon the N rasters that are produced by the N *GP*s. We have developed a unique VLSI architecture which solves this problem by implementing an anti-aliasing variation of Z-buffer called Composition [6, 10]. This architecture utilizes parallelism in a way that has not been explored satisfactorily to date. In particular, while the Host-GP setup shown in section 3 is not new, other methods that have been proposed for combining the resultant rasters have been unsatisfactory.

Our Compositor innovation makes this form of graphics parallelism feasible, since it solves the major problem of post-hoc raster combination in a non-restrictive manner. Moreover, the hardware solution simulated for this paper combines rasters in real time.

Such a possibility opens new avenues of research in parallel graphics since, while proposals are nice, only real experience with parallelism fosters true understanding of the problems at hand. Hopefully, this paper will be a tool to help researchers gain a true understanding of the best parallel graphics methods.

2. Prior Work

The object-level approach has been proposed before, namely in papers by Weinberg [13], and by Fussel & Rathi [8]. The system by Weinberg, shown in Figure 1, has four types of processing elements labeled "O", "B", "C" and "F". The O elements are the object processors which receive object descriptions from the host and output pixel spans where the objects cover the screen. Areas not covered by an object do not produce pixels.

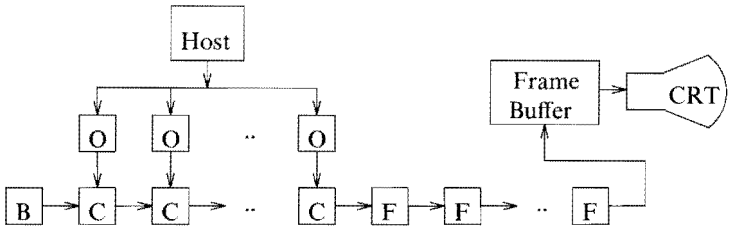

Figure 1: Object-Based Architecture by Weinberg.

The C elements are comparators which collect a list of contributions to the pixel. The image contributions are neighbourhoods of the current pixel. A background neighbourhood is fed by the B element of the pipe, and as the pixel passes from comparator to comparator, the adjacent object processor is checked for contributions. If an object is either fully or partially visible, it is added to the growing depth-sorted list passing through the comparator pipeline. Finally, the F processors perform a filtering process which resolves the final scan line colour from the contribution list computed by the comparators.

The advantages of this system are that it performs anti-aliasing, and it allows for real-time graphics. The disadvantage is complexity in the comparators and filter processors, since each must manage a variable-length list of pixel contributions. If each comparator and filter processor were a VLSI chip, each would need a substantial amount of buffer memory to hold the extra pixel information as it passed through the pipeline.

Also, given N object processors, it is possible that each one will have a pixel contribution at every point on the screen. This means that to maintain the raster throughput provided by each object processor, each comparator and each filter processor must maintain a throughput of N times that of an object processor. Clearly this is the worst case, but given the rapidly growing image complexity of modern computer graphics, N will have a fairly small upper bound unless the VLSI technology for the comparators and filters is significantly advanced. In his paper, Weinberg estimated that picture production speed is depended upon the sum of all polygon perimeters in the scene measured in pixels, as well as the number of processors and the screen size.

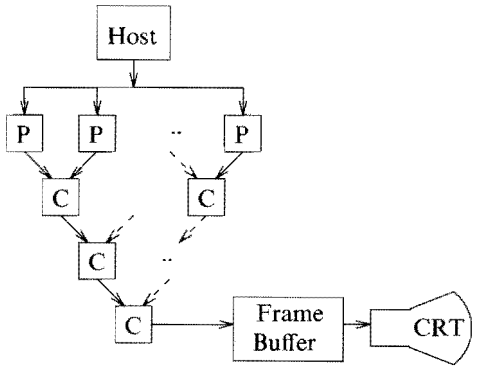

Figure 2: Fussel and Rathi's Object-Based System.

In Fussel and Rathi's system hardware, the host simplifies the model into triangles and distributes it to a number of processors denoted by "P" in figure 2. Each "P" contains a

geometric transformation engine and 1000 simple triangle processors, each of which contains only registers, I/O and a couple of adders. Each triangle processor receives one transformed triangle, renders it, and feeds it to a tree of comparators denoted by "C" in Figure 2. The comparators simply perform Z-buffer: the nearest contribution is the one passed on, and all other pixels are thrown away.

In sum, each object-level system has one of two correlated disadvantages: Weinberg's system performs anti-aliasing, but does so at the expense of low data throughput due to the buildup of pixel contributions. Fussel and Rathi's system allows for rapid lock-step pixel production, but does not perform anti-aliasing due to its use of Z-buffer. Our system uses a hybrid Z-buffer approach which suffers from neither of these problems.

3. The Composition Architecture

We have designed a unique pipelined VLSI chip which performs the combination task upon the N rasters that are produced by the N GPs. The combination is performed by a binary tree of composition processors called *Compositors*. Each Compositor takes two rasters in the coverage-enhanced Z-buffer format required for the composition operation, and composes this pair of rasters into one raster of the same format. Since the composition operation is associative and commutative, we can take a pair of composed frames and compose them also. Thus, we can form a tree of $N-1$ Compositors. If M is the height of the tree, we can combine $N = 2^M$ rasters into one final raster, as shown in Figure 3.

The N leaves of the Compositor tree are the N GPs. Each GP feeds one input of a Compositor. Given that $M > i > 0$, at each level i of the tree, 2^i Compositors combine 2^{i+1} raster inputs to form 2^i raster outputs. These 2^i outputs feed 2^{i-1} Compositors at the level below, and so on until the root of the tree composes the last 2^1 raster inputs to form the final raster output. The output of the root Compositor feeds data to a frame buffer which displays the raster on a CRT.

In total, there are $N-1$ Compositors, with the root of the tree producing the final raster picture of the entire model created by the modeling subtask. Each Compositor takes a fixed amount of time to compose each pair of pixels, so the root Compositor can feed results to the frame buffer at a fixed rate.

The advantages to consider with this system are as follows: The system can be expanded to any practical degree, simply by duplicating the whole system and adding a new Compositor to compose the two streams at the root. This system offers O(N) parallelism. To double nominal performance one need only double the amount of GP and Compositor hardware and add one Compositor to compose the final two rasters. The increase in composition time is equal to the time to pass one pair of pixels through a Compositor. Of course, one can take advantage of the performance increase either by increasing the production speed of a scene of fixed complexity, or by increasing the complexity while holding scene production speed constant.

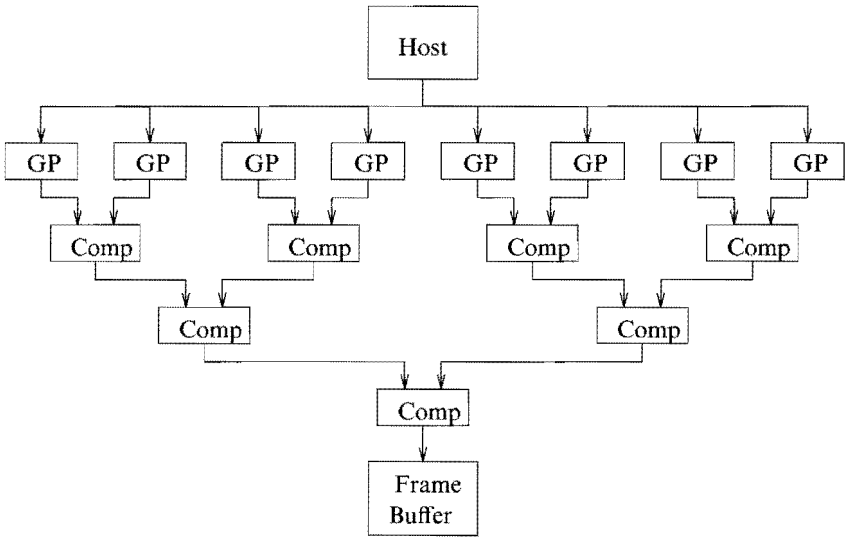

Figure 3: High-Level View of the Composition Architecture.

Another advantage is that one could install different GPs at the top level, which means that different types of picture modeling could be performed for different parts of the same picture, as appropriate.

With a change in the control structure, it is equally possible that this system could be built as a linear pipeline, in a manner similar to Weinberg's proposal. In this configuration, each Compositor takes data from the previous pipe element and from its local GP. Each Compositor passes its results to the next in the pipe, and the last passes its results to the frame buffer.

The advantage would be easy scalability to any number of processors without much effort. The problem lies in error accretion. The number of Compositors that a pixel must pass through is an average of $\frac{N}{2}$ in the linear setup versus exactly $\log_2 N$ with the tree arrangement. Since each Compositor approximates Duff's algorithm, errors will build up after a number of composition steps. From an error point of view, the least steps, the better, which is what the tree offers. Also, the latency from input to the output in the tree system is $\log_2 N$ vs. $\frac{N}{2}$ for the linear setup, but this is not likely to matter given the speed of each Compositor.

With either organization, a problem to be confronted is the communications of polygon data from the host system to the GPs. A simple broadcast bus may be sufficient, but this may not be the case with larger systems. This issue is beyond the scope of this paper. Similarly, issues such as windowing and so on are not addressed here.

In a related issue, the intended use for this system is animation, which means that it may be used to draw complex rasters one at a time. Speed gained due to parallelism may be lost to the overhead of broadcasting a lot of objects to the GPs. The question of how to manage graphics data in a parallel environment is unexplored, and we hope that this paper will foster interest in this area.

4. Which Algorithm?

The algorithm to perform composition relies upon coverage information stored in each pixel much in the same way that the Z-buffer algorithm stores depth information. Two algorithms have been developed and reported in the literature which use coverage data in two forms. Carpenter's A-Buffer system [1] is an anti-aliasing version of Z-buffer. The coverage measure used is a bitmap of the pixel at subpixel resolution. Each bit of the bitmap indicates whether its fraction of the pixel is covered by a polygon from the source raster. The bitmap approach to coverage estimation has antecedents in work by Catmull and by Crow [2, 5], which suggest the use of subpixel information to perform anti-aliasing. Similarly, work by Fiume et al [7]. advocates enhancing Z-buffer with subpixel resolution information for the purposes of parallel implementation on a shared-memory machine.

In Carpenter's implementation, the Z-buffer contains either positive Z and colour or negative Z and a pointer to a list of unrendered depth-sorted pixel fragments. When all the pixel fragments have been collected, the top (closest) fragment has its area-weighted colour added to the pixel. Its area, approximated by the bitmap, is removed from legal consideration, and all those pixels underneath are clipped to the top pixel's uncovered area. The process then continues on the next closest fragment. Its weighted colour is added, and its area clips all those pixels under it.

The key restriction with A-Buffer is that pixel contributions must be sorted in order of depth. This introduces two unpleasant problems, the first being that the algorithm does not work correctly when the pixels are out of order. This means that arbitrary pairs of pixels cannot be combined, since the clipping operation must take place in order of depth. If a Compositor were to implement A-Buffer, it would combine arbitrary pairs of pixels. The second problem with A-Buffer is that the sort must be performed on each pixel. With a large number of source rasters, this sort process will take a much longer time than the simple bit manipulation required by the core of the algorithm, since the sort does not have a linear time bound.

From an architectural perspective, A-Buffer requires a two-stage setup like Weinberg's pipeline architecture. The first stage sorts a list of pixel data, and the filter processors perform the contribution calculation of A-Buffer.

Duff's composition method [6] offers a slightly different approach to the coverage problem. Duff stores an area component α with each pixel. α is a real number in the closed range [0.0 $\cdots$ 1.0], where a value of zero indicates no coverage and one indicates full coverage. This component can also represent opacity for pixels of appropriate coverage. That is, if the actual coverage is 1.0 but α is given the value 0.5, the pixel will be "half transparent". Usually, though, α = coverage. When pixels for a source raster are produced, R, G, and B colour values are each multiplied by the coverage value for anti-aliasing. Thus, transparent black is where R, G, B and α all equal zero.

Aside from the addition of α, composition imposes a second change to Z-buffer organization, namely that Z values are moved from the center of each pixel to the pixel's corner. This means that the Z depth will be available to the four pixels that share each pixel corner (except at the raster's edges, of course). The composition takes pairs of rasters and composes them into one raster of the same format, so to compose a number of images, each image is composed with the destination raster. This movement of Z to the corner of the pixel requires that an extra row and column of Z's be supplied at the bottom and right edges of the raster in order to correctly process the last row and column.

A depth sort of the pixel contributions will produce the best results, but Duff's experiments show that ignoring the order of composition causes no error in most situations, and only a small detriment to the picture quality in certain special cases. Thus composition does not suffer the unboundedness of A-Buffer. There is a trade-off however, since Duff's coverage measure does not include any positional information.

Composition also does not have the problem that A-Buffer does with unsorted data, since the four *corner* values of Z are used instead of one simple minimum Z for each pixel contribution. When a pair of pixels are combined, the new corner Z values are the minimum of the respective corner contributions. Thus if one combines the nearest and farthest pixels, and if the new coverage is full, then a pixel of intermediate Z can still make a contribution if one of its Z values is less than the minimum of the nearest and farthest pixels. Of course, sometimes the result will not be quite right due to the linear-intersection algorithm used to determine β. This is much better than the possible shut-out that an unsorted A-Buffer may produce.

5. Duff's Composition Algorithm

With two rasters named Front and Back, compose two pixels $\text{pixel}_{\text{Front}}$ and $\text{pixel}_{\text{Back}}$ by first comparing the four corner Z values of each pixel. If the comparisons all yield the same sign, then the pixel which is in front is the result pixel. However, as shown in three examples in Figure 4, some pixels will intersect: that is, Z comparison in some corners will be the opposite sign of Z comparison in other corners. In this case, we must determine the fraction β, which is the coverage ratio between the two pixels.

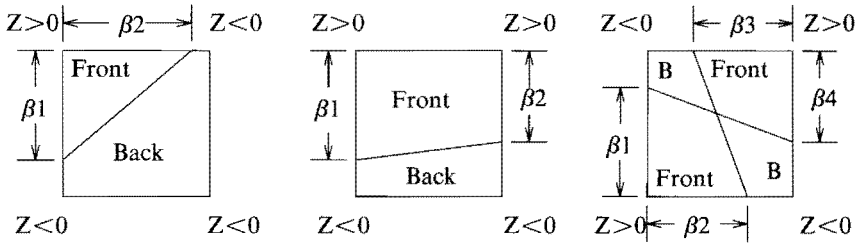

Figure 4: Three Examples of Intersecting Pixel Contributions.

β is determined by finding the points of Z intersection along pixel edges which have corners of opposite sign. These intersection points yield a dividing line between the contribution of $pixel_{Front}$ and the contribution of $pixel_{Back}$. The number β is the fraction of the pixel taken up by $pixel_{Front}$. In the left example in Figure 4, β would be the proportion of the pixel labeled "Front", which equals about 30% of the pixel area.

With β in hand, the following equations are evaluated to find the final values of $pixel_{comp}$. Here just the equation for Red is shown, since the equations for Green and Blue are identical.

$$R_{comp} = \beta \times (R_{Front} + (1-\alpha_{Front}) \times R_{Back}) + (1-\beta) \times (R_{Back} + (1-\alpha_{Back}) \times R_{Front}) \quad (1)$$

$$\alpha_{comp} = \alpha_{Back} + \alpha_{Front} - \alpha_{Back} \times \alpha_{Front} \quad (2)$$

$$Z_{comp} = Min(Z_{Front}, Z_{Back}) \quad (3)$$

6. Compositor Simplification

After equations (1-3) are algebraically simplified, they amount to one minimum operation to calculate Z, one add, one subtract and one multiply for α, and eight multiplies, three adds and three subtracts to calculate R, G, and B. However, β takes more effort since there are four classes of pixel intersection patterns:

Unconfused

In the two patterns of this class, all of the Z comparisons are of the same sign, and β simply equals either 1.0 or 0.0. Equivalent to Z-buffer.

Triangular

In the triangular class, one of the rasters is closest in only one corner, which results in a triangular pixel area being defined. The left sample in Figure 4 shows an example of

a pattern of this class. The area of β for each of the eight possible patterns is calculated using equation (4).

$$\beta = \frac{\beta 1 \times \beta 2}{2} \tag{4}$$

Trapezoidal

Four intersection patterns are of this class, in which one raster is closest in two adjacent corners, and the other raster is closest in the two other adjacent corners, forming a trapezoidal area. The centre example in Figure 4 shows one pattern of this class. The area for β is evaluated using equation (5).

$$\beta = \frac{\beta 1 + \beta 2}{2}, \tag{5}$$

Checker

With the two patterns of this type, one raster is closest in opposite corners, and the other raster is closest in the other two opposite corners, forming a pattern like a checkerboard. The right example in Figure 4 shows one pattern of this class. Equations (6.1-6.3) must be evaluated to get the correct β value for this class:

$$x = \frac{\beta 3 - \beta 1(\beta 3 + \beta 2 - 1)}{1 - (\beta 3 + \beta 2 - 1)(\beta 4 + \beta 1 - 1)} \tag{6.1}$$

$$y = \frac{\beta 4 - \beta 2(\beta 4 + \beta 1 - 1)}{1 - (\beta 3 + \beta 2 - 1)(\beta 4 + \beta 1 - 1)} \tag{6.2}$$

$$\beta = \frac{(\beta 2 - \beta 3)y + \beta 3 - (\beta 4 - \beta 1)x + \beta 4}{2} \tag{6.3}$$

As shown in Figure 4, $\beta 1$ and $\beta 2$ are the proportions of a pixel edge which are taken up by Front on pixel edges where the Z comparisons differ in sign. This fraction is edgeβ, and it is calculated by linearly interpolating the Z values to find where Front and Back meet:

$$\text{edge}\beta = \frac{|\text{diff}_{\text{Front}}|}{|\text{diff}_{\text{Front}}| + |\text{diff}_{\text{Back}}|} \tag{7}$$

Here $\text{diff}_{\text{Front}}$ is the difference between Front and Back **Z**'s in the corner where **Front** is nearest to the viewer. Conversely for $\text{diff}_{\text{Back}}$.

As one can see, the calculation involved is substantial, since for the triangular and trapezoidal classes, two divisions must be performed, and a total of six divisions must be performed for the checker class. Gate limitations in the gate array technology used to implement the Compositor preclude a full floating point calculation. Division is a complex operation to evaluate, and should be avoided if at all possible.

We have performed experiments over a number of raster composition situations using various approximation algorithms for β. Each raster produced by an approximation was statistically compared to a reference raster produced by Duff's algorithm.

The approximation that we will implement samples the Z differences at 9 points on the pixel, and assigns a nonzero weight to each sample if the sample yields **Front** as closest at that point. The weights which gave the best experimental results were $\frac{1}{16}$ for the corner samples, $\frac{1}{8}$ for the edge samples, and $\frac{1}{4}$ for the center sample. The sum of all 9 weights equals 1. This weighting scheme implements a 3×3 Bartlett filter.

This approximation had one of the best mean standard deviations, and was chosen because it had a simple hardware implementation. Experiments showed that this approximation for β had a mean error of 9.7%, and worst-case error of 12.2%. The best approximation tested had a mean error of 9.2%, and a worst-case error of 15.1%. By comparison, ordinary Z-buffer on the same pairs of rasters had a mean error of 54.3%, and a worst-case error of 63.5%. In practice, the approximation is almost indistinguishable from Duff's algorithm [11].

7. Compositor Implementation

Each Compositor takes a pair of rasters pixel-by-pixel, and composes one pixel at a time in a pipelined fashion. Each pixel is six bytes of data, with 16 bits for Z, and eight bits for each of R, G, B and α. Data flows from chip to chip along eight-bit buses.

For the two input streams there are two eight-bit input buses FRONT_DATA and BACK_DATA. These have the associated 16-bit address bus IN_ADDR. The previous row I/O stream has bidirectional eight-bit data buses PREV_FRONT and PREV_BACK with 14-bit address PREV_ADDR and control lines PREV_RD_STRB and PREV_WR_STRB. This I/O stream is needed to store the previous scan line of Z's so that it is available for Z comparisons at the top left and top right corners of the pixel on the current scan line. The fact that only Z's are addressed allows for a narrower address bus.

The output stream has eight-bit output bus OUT_DATA, with 16-bit address OUT_ADDR. Aside from the clock input CLK, there are three miscellaneous control lines: RESET, an input which resets the whole Compositor; START_ROW, an input which indicates that the next pixel fetched is to start at the beginning of the scan line; and OUT_START_ROW, an output which is the input START_ROW delayed by the number of clock cycles it takes to propagate one pixel through the Compositor.

Inside each Compositor, the Z values pass through a series of comparators which determine Z priority at each of the four corners of the pixel, as well as average priorities at the pixel edges and the pixel centre. The sample value arises from the sign bit of the two's complement difference. First, the corner comparisons.

$$\text{Bottom Right} = Z(x,y)_{\text{Back}} - Z(x,y)_{\text{Front}} \tag{8.1}$$

$$\text{Bottom Left} = Z(x-1,y)_{\text{Back}} - Z(x-1,y)_{\text{Front}} \tag{8.2}$$

$$\text{Top Left} = Z(x-1,y-1)_{\text{Back}} - Z(x-1,y-1)_{\text{Front}} \tag{8.3}$$

$$\text{Top Right} = Z(x,y-1)_{\text{Back}} - Z(x,y-1)_{\text{Front}} \tag{8.4}$$

The edge comparisons are as follows:

$$\begin{aligned}\text{Bottom Edge} &= Z(x,y)_{\text{Back}} + Z(x-1,y)_{\text{Back}} - Z(x,y)_{\text{Front}} - Z(x-1,y)_{\text{Front}} \\ &= (Z(x,y)_{\text{Back}} - Z(x,y)_{\text{Front}}) + (Z(x-1,y)_{\text{Back}} - Z(x-1,y)_{\text{Front}}) \\ &= \text{Bottom Right} + \text{Bottom Left}\end{aligned} \tag{9.1}$$

$$\text{Left Edge} = Z(x-1,y)_{\text{Back}} + Z(x-1,y-1)_{\text{Back}} - Z(x-1,y)_{\text{Front}} - Z(x-1,y-1)_{\text{Front}} \tag{9.2}$$

$$\text{Top Edge} = Z(x-1,y-1)_{\text{Back}} + Z(x,y-1)_{\text{Back}} - Z(x-1,y-1)_{\text{Front}} - Z(x,y-1)_{\text{Front}} \tag{9.3}$$

$$\text{Right Edge} = Z(x,y-1)_{\text{Back}} + Z(x,y)_{\text{Back}} - Z(x,y-1)_{\text{Front}} - Z(x,y)_{\text{Front}} \tag{9.4}$$

The centre comparison is as follows:

$$\begin{aligned}\text{Centre} &= Z(x,y)_{\text{Back}} + Z(x-1,y)_{\text{Back}} + Z(x,y-1)_{\text{Back}} + Z(x-1,y-1)_{\text{Back}} \\ &\quad - Z(x,y)_{\text{Front}} - Z(x-1,y)_{\text{Front}} - Z(x,y-1)_{\text{Front}} - Z(x-1,y-1)_{\text{Front}} \\ &= \text{Bottom Right} + \text{Bottom Left} + \text{Top Right} + \text{Top Left} \\ &= \text{Bottom Edge} + \text{Top Edge}\end{aligned} \tag{10}$$

These nine comparisons combine to form β by assigning a weight to each of the comparisons. If a comparison results in the Front pixel being closer at that point, then the appropriate weight is added to the β being collected. If the comparison results in the Front pixel being farther, then nothing is added. The weights were $\frac{1}{16}$ for the corner comparisons, $\frac{1}{8}$ for the edge comparisons, and $\frac{1}{4}$ for the center comparison. Using the nine weights developed above, the final equation for β is therefore:

$$\begin{aligned}\beta = (\text{Bottom Right} + \text{Bottom Left} + \text{Top Left} + \text{Top Right}) \times \frac{1}{16} \; + \\ (\text{Bottom Edge} + \text{Left Edge} + \text{Top Edge} + \text{Right Edge}) \times \frac{1}{8} \; + \; \text{Centre} \times \frac{1}{4}\end{aligned} \tag{11}$$

As the reader can readily see, the weights are all a fractional power of two, which means that the comparator sign bits can be directly added up with a simple adder circuit. Moreover, the calculation of the edge comparisons can be reduced to one addition since

the edge comparison can be expressed as the sum of two adjacent corner comparisons, as shown in equation 9.1. Equation 10 shows a similar reduction to two edge differences for the centre comparison.

8. β Hardware

Z must enter the Compositor eight bits at a time. This means that the low eight bits must enter and be compared first, since propagating carries from the low-order addition will affect the high-order addition. This defines the input order: Z_{low}, Z_{high}, α, Red, Green, Blue. For maximum throughput, it is important that these six data stream continuously from one Compositor to the next. Therefore, all data from one pixel should spend no more than six clock cycles in any one pipe stage inside the Compositor.

Each of the four corner comparisons can be done independently. If done in two's complement, the weight would be the inverse of the sign bit, shifted right the appropriate amount.

All the edge comparisons can take place at once. Each edge comparator gets input from two corner comparators. Conversely, the corner comparators send their results to two edge comparators. In this case the edge comparison uses a two's complement adder, which generates a weight from the sign bit in the same way that the corner subtractors do.

For the centre comparison, the same scheme is followed; the results from the previous comparison are added to form the final result. Note that the outputs of only two of the previous edge comparisons are needed, either ("top" + "bottom") or ("left" + "right"). This allows for optimization of the edge comparators whose outputs are unused. Again, the sign bit is used to denote the weight.

Each adder/subtractor has a D flip-flop to hold the carry-out of each compare. When the low byte is compared, the flip-flop holds the carry-in for the high-order compare. At the end of the high byte compare, the flip-flop holds the sign bit. Note that the sign bits from corner comparisons must go through two flip-flop delays, and the edge sign bits through one delay, since all the sign bits must arrive at the β adder at the same time.

Figure 5 shows the dataflow circuit for β. The thick lines indicate eight, nine or ten-bit buses, while the thin lines are one bit wide.

9. The Compositor Hardware

Now that β is ready, calculate the new **Z**, α, R, G, and B. Equation (1) has been algebraically simplified.

$$R_{comp} = (1 - \alpha_{Front} \times \beta) \times R_{Back} + (1 - \alpha_{Back} \times (1 - \beta)) \times R_{Front} \quad (1)$$

$$\alpha_{comp} = \alpha_{Back} + \alpha_{Front} - \alpha_{Back} \times \alpha_{Front} \quad (2)$$

$$Z_{comp} = \mathrm{Min}(Z_{Front}, Z_{Back}) \quad (3)$$

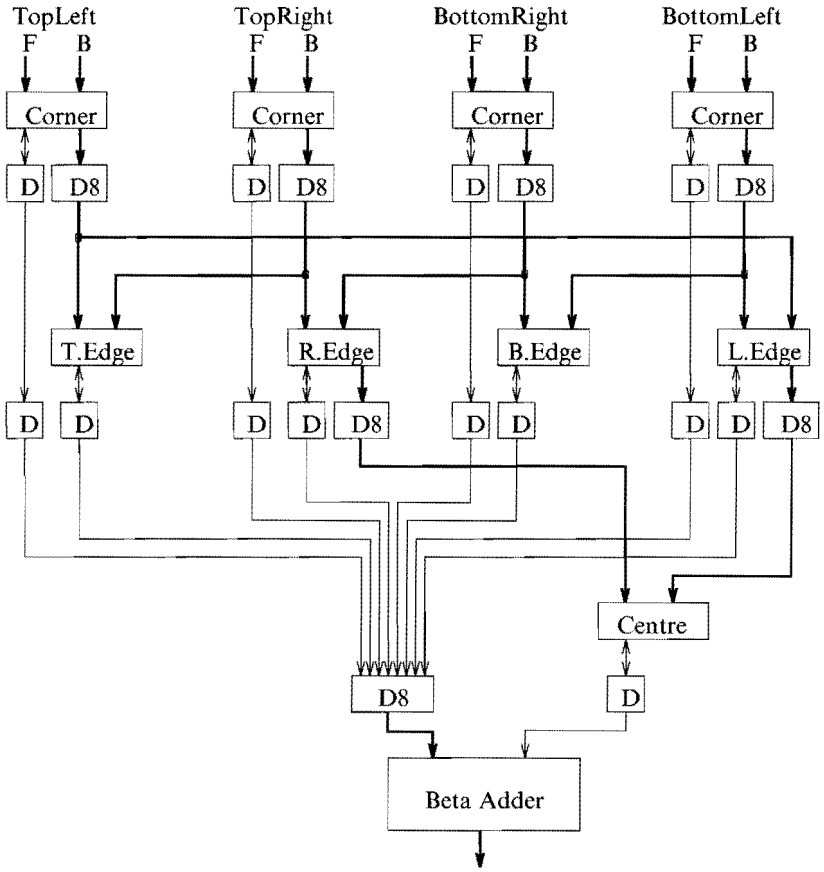

Figure 5: β Circuit.

To compute the new Z_{Min}, we must simply pick the minimum of the two new Z's. This information is available from the β calculation circuitry.

To compute the new α_{comp}, we must do an add and a multiply followed by a subtract. In this case, multiplication is of two nine-bit numbers with range [0.0 $\cdots$ 1.0]. α is eight bits wide outside the Compositor, using 11 11 11 11 as 1.0. When α is read into the chip, all values of α above 0.5 are augmented by 00 00 00 01. Similarly, all values above 0.5 are decreased by 00 00 00 01 when the new α is written. Thus 1.0 equals 1 00 00 00 00, with the binary point after the most significant digit.

To produce the new $(R, G, B)_{comp}$, the calculation occurs in two parts: For part one, $Factor_{Back} = (1-\alpha_{Front}\beta)$ and $Factor_{Front} = (1-\alpha_{Back}(1-\beta))$ is performed. This requires two multiplications and three subtracts. In this case, β is five bits wide in the range $[0.0 \cdots 1.0]$. Nominally, the multiplications are nine bits by five bits. The factors produced by this part are nine bits wide.

For part two of the $(R, G, B)_{comp}$ calculation, $Factor_{Back} \times R_{Back}$ and $Factor_{Front} \times R_{Front}$ must be performed (three pairs of multiplications). This is followed by three additions of the pairs of products. Obviously, the input to this second set of multiplications is supplied by the output of the first set of multiplications. The multiplications in this case are eight-bit by nine-bit, producing eight-bit products, which are then added to form three eight-bit sums, being the new R, G, and B.

Totaling the above operations, we have nine multiplications and eight adds/subtracts. Some of the subtractions are always from 1.0, so these can be optimized. However, we need two multipliers, since nine multiplications will take nine clock pulses, exceeding the upper limit of six pulses required for maximum throughput. A straightforward way of splitting up the work is to have one multiplier do $Factor_{Front} \times R_{Front}$, and the other do $Factor_{Back} \times R_{Back}$. Either one can do $\alpha_{Back} \times \alpha_{Front}$.

Figure 6 shows a box diagram of the dataflow part of the Compositor. The following paragraphs explain the function of each box in the figure.

There are four I/O subcircuits in Figure 6: *Input* is connected to FRONT_DATA and BACK_DATA. *Previous* is connected to PREV_FRONT and PREV_BACK. *D8* at the bottom of the figure is connected to OUT_DATA. *Save* indicates the store-back operation of the current **Z** into the previous row buffer, and is also connected to PREV_FRONT and PREV_BACK.

There are six major calculation subcircuits in Figure 6: *BETA* calculates β and selects Z_{Min}. *A+* takes an incoming α and augments it by one if $\alpha > 0.5$. *A-* does the opposite of *A+*. *Neg* does two's complement negation of its five-bit input. *Multiplier* takes either a pair of nine-bit factors or a five-bit and a nine-bit factor and multiplies them together. This circuit also contains an internal feedback path to calculate $Factor_{Back}$ and $Factor_{Front}$ as described above. *9-bit Add* adds two nine-bit numbers.

Lastly, the utility circuits are as follows: *Mux* is a multiplexor; it selects one of its data inputs and outputs it. *D8* is an eight-bit D-type flip-flop register, and *D9* is a nine-bit register. Both of these types load data at their inputs every clock cycle. The *D8S* and *D9S* circuits load when enabled. The *Z* circuit is a pair of *D8S*'s with the output of the first feeding the input of the second in a pipelined fashion.

With judicious pipeline scheduling, we can maintain a composition rate of one pixel every six clock cycles.

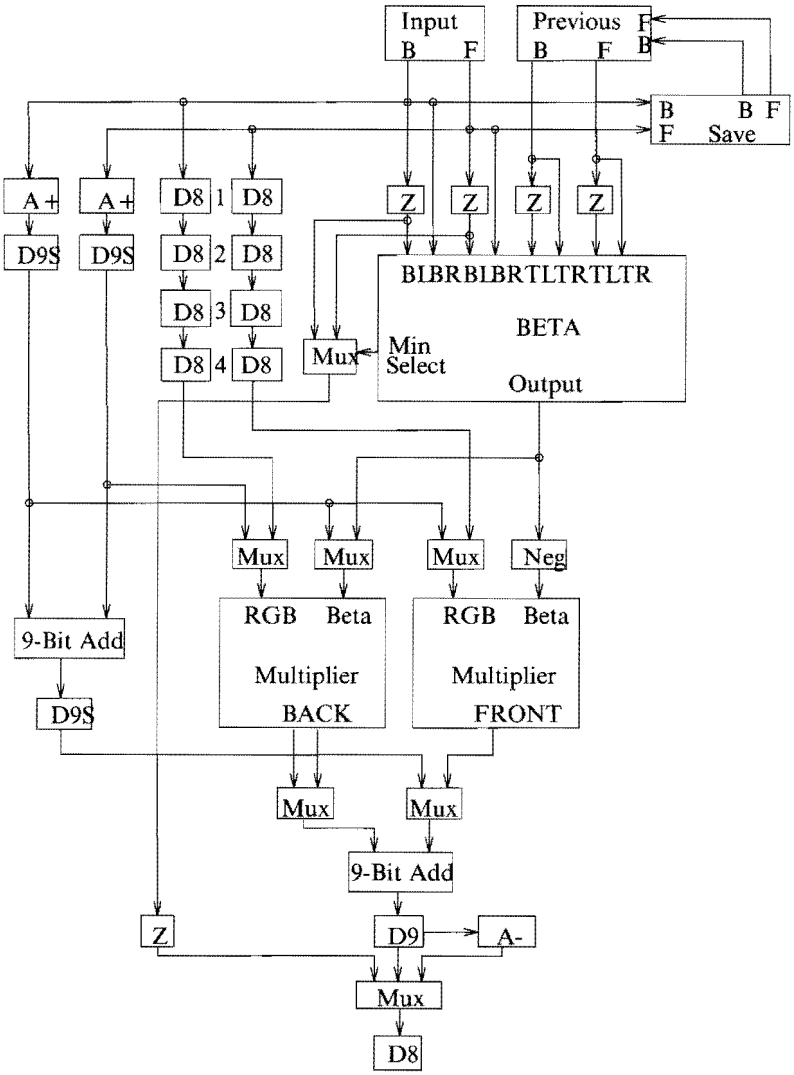

Figure 6: Data Flow Overview.

10. Compositor Performance

The design of the Compositor has been laid out and simulated using LSI Logic Inc.'s schematic entry system and hardware simulator. Simulations were performed using worst case temperature and voltage conditions. More importantly, simulations assumed that all non-primitive circuits were connected to each other using statistical wire lengths that are the average for a 6000-gate chip. The wire length restriction means that, for example, 9-bit adders have very long propagation delay caused by the "average length" wire between the eighth and ninth bits. This delay would not occur in a real layout due to the locality of all the gates in the adder.

Results show that the worst-case register-to-register propagation delay is 45.3nS, occurring at a 9-bit adder and carry feedback circuit in the β circuit. This implies a per-pixel throughput of one pixel every 271.8nS, or 3.679 megapixels per second. At a screen resolution of 512×512, this translates to a composition rate of about 14 frames per second. For a first attempt using old gate-array technology, this level of performance is clearly a lower limit on performance. With up-to-date fabrication technology, sensible layout, and better temperature and voltage conditions, we feel confident in predicting the existence of Compositors running at 30 frames per second and beyond.

We are currently in the process of building a Compositor chip.

11. Conclusion

We have described a computer graphics hardware architecture that utilizes a new paradigm of hardware parallelism. While it is clear that the paradigm of object-level parallelism is not yet well understood, we feel that the drawback of this approach has been eliminated. Namely, the combination of many rasters into one raster can be performed by our Compositor architecture at the rate of normal video refresh.

Future directions for research lie in two areas. The first area is the exploration of object-level parallelism for graphics modeling and production. One can forsee many problems that should be solved. The other area is for possible generalizations of the Compositor architecture.

References

1. Loren Carpenter, "The A-Buffer, an Antialiased Hidden Surface Method," *Computer Graphics (Proceedings of SIGGRAPH '84)*, vol. 18, no. 3, pp. 103-108, ACM SIGGRAPH, Minneapolis, Minn., July 1984.
2. Edwin C Catmull, "A Hidden-Surface Algorithm with Anti-Aliasing," *Computer Graphics (Proceedings of SIGGRAPH '78)*, vol. 12, no. 3, pp. 6-11, ACM SIGGRAPH, August 1978.

3. James Clark, "A VLSI Geometry Processor for Graphics," *Computer*, pp. 59-69, IEEE Computer Society, New York, July 1980.

4. James Clark, "The Geometry Engine: A VLSI Geometry System for Graphics," *Computer Graphics (Proceedings of SIGGRAPH '82)*, vol. 16, no. 3, pp. 127-133, ACM SIGGRAPH, Boston, Mass., July 1982.

5. Franklin C Crow, "A Comparison of Antialiasing Techniques," *IEEE Computer Graphics & Applications*, vol. 1, no. 1, pp. 40-48, IEEE Computer Society, New York, January 1981.

6. Tom Duff, "Compositing 3-D Rendered Images," *Computer Graphics (Proceedings of SIGGRAPH '85)*, vol. 19, no. 3, pp. 41-44, ACM SIGGRAPH, San Francisco, Calif., July 1985.

7. Eugene Fiume, Alain Fournier, and Larry Rudolph, "A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General-Purpose Ultracomputer," *Computer Graphics (Proceedings of SIGGRAPH '83)*, vol. 17, no. 3, pp. 141-150, ACM SIGGRAPH, Detroit, Mich, July 1983.

8. Donald Fussel and Bharat Deep Rathi, "A VLSI-Oriented Architecture for Real-Time Raster Display of Shaded Polygons," *Graphics Interface '82*, pp. 373-380, 1982.

9. Adam Levinthal and Thomas Porter, "Chap - A SIMD Graphics Processor," *Computer Graphics (Proceedings of SIGGRAPH '84)*, vol. 18, no. 3, pp. 77-82, ACM SIGGRAPH, Minneapolis, Minn., July 1984.

10. Thomas Porter and Tom Duff, "Compositing Digital Images," *Computer Graphics (Proceedings of SIGGRAPH '84)*, vol. 18, no. 3, pp. 253-259, ACM SIGGRAPH, Minneapolis, Minn, July 1984.

11. Christopher D Shaw, "The Image Composition Architecture: A Highly Parallel Graphics System," *University of Alberta Master's Thesis*, University of Alberta, Edmonton, Alberta, August 1988.

12. Roger W Swanson and Larry J Thayer, "A Fast Shaded-Polygon Renderer," *Computer Graphics (Proceedings of SIGGRAPH '86)*, vol. 20, no. 4, pp. 95-101, ACM SIGGRAPH, Dallas, Texas, August 1986.

13. Richard Weinberg, "Parallel Processing Image Synthesis and Anti-Aliasing," *Computer Graphics (Proceedings of SIGGRAPH '81)*, vol. 15, no. 3, pp. 55-61, ACM SIGGRAPH, Dallas, Texas, August 1981.